Internet Engineering Task Force (IETF)
Request for Comments: 6095
Category: Experimental
ISSN: 2070-1721

B. Linowski
TCS/Nokia Siemens Networks
M. Ersue
Nokia Siemens Networks
S. Kuryla
360 Treasury Systems

March 2011

Extending YANG with Language Abstractions

Abstract

YANG -- the Network Configuration Protocol (NETCONF) Data Modeling Language -- supports modeling of a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF. This memo suggests enhancing YANG with supplementary modeling features and language abstractions with the aim to improve the model extensibility and reuse.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6095.

Table of Contents

1. Introduction

YANG -- the NETCONF Data Modeling Language [RFC6020] -- supports modeling of a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF. This document defines extensions for the modeling language YANG as new language statements, which introduce language abstractions to improve the model extensibility and reuse. The document reports from modeling experience in the telecommunication industry and gives model examples from an actual network management system to highlight the value of proposed language extensions, especially class inheritance and recursiveness. The language extensions defined in this document have been implemented with two open source tools. These tools have been used to validate the model examples through the document. If this experimental specification results in successful usage, it is possible that the language extensions defined herein could be updated to incorporate implementation and deployment experience, then pursued on the Standards Track, possibly as part of a future version of YANG.

1.1. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.2. Motivation

Following are non-exhaustive motivation examples highlighting usage scenarios for language abstractions.

- Many systems today have a Management Information Base (MIB) that in effect is organized as a tree build of recursively nested container nodes. For example, the physical resources in the ENTITY-MIB conceptually form a containment tree. The index

entPhysicalContainedIn points to the containing entity in a flat list. The ability to represent nested, recursive data structures of arbitrary depth would enable the representation of the primary containment hierarchy of physical entities as a node tree in the server MIB and in the NETCONF payload.

- A manager scanning the network in order to update the state of an inventory management system might be only interested in data structures that represent a specific type of hardware. Such a manager would then look for entities that are of this specific type, including those that are an extension or specialization of this type. To support this use case, it is helpful to bear the corresponding type information within the data structures, which describe the network element hardware.

- A system that is managing network elements is concerned, e.g., with managed objects of type "plug-in modules" that have a name, a version, and an activation state. In this context, it is useful to define the "plug-in module" as a concept that is supposed to be further detailed and extended by additional concrete model elements. In order to realize such a system, it is worthwhile to model abstract entities, which enable reuse and ease concrete refinements of that abstract entity in a second step.

- As particular network elements have specific types of components that need to be managed (OS images, plug-in modules, equipment, etc.), it should be possible to define concrete types, which describe the managed object precisely. By using type-safe extensions of basic concepts, a system in the manager role can safely and explicitly determine that e.g., the "equipment" is actually of type "network card".

- Currently, different SDOs are working on the harmonization of their management information models. Often, a model mapping or transformation between systems becomes necessary. The harmonization of the models is done e.g., by mapping of the two models on the object level or integrating an object hierarchy into an existing information model. On the one hand, extending YANG with language abstractions can simplify the adoption of IETF resource models by other SDOs and facilitate the alignment with other SDOs' resource models (e.g., TM Forum SID [SID_V8]). On the other hand, the proposed YANG extensions can enable the utilization of the YANG modeling language in other SDOs, which usually model complex management systems in a top-down manner and use high-level language features frequently.

This memo specifies additional modeling features for the YANG language in the area of structured model abstractions, typed references, as well as recursive data structures, and it discusses how these new features can improve the modeling capabilities of YANG.

Section 1.5.1 contains a physical resource model that deals with some of the modeling challenges illustrated above. Section 1.5.2 gives an example that uses the base classes defined in the physical resource model and derives a model for physical entities defined in the Entity MIB.

1.3. Modeling Improvements with Language Abstractions

As an enhancement to YANG 1.0, complex types and typed instance identifiers provide different technical improvements on the modeling level:

- In case the model of a system that should be managed with NETCONF makes use of inheritance, complex types enable an almost one-to-one mapping between the classes in the original model and the YANG module.

- Typed instance identifiers allow representing associations between the concepts in a type-safe way to prevent type errors caused by referring to data nodes of incompatible types. This avoids referring to a particular location in the MIB. Referring to a particular location in the MIB is not mandated by the domain model.

- Complex types allow defining complete, self-contained type definitions. It is not necessary to explicitly add a key statement to lists, which use a grouping that defines the data nodes.

- Complex types simplify concept refinement by extending a base complex type and make it superfluous to represent concept refinements with workarounds such as huge choice-statements with complex branches.

- Abstract complex types ensure correct usage of abstract concepts by enforcing the refinement of a common set of properties before instantiation.

- Complex types allow defining recursive structures. This enables representing complex structures of arbitrary depth by nesting instances of basic complex types that may contain themselves.

- Complex types avoid introducing metadata types (e.g., type code enumerations) and metadata leafs (e.g., leafs containing a type code) to indicate which concrete type of object is actually represented by a generic container in the MIB. This also avoids explicitly ruling out illegal use of subtype-specific properties in generic containers.

- Complex type instances include the type information in the NETCONF payload. This allows determining the actual type of an instance during the NETCONF payload parsing and avoids the use in the model of additional leafs, which provide the type information as content.

- Complex types may be declared explicitly as optional features, which is not possible when the actual type of an entity represented by a generic container is indicated with a type code enumeration.

Appendix B, "Example YANG Module for the IPFIX/PSAMP Model", lists technical improvements for modeling with complex types and typed instance identifiers and exemplifies the usage of the proposed YANG extensions based on the IP Flow Information Export (IPFIX) / Packet Sampling (PSAMP) configuration model in [IPFIXCONF].

1.4. Design Approach

The proposed additional features for YANG in this memo are designed to reuse existing YANG statements whenever possible. Additional semantics is expressed by an extension that is supposed to be used as a substatement of an existing statement.

The proposed features don’t change the semantics of models that is valid with respect to the YANG specification [RFC6020].

1.5. Modeling Resource Models with YANG

1.5.1. Example of a Physical Network Resource Model

The diagram below depicts a portion of an information model for manageable network resources used in an actual network management system.

Note: The referenced model (UDM, Unified Data Model) is based on key resource modeling concepts from [SID_V8] and is compliant with selected parts of SID Resource Abstract Business Entities domain [UDM].

The class diagram in Figure 1 and the corresponding YANG module excerpt focus on basic resource ("Resource" and the distinction between logical and physical resources) and hardware abstractions ("Hardware", "Equipment", and "EquipmentHolder"). Class attributes were omitted to achieve decent readability.

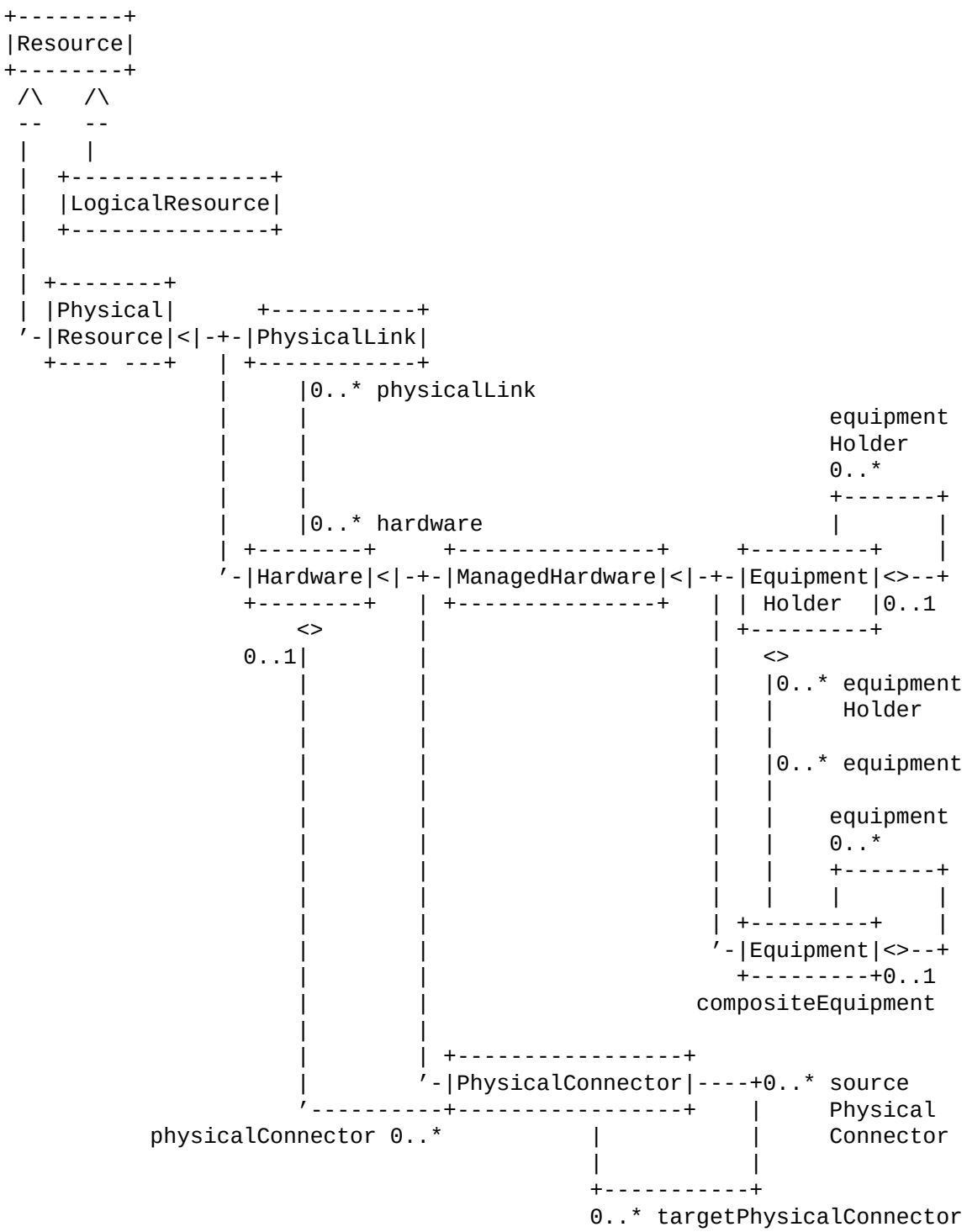

Figure 1: Physical Network Resource Model

Since this model is an abstraction of network-element-specific MIB topologies, modeling it with YANG creates some challenges. Some of these challenges and how they can be addressed with complex types are explained below:

- Modeling of abstract concepts: Classes like "Resource" represent concepts that primarily serve as a base class for derived classes. With complex types, such an abstract concept could be represented by an abstract complex type (see "complex-type extension statement" and "abstract extension statement").

- Class Inheritance: Information models for complex management domains often use class inheritance to create specialized classes like "PhysicalConnector" from a more generic base class (here, "Hardware"), which itself might inherit from another base class ("PhysicalResource"), etc. Complex types allow creating enhanced versions of an existing (abstract or concrete) base type via an extension (see "extends extension statement").

- Recursive containment: In order to specify containment hierarchies, models frequently contain different aggregation associations, in which the target (contained element) is either the containing class itself or a base class of the containing class. In the model above, the recursive containment of "EquipmentHolder" is an example of such a relationship (see the description for the "complex-type EquipmentHolder" in the example model "udmcore" below).

- Complex types support such a containment by using a complex type (or one of its ancestor types) as the type of an instance or instance list that is part of its definition (see "instance(-list) extension statement").

- Reference relationships: A key requirement on large models for network domains with many related managed objects is the ability to define inter-class associations that represent essential relationships between instances of such a class. For example, the relationship between "PhysicalLink" and "Hardware" tells which physical link is connecting which hardware resources. It is important to notice that this kind of relationship does not mandate any particular location of the two connected hardware instances in any MIB module. Such containment-agnostic relationships can be represented by a typed instance identifier that embodies one direction of such an association (see Section 3, "Typed Instance Identifier").

The YANG module excerpt below shows how the challenges listed above can be addressed by the Complex Types extension (module import prefix "ct:"). The complete YANG module for the physical resource model in Figure 1 can be found in Appendix A, "YANG Modules for Physical Network Resource Model and Hardware Entities Model".

Note: The YANG extensions proposed in this document have been implemented as the open source tools "Pyang Extension for Complex Types" [Pyang-ct], [Pyang], and "Libsmi Extension for Complex Types" [Libsmi]. All model examples in the document have been validated with the tools Pyang-ct and Libsmi.

```
<CODE BEGINS>

module udmcore {

   namespace "http://example.com/udmcore";
   prefix "udm";

   import ietf-complex-types {prefix "ct"; }

        // Basic complex types...

   ct:complex-type PhysicalResource {
      ct:extends Resource;
        ct:abstract true;
        // ...
        leaf serialNumber {
         type string;
         description "'Manufacturer-allocated part number' as
           defined in SID, e.g., the part number of a fiber link
           cable.";
      }
   }


   ct:complex-type Hardware {
      ct:extends PhysicalResource;
        ct:abstract true;
        // ...
       leaf-list physicalLink {
          type instance-identifier {ct:instance-type PhysicalLink;}
       }
       ct:instance-list containedHardware {
       ct:instance-type Hardware;
}
       ct:instance-list physicalConnector {
       ct:instance-type PhysicalConnector;
```

```
}
   }


   ct:complex-type PhysicalLink {
       ct:extends PhysicalResource;
       // ...
       leaf-list hardware {
          type instance-identifier {ct:instance-type Hardware;}
       }
   }


   ct:complex-type ManagedHardware {
      ct:extends Hardware;
        ct:abstract true;
        // ...
   }


   ct:complex-type PhysicalConnector {
      ct:extends Hardware;
        leaf location {type string;}
        // ...
      leaf-list sourcePhysicalConnector {
         type instance-identifier {ct:instance-type PhysicalConnector;}
      }
      leaf-list targetPhysicalConnector {
         type instance-identifier {ct:instance-type PhysicalConnector;}
      }
   }


   ct:complex-type Equipment {
      ct:extends ManagedHardware;
        // ...
      ct:instance-list equipment {
    ct:instance-type Equipment;
}
   }


   ct:complex-type EquipmentHolder {
      ct:extends ManagedHardware;
      description "In the SID V8 definition, this is a class based on
        the M.3100 specification.  A base class that represents physical
        objects that are both manageable as well as able to host,
        hold, or contain other physical objects.  Examples of physical
```

```
        objects that can be represented by instances of this object
        class are Racks, Chassis, Cards, and Slots.
        A piece of equipment with the primary purpose of containing
        other equipment.";
        leaf vendorName {type string;}
        // ...
      ct:instance-list equipment {
       ct:instance-type Equipment;
        }
      ct:instance-list equipmentHolder {
       ct:instance-type EquipmentHolder;
       }
   }
   // ...
}
```

<CODE ENDS>

1.5.2. Modeling Entity MIB Entries as Physical Resources

The physical resource module described above can now be used to model physical entities as defined in the Entity MIB [RFC4133]. For each physical entity class listed in the "PhysicalClass" enumeration, a complex type is defined. Each of these complex types extends the most specific complex type already available in the physical resource module. For example, the type "HWModule" extends the complex type "Equipment" as a hardware module. Physical entity properties that should be included in a physical entity complex type are combined in a grouping, which is then used in each complex type definition of an entity.

This approach has following benefits:

- The definition of the complex types for hardware entities becomes compact as many of the features can be reused from the basic complex type definition.

- Physical entities are modeled in a consistent manner as predefined concepts are extended.

- Entity-MIB-specific attributes as well as vendor-specific attributes can be added without having to define separate extension data nodes.

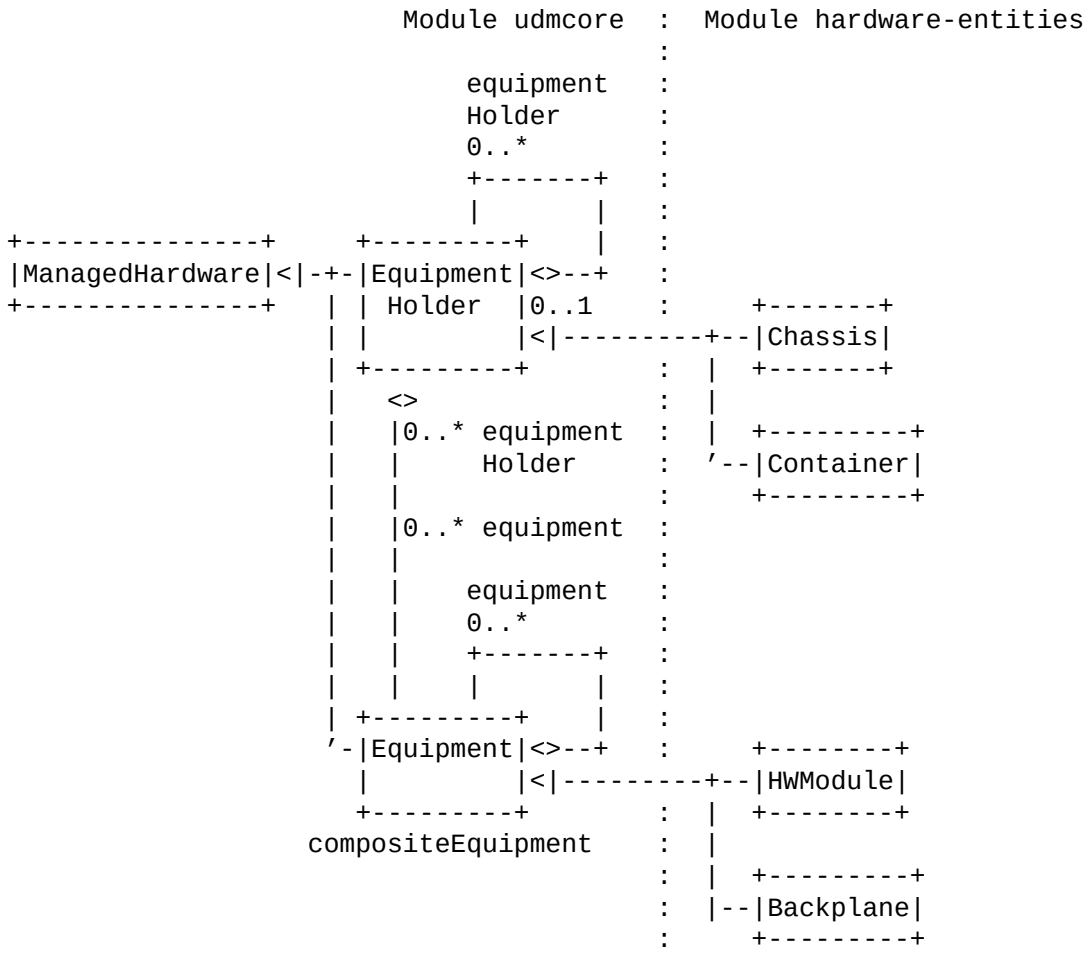

Figure 2: Hardware Entities Model

Below is an excerpt of the corresponding YANG module using complex types to model hardware entities. The complete YANG module for the Hardware Entities model in Figure 2 can be found in Appendix A, "YANG Modules for Physical Network Resource Model and Hardware Entities Model".

```
<CODE BEGINS>

module hardware-entities {


   namespace "http://example.com/hardware-entities";
   prefix "hwe";

   import ietf-yang-types {prefix "yt";}
   import ietf-complex-types {prefix "ct";}
   import udmcore {prefix "uc";}


   grouping PhysicalEntityProperties {
      // ...
      leaf mfgDate {type yang:date-and-time; }
      leaf-list uris {type string; }
   }


   // Physical entities representing equipment

   ct:complex-type HWModule {
      ct:extends uc:Equipment;
      description "Complex type representing module entries
                   (entPhysicalClass = module(9)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   // ...

   // Physical entities representing equipment holders

   ct:complex-type Chassis {
      ct:extends uc:EquipmentHolder;
      description "Complex type representing chassis entries
                   (entPhysicalClass = chassis(3)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   // ...
}

<CODE ENDS>
```

2. Complex Types

2.1. Definition

YANG type concept is currently restricted to simple types, e.g., restrictions of primitive types, enumerations, or union of simple types.

Complex types are types with a rich internal structure, which may be composed of substatements defined in Table 1 (e.g., lists, leafs, containers, choices). A new complex type may extend an existing complex type. This allows providing type-safe extensions to existing YANG models as instances of the new type.

Complex types have the following characteristics:

- Introduction of new types, as a named, formal description of a concrete manageable resource as well as abstract concepts.

- Types can be extended, i.e., new types can be defined by specializing existing types and adding new features. Instances of such an extended type can be used wherever instances of the base type may appear.

- The type information is made part of the NETCONF payload in case a derived type substitutes a base type. This enables easy and efficient consumption of payload elements representing complex type instances.

2.2. complex-type Extension Statement

The extension statement "complex-type" is introduced; it accepts an arbitrary number of statements that define node trees, among other common YANG statements ("YANG Statements", Section 7 of [RFC6020]).

substatement	cardinality
abstract	0..1
anyxml	0..n
choice	0..n
container	0..n
description	0..1
ct:instance	0..n
ct:instance-list	0..n
ct:extends	0..1
grouping	0..n
if-feature	0..n
key	0..1
leaf	0..n
leaf-list	0..n
list	0..n
must	0..n
ordered-by	0..n
reference	0..1
refine	0..n
status	0..1
typedef	0..n
uses	0..n

Table 1: complex-type's Substatements

Complex type definitions may appear at every place where a grouping may be defined. That includes the module, submodule, rpc, input, output, notification, container, and list statements.

Complex type names populate a distinct namespace. As with YANG groupings, it is possible to define a complex type and a data node (e.g., leaf, list, instance statements) with the same name in the same scope. All complex type names defined within a parent node or at the top level of the module or its submodules share the same type identifier namespace. This namespace is scoped to the parent node or module.

A complex type MAY have an instance key. An instance key is either defined with the "key" statement as part of the complex type or is inherited from the base complex type. It is not allowed to define an additional key if the base complex type or one of its ancestors already defines a key.

Complex type definitions do not create nodes in the schema tree.

2.3. instance Extension Statement

The "instance" extension statement is used to instantiate a complex type by creating a subtree in the management information node tree. The instance statement takes one argument that is the identifier of the complex type instance. It is followed by a block of substatements.

The type of the instance is specified with the mandatory "ct: instance-type" substatement. The type of an instance MUST be a complex type. Common YANG statements may be used as substatements of the "instance" statement. An instance is optional by default. To make an instance mandatory, "mandatory true" has to be applied as a substatement.

substatement	cardinality
description	0..1
config	0..1
ct:instance-type	1
if-feature	0..n
mandatory	0..1
must	0..n
reference	0..1
status	0..1
when	0..1
anyxml	0..n
choice	0..n
container	0..n
ct:instance	0..n
ct:instance-list	0..n
leaf	0..n
leaf-list	0..n
list	0..n

Table 2: instance's Substatements

The "instance" and "instance-list" extension statements (see Section 2.4, "instance-list Extension Statement") are similar to the existing "leaf" and "leaf-list" statements, with the exception that the content is composed of subordinate elements according to the instantiated complex type.

It is also possible to add additional data nodes by using the corresponding leaf, leaf-list, list, and choice-statements, etc., as substatements of the instance declaration. This is an in-place

augmentation of the used complex type confined to a complex type instantiation (see also Section 2.13, "Using Complex Types", for details on augmenting complex types).

2.4. instance-list Extension Statement

The "instance-list" extension statement is used to instantiate a complex type by defining a sequence of subtrees in the management information node tree. In addition, the "instance-list" statement takes one argument that is the identifier of the complex type instances. It is followed by a block of substatements.

The type of the instance is specified with the mandatory "ct: instance-type" substatement. In addition, it can be defined how often an instance may appear in the schema tree by using the "min-elements" and "max-elements" substatements. Common YANG statements may be used as substatements of the "instance-list" statement.

In analogy to the "instance" statement, YANG substatements like "list", "choice", "leaf", etc., MAY be used to augment the "instance-list" elements at the root level with additional data nodes.

substatementc	cardinality
description	0..1
config	0..1
ct:instance-type	1
if-feature	0..n
max-elements	0..1
min-elements	0..1
must	0..n
ordered-by	0..1
reference	0..1
status	0..1
when	0..1
anyxml	0..n
choice	0..n
container	0..n
ct:instance	0..n
ct:instance-list	0..n
leaf	0..n
leaf-list	0..n
list	0..n

Table 3: instance-list's Substatements

In case the instance list represents configuration data, the used complex type of an instance MUST have an instance key.

Instances as well as instance lists may appear as arguments of the "deviate" statement.

2.5. extends Extension Statement

A complex type MAY extend exactly one existing base complex type by using the "extends" extension statement. The keyword "extends" MAY occur as a substatement of the "complex-type" extension statement. The argument of the "complex-type" extension statement refers to the base complex type via its name. In case a complex type represents configuration data (the default), it MUST have a key; otherwise, it MAY have a key. A key is either defined with the "key" statement as part of the complex type or is inherited from the base complex type.

substatement	cardinality
description	0..1
reference	0..1
status	0..1

Table 4: extends' Substatements

2.6. abstract Extension Statement

Complex types may be declared to be abstract by using the "abstract" extension statement. An abstract complex type cannot be instantiated, meaning it cannot appear as the most specific type of an instance in the NETCONF payload. In case an abstract type extends a base type, the base complex type MUST be also abstract. By default, complex types are not abstract.

The abstract complex type serves only as a base type for derived concrete complex types and cannot be used as a type for an instance in the NETCONF payload.

The "abstract" extension statement takes a single string argument, which is either "true" or "false". In case a "complex-type" statement does not contain an "abstract" statement as a substatement, the default is "false". The "abstract" statement does not support any substatements.

2.7. XML Encoding Rules

An "instance" node is encoded as an XML element, where an "instance-list" node is encoded as a series of XML elements. The corresponding XML element names are the "instance" and "instance-list" identifiers, respectively, and they use the same XML namespace as the module.

Instance child nodes are encoded as subelements of the instance XML element. Subelements representing child nodes defined in the same complex type may appear in any order. However, child nodes of an extending complex type follow the child nodes of the extended complex type. As such, the XML encoding of lists is similar to the encoding of containers and lists in YANG.

Instance key nodes are encoded as subelements of the instance XML element. Instance key nodes must appear in the same order as they are defined within the "key" statement of the corresponding complex type definition and precede all other nodes defined in the same complex type. That is, if key nodes are defined in an extending complex type, XML elements representing key data precede all other XML elements representing child nodes. On the other hand, XML elements representing key data follow the XML elements representing data nodes of the base type.

The type of the actual complex type instance is encoded in a type element, which is put in front of all instance child elements, including key nodes, as described in Section 2.8 ("Type Encoding Rules").

The proposed XML encoding rules conform to the YANG XML encoding rules in [RFC6020]. Compared to YANG, enabling key definitions in derived hierarchies is a new feature introduced with the complex types extension. As a new language feature, complex types also introduce a new payload entry for the instance type identifier.

Based on our implementation experience, the proposed XML encoding rules support consistent mapping of YANG models with complex types to an XML schema using XML complex types.

2.8. Type Encoding Rules

In order to encode the type of an instance in the NETCONF payload, XML elements named "type" belonging to the XML namespace "urn:ietf:params:xml:ns:yang:ietf-complex-type-instance" are added to the serialized form of instance and instance-list nodes in the payload. The suggested namespace prefix is "cti". The "cti:type" XML elements are inserted before the serialized form of all members that have been declared in the corresponding complex type definition.

The "cti:type" element is inserted for each type in the extension chain to the actual type of the instance (most specific last). Each type name includes its corresponding namespace.

The type of a complex type instance MUST be encoded in the reply to NETCONF <get> and <get-config> operations, and in the payload of a NETCONF <edit-config> operation if the operation is "create" or "replace". The type of the instance MUST also be specified in case <copy-config> is used to export a configuration to a resource addressed with an URI. The type of the instance has to be specified in user-defined remote procedure calls (RPCs).

The type of the instance MAY be specified in case the operation is "merge" (either because this is explicitly specified or no operation attribute is provided).

In case the node already exists in the target configuration and the type attribute (type of a complex type instance) is specified but differs from the data in the target, an <rpc-error> element is returned with an <error-app-tag> value of "wrong-complex-type". In case no such element is present in the target configuration but the type attribute is missing in the configuration data, an <rpc-error> element is returned with an <error-tag> value of "missing-attribute".

The type MUST NOT be specified in case the operation is "delete".

2.9. Extension and Feature Definition Module

The module below contains all YANG extension definitions for complex types and typed instance identifiers. In addition, a "complex-type" feature is defined, which may be used to provide conditional or alternative modeling, depending on the support status of complex types in a NETCONF server. A NETCONF server that supports the modeling features for complex types and the XML encoding for complex types as defined in this document MUST advertise this as a feature. This is done by including the feature name "complex-types" in the feature parameter list as part of the NETCONF <hello> message as described in Section 5.6.4 in [RFC6020].

```
<CODE BEGINS> file "ietf-complex-types@2011-03-15.yang"

module ietf-complex-types {

   namespace "urn:ietf:params:xml:ns:yang:ietf-complex-types";
   prefix "ct";

   organization
```

```
      "NETMOD WG";

    contact
      "Editor:  Bernd Linowski
                <bernd.linowski.ext@nsn.com>
       Editor:  Mehmet Ersue
                <mehmet.ersue@nsn.com>
       Editor:  Siarhei Kuryla
                <s.kuryla@gmail.com>";

    description
       "YANG extensions for complex types and typed instance
       identifiers.

       Copyright (c) 2011 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject
       to the license terms contained in, the Simplified BSD License
       set forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (http://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC 6095; see
       the RFC itself for full legal notices.";

        revision 2011-03-15 {
            description "Initial revision.";
        }


        extension complex-type {
            description "Defines a complex-type.";
            reference "Section 2.2, complex-type Extension Statement";
            argument type-identifier {
                yin-element true;
            }
        }


        extension extends {
            description "Defines the base type of a complex-type.";
            reference "Section 2.5, extends Extension Statement";
            argument base-type-identifier {
                yin-element true;
            }
        }
```

```
        extension abstract {
            description "Makes the complex-type abstract.";
            reference "Section 2.6, abstract Extension Statement";
            argument status;
        }


        extension instance {
            description "Declares an instance of the given
                 complex type.";
            reference "Section 2.3, instance Extension Statement";
            argument ct-instance-identifier {
                yin-element true;
            }
        }

        extension instance-list {
            description "Declares a list of instances of the given
                 complex type";
            reference "Section 2.4, instance-list Extension Statement";
            argument ct-instance-identifier {
                yin-element true;
            }
        }

        extension instance-type {
            description "Tells to which type instance the instance
                         identifier refers.";
            reference "Section 3.2, instance-type Extension Statement";
            argument target-type-identifier {
                yin-element true;
            }
        }


        feature complex-types {
            description "Indicates that the server supports
                         complex types and instance identifiers.";
        }

    }

 <CODE ENDS>
```

2.10. Example Model for Complex Types

The example model below shows how complex types can be used to represent physical equipment in a vendor-independent, abstract way. It reuses the complex types defined in the physical resource model in Section 1.5.1.

```
<CODE BEGINS>

module hw {

   namespace "http://example.com/hw";
   prefix "hw";

   import ietf-complex-types {prefix "ct"; }
   import udmcore {prefix "uc"; }

   // Holder types

   ct:complex-type Slot {
           ct:extends uc:EquipmentHolder;
           leaf slotNumber { type uint16; config false; }
             // ...
   }

   ct:complex-type Chassis {
           ct:extends uc:EquipmentHolder;
           leaf numberOfChassisSlots { type uint32; config false; }
             // ..
   }

   // Equipment types

   ct:complex-type Card {
           ct:extends uc:Equipment;
           leaf position { type uint32; mandatory true; }
             leaf slotsRequired {type unit32; }
   }

   // Root Element
    ct:instance hardware { type uc:ManagedHardware; }

} // hw module

<CODE ENDS>
```

2.11. NETCONF Payload Example

Following example shows the payload of a reply to a NETCONF <get> command. The actual type of managed hardware instances is indicated with the "cti:type" elements as required by the type encoding rules. The containment hierarchy in the NETCONF XML payload reflects the containment hierarchy of hardware instances. This makes filtering based on the containment hierarchy possible without having to deal with values of leafs of type leafref that represent the tree structure in a flattened hierarchy.

```
<hardware>
    <cti:type>uc:BasicObject</cti:type>
    <distinguishedName>/R-T31/CH-2</distinguishedName>
    <globalId>6278279001</globalId>
    <cti:type>uc:Resource</cti:type>
    <cti:type>uc:PhysicalResource</cti:type>
    <otherIdentifier>Rack R322-1</otherIdentifier>
    <serialNumber>R-US-3276279a</serialNumber>
    <cti:type>uc:Hardware</cti:type>
    <cti:type>uc:ManagedHardware</cti:type>
    <cti:type>hw:EquipmentHolder</cti:type>
    <equipmentHolder>
        <cti:type>uc:BasicObject</cti:type>
        <distinguishedName>/R-T31/CH-2/SL-1</distinguishedName>
        <globalId>548872003</globalId>
        <cti:type>uc:Resource</cti:type>
        <cti:type>uc:PhysicalResource</cti:type>
        <otherIdentifier>CU-Slot</otherIdentifier>
        <serialNumber>T-K4733890x45</serialNumber>
        <cti:type>uc:Hardware</cti:type>
        <cti:type>uc:ManagedHardware</cti:type>
        <cti:type>uc:EquipmentHolder</cti:type>
        <equipment>
            <cti:type>uc:BasicObject</cti:type>
            <distinguishedName>/R-T31/CH-2/SL-1/C-3</distinguishedName>
            <globalId>89772001</globalId>
            <cti:type>uc:Resource</cti:type>
            <cti:type>uc:PhysicalResource</cti:type>
            <otherIdentifier>ATM-45252</otherIdentifier>
            <serialNumber>A-778911-b</serialNumber>
            <cti:type>uc:Hardware</cti:type>
            <cti:type>uc:ManagedHardware</cti:type>
            <cti:type>uc:Equipment</cti:type>
            <installed>true</installed>
            <version>A2</version>
            <redundancy>1</redundancy>
            <cti:type>hw:Card</cti:type>
```

```
            <usedSlots>1</usedSlots>
        </equipment>
        <cti:type>hw:Slot</cti:type>
        <slotNumber>1</slotNumber>
    </equipmentHolder>
    <cti:type>hw:Chassis</cti:type>
    <numberOfChassisSlots>6</numberOfChassisSlots>
    // ...
</hardware>
```

2.12. Update Rules for Modules Using Complex Types

In addition to the module update rules specified in Section 10 in [RFC6020], modules that define complex types, instances of complex types, and typed instance identifiers must obey following rules:

- New complex types MAY be added.
- A new complex type MAY extend an existing complex type.
- New data definition statements MAY be added to a complex type only if:
  - they are not mandatory or
  - they are not conditionally dependent on a new feature (i.e., they do not have an "if-feature" statement that refers to a new feature).
- The type referred to by the instance-type statement may be changed to a type that derives from the original type only if the original type does not represent configuration data.

2.13. Using Complex Types

All data nodes defined inside a complex type reside in the complex type namespace, which is their parent node namespace.

2.13.1. Overriding Complex Type Data Nodes

It is not allowed to override a data node inherited from a base type. That is, it is an error if a type "base" with a leaf named "foo" is extended by another complex type ("derived") with a leaf named "foo" in the same module. In case they are derived in different modules, there are two distinct "foo" nodes that are mapped to the XML namespaces of the module, where the complex types are specified.

A complex type that extends a basic complex type may use the "refine" statement in order to improve an inherited data node. The target node identifier must be qualified by the module prefix to indicate clearly which inherited node is refined.

The following refinements can be done:

- A leaf or choice node may have a default value, or a new default value if it already had one.
- Any node may have a different "description" or "reference" string.
- A leaf, anyxml, or choice node may have a "mandatory true" statement. However, it is not allowed to change from "mandatory true" to "mandatory false".
- A leaf, leaf-list, list, container, or anyxml node may have additional "must" expressions.
- A list, leaf-list, instance, or instance-list node may have a "min-elements" statement, if the base type does not have one or does not have one with a value that is greater than the minimum value of the base type.
- A list, leaf-list, instance, or instance-list node may have a "max-elements" statement, if the base type does not have one or does not have one with a value that is smaller than the maximum value of the base type.

It is not allowed to refine complex-type nodes inside "instance" or "instance-list" statements.

2.13.2. Augmenting Complex Types

Augmenting complex types is only allowed if a complex type is instantiated in an "instance" or "instance-list" statement. This confines the effect of the augmentation to the location in the schema tree where the augmentation is done. The argument of the "augment" statement MUST be in the descendant form (as defined by the rule "descendant-schema-nodeid" in Section 12 in [RFC6020]).

```
ct:complex-type Chassis {
        ct:extends EquipmentHolder;
        container chassisInfo {
               config false;
               leaf numberOfSlots { type uint16; }
               leaf occupiedSlots { type uint16; }
               leaf height {type unit16;}
               leaf width {type unit16;}
          }
}

ct:instance-list chassis {
        type Chassis;
        augment "chassisInfo" {
               leaf modelId { type string; }
        }
}
```

When augmenting a complex type, only the "container", "leaf", "list", "leaf-list", "choice", "instance", "instance-list", and "if-feature" statements may be used within the "augment" statement. The nodes added by the augmentation MUST NOT be mandatory nodes. One or many "augment" statements may not cause the creation of multiple nodes with the same name from the same namespace in the target node.

To achieve less-complex modeling, this document proposes the augmentation of complex type instances without recursion.

2.13.3. Controlling the Use of Complex Types

A server might not want to support all complex types defined in a supported module. This issue can be addressed with YANG features as follows:

- Features are defined that are used inside complex type definitions (by using "if-feature" as a substatement) to make them optional. In this case, such complex types may only be instantiated if the feature is supported (advertised as a capability in the NETCONF <hello> message).

- The "deviation" statement may be applied to node trees, which are created by "instance" and "instance-list" statements. In this case, only the substatement "deviate not-supported" is allowed.

- It is not allowed to apply the "deviation" statement to node tree elements that may occur because of the recursive use of a complex type. Other forms of deviations ("deviate add", "deviate replace", "deviate delete") are NOT supported inside node trees spanned by "instance" or "instance-list".

As complex type definitions do not contribute by themselves to the data node tree, data node declarations inside complex types cannot be the target of deviations.

In the example below, client applications are informed that the leaf "occupiedSlots" is not supported in the top-level chassis. However, if a chassis contains another chassis, the contained chassis may support the leaf that reports the number of occupied slots.

```
deviation "/chassis/chassisSpec/occupiedSlots" {
   deviate not-supported;
}
```

3. Typed Instance Identifier

3.1. Definition

Typed instance identifier relationships are an addition to the relationship types already defined in YANG, where the leafref relationship is location dependent, and the instance-identifier does not specify to which type of instances the identifier points.

A typed instance identifier represents a reference to an instance of a complex type without being restricted to a particular location in the containment tree. This is done by using the extension statement "instance-type" as a substatement of the existing "type instance identifier" statement.

Typed instance identifiers allow referring to instances of complex types that may be located anywhere in the schema tree. The "type" statement plays the role of a restriction that must be fulfilled by the target node, which is referred to with the instance identifier. The target node MUST be of a particular complex type, either the type itself or any type that extends this complex type.

3.2. instance-type Extension Statement

The "instance-type" extension statement specifies the complex type of the instance to which the instance-identifier refers. The referred instance may also instantiate any complex type that extends the specified complex type.

The instance complex type is identified by the single name argument. The referred complex type MUST have a key. This extension statement MUST be used as a substatement of the "type instance-identifier" statement. The "instance-type" extension statement does not support any substatements.

3.3. Typed Instance Identifier Example

In the example below, a physical link connects an arbitrary number of physical ports. Here, typed instance identifiers are used to denote which "PhysicalPort" instances (anywhere in the data tree) are connected by a "PhysicalLink".

```
// Extended version of type Card
ct:complex-type Card {
   ct:extends Equipment;
   leaf usedSlot { type uint16; mandatory true; }
   ct:instance-list port {
       type PhysicalPort;
   }
}

ct:complex-type PhysicalPort {
   ct:extends ManagedHardware;
   leaf portNumber { type int32; mandatory true; }
}

ct:complex-type PhysicalLink {
   ct:extends ManagedHardware;
   leaf media { type string; }
   leaf-list connectedPort {
      type instance-identifier {
        ct:instance-type PhysicalPort;
      }
      min-elements 2;
   }
}
```

Below is the XML encoding of an element named "link" of type "PhysicalLink":

```
      <link>
          <objectId>FTCL-771</objectId>
          <media>Fiber</media>
          <connectedPort>/hw:hardware[objectId='R-11']
            /hw:equipment[objectId='AT22']/hw:port[objectId='P12']
          </connectedPort>
          <connectedPort>/hw:hardware[objectId='R-42]
            /hw:equipment[objectId='AT30']/hw:port[objectId='P3']
          </connectedPort>
          <serialNumber>F-7786828</serialNumber>
          <commonName>FibCon 7</commonName>
      </link>
```

4. IANA Considerations

This document registers two URIs in the IETF XML registry. IANA registered the following URIs, according to [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-complex-types
URI: urn:ietf:params:xml:ns:yang:ietf-complex-type-instance

Registrant Contact:
Bernd Linowski (bernd.linowski.ext@nsn.com)
Mehmet Ersue (mehmet.ersue@nsn.com)
Siarhei Kuryla (s.kuryla@gmail.com)

XML: N/A, the requested URIs are XML namespaces.

This document registers one module name in the "YANG Module Names" registry, defined in [RFC6020].

name: ietf-complex-types

namespace: urn:ietf:params:xml:ns:yang:ietf-complex-types

prefix: ct

RFC: 6095

5. Security Considerations

The YANG module "complex-types" in this memo defines YANG extensions for complex types and typed instance identifiers as new language statements.

Complex types and typed instance identifiers themselves do not have any security impact on the Internet.

The security considerations described throughout [RFC6020] apply here as well.

6. Acknowledgements

The authors would like to thank to Martin Bjorklund, Balazs Lengyel, Gerhard Muenz, Dan Romascanu, Juergen Schoenwaelder, and Martin Storch for their valuable review and comments on different versions of the document.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

7.2. Informative References

[IPFIXCONF] Muenz, G., Claise, B., and P. Aitken, "Configuration Data Model for IPFIX and PSAMP", Work in Progress, March 2011.

[Libsmi] Kuryla, S., "Libsmi Extension for Complex Types", April 2010, <http://www.ibr.cs.tu-bs.de/svn/libsmi>.

[Pyang] Bjorklund, M., "An extensible YANG validator and converter in python", October 2010, <http://code.google.com/p/pyang/>.

[Pyang-ct] Kuryla, S., "Complex type extension for an extensible YANG validator and converter in python", April 2010, <http://code.google.com/p/pyang-ct/>.

[RFC4133] Bierman, A. and K. McCloghrie, "Entity MIB (Version 3)", RFC 4133, August 2005.

[SID_V8] TeleManagement Forum, "GB922, Information Framework (SID) Solution Suite, Release 8.0", July 2008, <http://www.tmforum.org/DocumentsInformation/GB922InformationFramework/35499/article.html>.

[UDM] NSN, "Unified Data Model SID Compliance Statement", May 2010, <http://www.tmforum.org/InformationFramework/NokiaSiemensNetworks/8815/home.html>.

Appendix A. YANG Modules for Physical Network Resource Model and Hardware Entities Model

YANG module for the 'Physical Network Resource Model':

<CODE BEGINS>

```
module udmcore {

   namespace "http://example.com/udmcore";
   prefix "udm";

   import ietf-yang-types {prefix "yang";}
   import ietf-complex-types {prefix "ct";}


   ct:complex-type BasicObject {
      ct:abstract true;
      key "distinguishedName";
        leaf globalId {type int64;}
        leaf distinguishedName {type string; mandatory true;}
   }


   ct:complex-type ManagedObject {
      ct:extends BasicObject;
      ct:abstract true;
      leaf instance {type string;}
      leaf objectState {type int32;}
      leaf release {type string;}
   }

   ct:complex-type Resource {
            ct:extends ManagedObject;
            ct:abstract true;
      leaf usageState {type int16;}
      leaf managementMethodSupported {type string;}
      leaf managementMethodCurrent {type string;}
      leaf managementInfo {type string;}
      leaf managementDomain {type string;}
      leaf version {type string;}
      leaf entityIdentification {type string;}
      leaf description {type string;}
      leaf rootEntityType {type string;}
   }
```

```
ct:complex-type LogicalResource {
   ct:extends Resource;
   ct:abstract true;
   leaf lrStatus {type int32;}
   leaf serviceState {type int32;}
   leaf isOperational {type boolean;}
}


ct:complex-type PhysicalResource {
   ct:extends Resource;
   ct:abstract true;
   leaf manufactureDate {type string;}
   leaf otherIdentifier {type string;}
   leaf powerState {type int32;}
   leaf serialNumber {type string;}
   leaf versionNumber {type string;}
}


ct:complex-type Hardware {
   ct:extends PhysicalResource;
   ct:abstract true;
   leaf width {type string;}
   leaf height {type string;}
   leaf depth {type string;}
   leaf measurementUnits {type int32;}
   leaf weight {type string;}
   leaf weightUnits {type int32;}
   leaf-list physicalLink {
      type instance-identifier {
         ct:instance-type PhysicalLink;
      }
   }
   ct:instance-list containedHardware {
      ct:instance-type Hardware;
   }
   ct:instance-list physicalConnector {
      ct:instance-type PhysicalConnector;
   }
}


ct:complex-type PhysicalLink {
   ct:extends PhysicalResource;
   leaf isWireless {type boolean;}
   leaf currentLength {type string;}
   leaf maximumLength {type string;}
```

```
         leaf mediaType {type int32;}
         leaf-list hardware {
            type instance-identifier {
               ct:instance-type Hardware;
            }
         }
      }


      ct:complex-type ManagedHardware {
         ct:extends Hardware;
         leaf additionalinfo {type string;}
         leaf physicalAlarmReportingEnabled {type boolean;}
         leaf pyhsicalAlarmStatus {type int32;}
         leaf coolingRequirements {type string;}
         leaf hardwarePurpose {type string;}
         leaf isPhysicalContainer {type boolean;}
      }


      ct:complex-type AuxiliaryComponent {
         ct:extends ManagedHardware;
         ct:abstract true;
      }


      ct:complex-type PhysicalPort {
         ct:extends ManagedHardware;
         leaf portNumber {type int32;}
         leaf duplexMode {type int32;}
         leaf ifType {type int32;}
         leaf vendorPortName {type string;}
      }


      ct:complex-type PhysicalConnector {
         ct:extends Hardware;
         leaf location {type string;}
         leaf cableType {type int32;}
         leaf gender {type int32;}
         leaf inUse {type boolean;}
         leaf pinDescription {type string;}
         leaf typeOfConnector {type int32;}
         leaf-list sourcePhysicalConnector {
            type instance-identifier {
               ct:instance-type PhysicalConnector;
            }
         }
```

```
         leaf-list targetPhysicalConnector {
            type instance-identifier {
               ct:instance-type PhysicalConnector;
            }
         }
      }


      ct:complex-type Equipment {
         ct:extends ManagedHardware;
         leaf installStatus {type int32;}
         leaf expectedEquipmentType {type string;}
         leaf installedEquipmentType {type string;}
         leaf installedVersion {type string;}
         leaf redundancy {type int32;}
         leaf vendorName {type string;}
         leaf dateOfLastService {type yang:date-and-time;}
         leaf interchangeability {type string;}
         leaf identificationCode {type string;}
         ct:instance-list equipment {
            ct:instance-type Equipment;
         }
      }


      ct:complex-type EquipmentHolder {
         ct:extends ManagedHardware;
         leaf vendorName {type string;}
         leaf locationName {type string;}
         leaf dateOfLastService {type yang:date-and-time;}
         leaf partNumber {type string;}
         leaf availabilityStatus {type int16;}
         leaf nameFromPlanningSystem {type string;}
         leaf modelNumber {type string;}
         leaf acceptableEquipmentList {type string;}
         leaf isSolitaryHolder {type boolean;}
         leaf holderStatus {type int16;}
         leaf interchangeability {type string;}
         leaf equipmentHolderSpecificType {type string; }
         leaf position {type string;}
         leaf atomicCompositeType {type int16;}
         leaf uniquePhysical {type boolean;}
         leaf physicalDescription {type string;}
         leaf serviceApproach {type string;}
         leaf mountingOptions {type int32;}
         leaf cableManagementStrategy {type string;}
         leaf isSecureHolder {type boolean;}
         ct:instance-list equipment {
```

```
            ct:instance-type Equipment;
               }
         ct:instance-list equipmentHolder {
            ct:instance-type EquipmentHolder;
         }
      }

      // ... other resource complex types ...

   }
   <CODE ENDS>
```

YANG module for the 'Hardware Entities Model':

```
   <CODE BEGINS>

   module hardware-entities {

      namespace "http://example.com/:hardware-entities";
      prefix "hwe";

      import ietf-yang-types {prefix "yang";}
      import ietf-complex-types {prefix "ct";}
      import udmcore {prefix "uc";}


      grouping PhysicalEntityProperties {
         leaf hardwareRev {type string; }
         leaf firmwareRev {type string; }
         leaf softwareRev {type string; }
         leaf serialNum {type string; }

         leaf mfgName {type string; }
         leaf modelName {type string; }
         leaf alias {type string; }
         leaf ssetID{type string; }
         leaf isFRU {type boolean; }
         leaf mfgDate {type yang:date-and-time; }
         leaf-list uris {type string; }
      }


      // Physical entities representing equipment

      ct:complex-type Module {
         ct:extends uc:Equipment;
         description "Complex type representing module entries
```

```
         (entPhysicalClass = module(9)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   ct:complex-type Backplane {
      ct:extends uc:Equipment;
      description "Complex type representing backplane entries
         (entPhysicalClass = backplane(4)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }


   // Physical entities representing auxiliary hardware components

   ct:complex-type PowerSupply {
      ct:extends uc:AuxiliaryComponent;
      description "Complex type representing power supply entries
         (entPhysicalClass = powerSupply(6)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   ct:complex-type Fan {
      ct:extends uc:AuxiliaryComponent;
      description "Complex type representing fan entries
         (entPhysicalClass = fan(7)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   ct:complex-type Sensor {
      ct:extends uc:AuxiliaryComponent;
      description "Complex type representing sensor entries
         (entPhysicalClass = sensor(8)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }


   // Physical entities representing equipment holders

   ct:complex-type Chassis {
      ct:extends uc:EquipmentHolder;
      description "Complex type representing chassis entries
         (entPhysicalClass = chassis(3)) in entPhysicalTable";
      uses PhysicalEntityProperties;
   }

   ct:complex-type Container {
      ct:extends uc:EquipmentHolder;
      description "Complex type representing container entries
```

```
            (entPhysicalClass = container(5)) in entPhysicalTable";
         uses PhysicalEntityProperties;
      }

      ct:complex-type Stack {
         ct:extends uc:EquipmentHolder;
         description "Complex type representing stack entries
            (entPhysicalClass = stack(11)) in entPhysicalTable";
         uses PhysicalEntityProperties;
      }


      // Other kinds of physical entities

      ct:complex-type Port {
         ct:extends uc:PhysicalPort;
         description "Complex type representing port entries
            (entPhysicalClass = port(10)) in entPhysicalTable";
         uses PhysicalEntityProperties;
      }

      ct:complex-type CPU {
         ct:extends uc:Hardware;
         description "Complex type representing cpu entries
            (entPhysicalClass = cpu(12)) in entPhysicalTable";
         uses PhysicalEntityProperties;
      }

   }
   <CODE ENDS>
```

Appendix B. Example YANG Module for the IPFIX/PSAMP Model

B.1. Modeling Improvements for the IPFIX/PSAMP Model with Complex Types and Typed Instance Identifiers

The module below is a variation of the IPFIX/PSAMP configuration model, which uses complex types and typed instance identifiers to model the concept outlined in [IPFIXCONF].

When looking at the YANG module with complex types and typed instance identifiers, various technical improvements on the modeling level become apparent.

- There is almost a one-to-one mapping between the domain concepts introduced in IPFIX and the complex types in the YANG module.

- All associations between the concepts (besides containment) are represented with typed identifiers. That avoids having to refer to a particular location in the tree. Referring to a particular in the tree is not mandated by the original model.

- It is superfluous to represent concept refinement (class inheritance in the original model) with containment in the form of quite big choice-statements with complex branches. Instead, concept refinement is realized by complex types extending a base complex type.

- It is unnecessary to introduce metadata identities and leafs (e.g., "identity cacheMode" and "leaf cacheMode" in "grouping cacheParameters") that just serve the purpose of indicating which concrete subtype of a generic type (modeled as grouping, which contains the union of all features of all subtypes) is actually represented in the MIB.

- Ruling out illegal use of subtype-specific properties (e.g., "leaf maxFlows") by using "when" statements that refer to a subtype discriminator is not necessary (e.g., when "../cacheMode != 'immediate'").

- Defining properties like the configuration status wherever a so called "parameter grouping" is used is not necessary. Instead, those definitions can be put inside the complex type definition itself.

- Separating the declaration of the key from the related data nodes definitions in a grouping (see use of "grouping selectorParameters") can be avoided.

- Complex types may be declared as optional features. If the type is indicated with an identity (e.g., "identity immediate"), this is not possible, since "if-feature" is not allowed as a substatement of "identity".

B.2. IPFIX/PSAMP Model with Complex Types and Typed Instance Identifiers

```
<CODE BEGINS>
module ct-ipfix-psamp-example {
     namespace "http://example.com/ns/ct-ipfix-psamp-example";
     prefix ipfix;

     import ietf-yang-types { prefix yang; }
     import ietf-inet-types { prefix inet; }
     import ietf-complex-types {prefix "ct"; }
```

```
    description "Example IPFIX/PSAMP Configuration Data Model
      with complex types and typed instance identifiers";

    revision 2011-03-15 {
       description "The YANG Module ('YANG Module of the IPFIX/PSAMP
         Configuration Data Model') in [IPFIXCONF] modeled with
         complex types and typed instance identifiers.
         Disclaimer: This example model illustrates the use of the
         language extensions defined in this document and does not
         claim to be an exact reproduction of the original YANG
         model referred above. The original description texts have
         been shortened to increase the readability of the model
         example.";
   }

    /*****************************************************************
    * Features
    *****************************************************************/

    feature exporter {
      description "If supported, the Monitoring Device can be used as
        an Exporter.  Exporting Processes can be configured.";
    }

    feature collector {
      description "If supported, the Monitoring Device can be used as
        a Collector.  Collecting Processes can be configured.";
    }

    feature meter {
      description "If supported, Observation Points, Selection
        Processes, and Caches can be configured.";
    }

    feature psampSampCountBased {
      description "If supported, the Monitoring Device supports
        count-based Sampling...";
    }

    feature psampSampTimeBased {
      description "If supported, the Monitoring Device supports
        time-based Sampling...";
    }

    feature psampSampRandOutOfN {
      description "If supported, the Monitoring Device supports
        random n-out-of-N Sampling...";
    }
```

```
  feature psampSampUniProb {
    description "If supported, the Monitoring Device supports
      uniform probabilistic Sampling...";
  }

  feature psampFilterMatch {
    description "If supported, the Monitoring Device supports
      property match Filtering...";
  }

  feature psampFilterHash {
    description "If supported, the Monitoring Device supports
      hash-based Filtering...";
  }

  feature cacheModeImmediate {
    description "If supported, the Monitoring Device supports
      Cache Mode 'immediate'.";
  }

  feature cacheModeTimeout {
    description "If supported, the Monitoring Device supports
      Cache Mode 'timeout'.";
  }

  feature cacheModeNatural {
    description "If supported, the Monitoring Device supports
      Cache Mode 'natural'.";
  }

  feature cacheModePermanent {
    description "If supported, the Monitoring Device supports
      Cache Mode 'permanent'.";
  }

  feature udpTransport {
    description "If supported, the Monitoring Device supports UDP
      as transport protocol.";
  }

  feature tcpTransport {
    description "If supported, the Monitoring Device supports TCP
      as transport protocol.";
  }

  feature fileReader {
    description "If supported, the Monitoring Device supports the
      configuration of Collecting Processes as File Readers.";
```

```
  }

  feature fileWriter {
    description "If supported, the Monitoring Device supports the
      configuration of Exporting Processes as File Writers.";
  }

  /****************************************************************
  * Identities
  ****************************************************************/

  /*** Hash function identities ***/
  identity hashFunction {
    description "Base identity for all hash functions...";
  }
  identity BOB {
    base "hashFunction";
    description "BOB hash function";
    reference "RFC 5475, Section 6.2.4.1.";
  }
  identity IPSX {
    base "hashFunction";
    description "IPSX hash function";
    reference "RFC 5475, Section 6.2.4.1.";
  }
  identity CRC {
    base "hashFunction";
    description "CRC hash function";
    reference "RFC 5475, Section 6.2.4.1.";
  }

  /*** Export mode identities ***/
  identity exportMode {
    description "Base identity for different usages of export
      destinations configured for an Exporting Process...";
  }
  identity parallel {
    base "exportMode";
    description "Parallel export of Data Records to all
      destinations configured for the Exporting Process.";
  }
  identity loadBalancing {
    base "exportMode";
    description "Load-balancing between the different
      destinations...";
  }
  identity fallback {
    base "exportMode";
```

```
      description "Export to the primary destination...";
    }

    /*** Options type identities ***/
    identity optionsType {
      description "Base identity for report types exported
         with options...";
    }
    identity meteringStatistics {
      base "optionsType";
      description "Metering Process Statistics.";
      reference "RFC 5101, Section 4.1.";
    }
    identity meteringReliability {
      base "optionsType";
      description "Metering Process Reliability Statistics.";
      reference "RFC 5101, Section 4.2.";
    }
    identity exportingReliability {
      base "optionsType";
      description "Exporting Process Reliability
        Statistics.";
      reference "RFC 5101, Section 4.3.";
    }
    identity flowKeys {
      base "optionsType";
      description "Flow Keys.";
      reference "RFC 5101, Section 4.4.";
    }
    identity selectionSequence {
      base "optionsType";
      description "Selection Sequence and Selector Reports.";
      reference "RFC 5476, Sections 6.5.1 and 6.5.2.";
    }
    identity selectionStatistics {
      base "optionsType";
      description "Selection Sequence Statistics Report.";
      reference "RFC 5476, Sections 6.5.3.";
    }
    identity accuracy {
      base "optionsType";
      description "Accuracy Report.";
      reference "RFC 5476, Section 6.5.4.";
    }
    identity reducingRedundancy {
      base "optionsType";
      description "Enables the utilization of Options Templates to
        reduce redundancy in the exported Data Records.";
```

```
      reference "RFC 5473.";
    }
    identity extendedTypeInformation {
      base "optionsType";
      description "Export of extended type information for
        enterprise-specific Information Elements used in the
        exported Templates.";
      reference "RFC 5610.";
    }

    /*****************************************************************
    * Type definitions
    *****************************************************************/

    typedef nameType {
      type string {
        length "1..max";
        pattern "\S(.*\S)?";
      }
      description "Type for 'name' leafs...";
    }

    typedef direction {
      type enumeration {
        enum ingress {
          description "This value is used for monitoring incoming
            packets.";
        }
        enum egress {
          description "This value is used for monitoring outgoing
            packets.";
        }
        enum both {
          description "This value is used for monitoring incoming and
            outgoing packets.";
        }
      }
      description "Direction of packets going through an interface or
        linecard.";
    }

    typedef transportSessionStatus {
      type enumeration {
        enum inactive {
          description "This value MUST be used for...";
        }
        enum active {
          description "This value MUST be used for...";
```

```
        }
        enum unknown {
          description "This value MUST be used if the status...";
        }
      }
      description "Status of a Transport Session.";
      reference "RFC 5815, Section 8 (ipfixTransportSessionStatus).";
    }

    /*****************************************************************
    * Complex types
    *****************************************************************/

    ct:complex-type ObservationPoint {
      description "Observation Point";
      key name;
      leaf name {
        type nameType;
        description "Key of an observation point.";
      }
      leaf observationPointId {
        type uint32;
        config false;
        description "Observation Point ID...";
        reference "RFC 5102, Section 5.1.10.";
      }
      leaf observationDomainId {
        type uint32;
        mandatory true;
        description "The Observation Domain ID associates...";
        reference "RFC 5101.";
      }
      choice OPLocation {
        mandatory true;
        description "Location of the Observation Point.";
        leaf ifIndex {
          type uint32;
          description "Index of an interface...";
          reference "RFC 2863.";
        }
        leaf ifName {
          type string;
          description "Name of an interface...";
          reference "RFC 2863.";
        }
        leaf entPhysicalIndex {
          type uint32;
          description "Index of a linecard...";
```

```
          reference "RFC 4133.";
        }
        leaf entPhysicalName {
          type string;
          description "Name of a linecard...";
          reference "RFC 4133.";
        }
      }
      leaf direction {
        type direction;
        default both;
        description "Direction of packets....";
      }
      leaf-list selectionProcess {
        type instance-identifier { ct:instance-type SelectionProcess; }
        description "Selection Processes in this list process packets
          in parallel.";
      }
    }

    ct:complex-type Selector {
      ct:abstract true;
      description "Abstract selector";
      key name;
      leaf name {
          type nameType;
          description "Key of a selector";
      }
      leaf packetsObserved {
        type yang:counter64;
        config false;
        description "The number of packets observed ...";
        reference "RFC 5815, Section 8
          (ipfixSelectionProcessStatsPacketsObserved).";
      }
      leaf packetsDropped {
        type yang:counter64;
        config false;
        description "The total number of packets discarded ...";
        reference "RFC 5815, Section 8
          (ipfixSelectionProcessStatsPacketsDropped).";
      }
      leaf selectorDiscontinuityTime {
        type yang:date-and-time;
        config false;
        description "Timestamp of the most recent occasion at which
          one or more of the Selector counters suffered a
          discontinuity...";
```

```
        reference "RFC 5815, Section 8
          (ipfixSelectionProcessStatsDiscontinuityTime).";
      }
    }

    ct:complex-type SelectAllSelector {
      ct:extends Selector;
      description "Method that selects all packets.";
    }

    ct:complex-type SampCountBasedSelector {
          if-feature psampSampCountBased;
          ct:extends Selector;
          description "Selector applying systematic count-based
            packet sampling to the packet stream.";
          reference "RFC 5475, Section 5.1;
            RFC 5476, Section 6.5.2.1.";
          leaf packetInterval {
            type uint32;
            units packets;
            mandatory true;
            description "The number of packets that are consecutively
              sampled between gaps of length packetSpace.
              This parameter corresponds to the Information Element
              samplingPacketInterval.";
            reference "RFC 5477, Section 8.2.2.";
          }
          leaf packetSpace {
            type uint32;
            units packets;
            mandatory true;
            description "The number of unsampled packets between two
              sampling intervals.
              This parameter corresponds to the Information Element
              samplingPacketSpace.";
            reference "RFC 5477, Section 8.2.3.";
          }
    }

    ct:complex-type SampTimeBasedSelector {
          if-feature psampSampTimeBased;
          ct:extends Selector;
          description "Selector applying systematic time-based
            packet sampling to the packet stream.";
          reference "RFC 5475, Section 5.1;
            RFC 5476, Section 6.5.2.2.";
          leaf timeInterval {
            type uint32;
```

```
            units microseconds;
            mandatory true;
            description "The time interval in microseconds during
              which all arriving packets are sampled between gaps
              of length timeSpace.
              This parameter corresponds to the Information Element
              samplingTimeInterval.";
            reference "RFC 5477, Section 8.2.4.";
          }
          leaf timeSpace {
            type uint32;
            units microseconds;
            mandatory true;
            description "The time interval in microseconds during
              which no packets are sampled between two sampling
              intervals specified by timeInterval.
              This parameter corresponds to the Information Element
              samplingTimeInterval.";
            reference "RFC 5477, Section 8.2.5.";
          }
      }

      ct:complex-type SampRandOutOfNSelector {
            if-feature psampSampRandOutOfN;
            ct:extends Selector;
            description "This container contains the configuration
              parameters of a Selector applying n-out-of-N packet
              sampling to the packet stream.";
            reference "RFC 5475, Section 5.2.1;
              RFC 5476, Section 6.5.2.3.";
            leaf size {
              type uint32;
              units packets;
              mandatory true;
              description "The number of elements taken from the parent
                population.
                This parameter corresponds to the Information Element
                samplingSize.";
              reference "RFC 5477, Section 8.2.6.";
            }
            leaf population {
              type uint32;
              units packets;
              mandatory true;
              description "The number of elements in the parent
                population.
                This parameter corresponds to the Information Element
                samplingPopulation.";
```

```
            reference "RFC 5477, Section 8.2.7.";
          }
      }

      ct:complex-type SampUniProbSelector {
          if-feature psampSampUniProb;
          ct:extends Selector;
          description "Selector applying uniform probabilistic
            packet sampling (with equal probability per packet) to the
            packet stream.";
          reference "RFC 5475, Section 5.2.2.1;
            RFC 5476, Section 6.5.2.4.";
          leaf probability {
            type decimal64 {
              fraction-digits 18;
              range "0..1";
            }
            mandatory true;
            description "Probability that a packet is sampled,
              expressed as a value between 0 and 1.  The probability
              is equal for every packet.
              This parameter corresponds to the Information Element
              samplingProbability.";
            reference "RFC 5477, Section 8.2.8.";
          }
      }

      ct:complex-type FilterMatchSelector {
          if-feature psampFilterMatch;
          ct:extends Selector;
          description "This container contains the configuration
            parameters of a Selector applying property match filtering
            to the packet stream.";
          reference "RFC 5475, Section 6.1;
            RFC 5476, Section 6.5.2.5.";
          choice nameOrId {
            mandatory true;
            description "The field to be matched is specified by
              either the name or the ID of the Information
              Element.";
            leaf ieName {
              type string;
              description "Name of the Information Element.";
            }
            leaf ieId {
              type uint16 {
                range "1..32767" {
                  description "Valid range of Information Element
```

```
                    identifiers.";
                  reference "RFC 5102, Section 4.";
                }
              }
              description "ID of the Information Element.";
            }
          }
          leaf ieEnterpriseNumber {
            type uint32;
            description "If present, ... ";
          }
          leaf value {
            type string;
            mandatory true;
            description "Matching value of the Information Element.";
          }
      }

      ct:complex-type FilterHashSelector {
            if-feature psampFilterHash;
            ct:extends Selector;
            description "This container contains the configuration
              parameters of a Selector applying hash-based filtering
              to the packet stream.";
            reference "RFC 5475, Section 6.2;
              RFC 5476, Section 6.5.2.6.";
            leaf hashFunction {
              type identityref {
                base "hashFunction";
              }
              default BOB;
              description "Hash function to be applied.  According to
                RFC 5475, Section 6.2.4.1, BOB hash function must be
                used in order to be compliant with PSAMP.";
            }
            leaf ipPayloadOffset {
              type uint64;
              units octets;
              default 0;
              description "IP payload offset ... ";
              reference "RFC 5477, Section 8.3.2.";
            }
            leaf ipPayloadSize {
              type uint64;
              units octets;
              default 8;
              description "Number of IP payload bytes ... ";
              reference "RFC 5477, Section 8.3.3.";
```

```
        }
        leaf digestOutput {
          type boolean;
          default false;
          description "If true, the output ... ";
          reference "RFC 5477, Section 8.3.8.";
        }
        leaf initializerValue {
          type uint64;
          description "Initializer value to the hash function.
            If not configured by the user, the Monitoring Device
            arbitrarily chooses an initializer value.";
          reference "RFC 5477, Section 8.3.9.";
        }
        list selectedRange {
          key name;
          min-elements 1;
          description "List of hash function return ranges for
            which packets are selected.";
          leaf name {
            type nameType;
            description "Key of this list.";
          }
          leaf min {
            type uint64;
            description "Beginning of the hash function's selected
              range.
              This parameter corresponds to the Information Element
              hashSelectedRangeMin.";
            reference "RFC 5477, Section 8.3.6.";
          }
          leaf max {
            type uint64;
            description "End of the hash function's selected range.
              This parameter corresponds to the Information Element
              hashSelectedRangeMax.";
            reference "RFC 5477, Section 8.3.7.";
          }
        }
    }

    ct:complex-type Cache {
      ct:abstract true;
      description "Cache of a Monitoring Device.";
      key name;
      leaf name {
        type nameType;
        description "Key of a cache";
```

```
      }
      leaf-list exportingProcess {
        type leafref { path "/ipfix/exportingProcess/name"; }
        description "Records are exported by all Exporting Processes
          in the list.";
      }
      description "Configuration and state parameters of a Cache.";
      container cacheLayout {
        description "Cache Layout.";
        list cacheField {
          key name;
          min-elements 1;
          description "List of fields in the Cache Layout.";
          leaf name {
            type nameType;
            description "Key of this list.";
          }
          choice nameOrId {
            mandatory true;
            description "Name or ID of the Information Element.";
            reference "RFC 5102.";
            leaf ieName {
              type string;
              description "Name of the Information Element.";
            }
            leaf ieId {
              type uint16 {
                range "1..32767" {
                  description "Valid range of Information Element
                      identifiers.";
                  reference "RFC 5102, Section 4.";
                }
              }
              description "ID of the Information Element.";
            }
          }
          leaf ieLength {
            type uint16;
            units octets;
            description "Length of the field ... ";
            reference "RFC 5101, Section 6.2; RFC 5102.";
          }
          leaf ieEnterpriseNumber {
            type uint32;
            description "If present, the Information Element is
              enterprise-specific. ... ";
            reference "RFC 5101; RFC 5102.";
          }
```

```
          leaf isFlowKey {
            when "(../../../cacheMode != 'immediate')
              and
              ((count(../ieEnterpriseNumber) = 0)
              or
              (../ieEnterpriseNumber != 29305))" {
              description "This parameter is not available
                for Reverse Information Elements (which have
                enterprise number 29305) or if the Cache Mode
                is 'immediate'.";
            }
            type empty;
            description "If present, this is a flow key.";
          }
        }
      }
      leaf dataRecords {
        type yang:counter64;
        units "Data Records";
        config false;
        description "The number of Data Records generated ... ";
        reference "RFC 5815, Section 8
          (ipfixMeteringProcessCacheDataRecords).";
      }
      leaf cacheDiscontinuityTime {
        type yang:date-and-time;
        config false;
        description "Timestamp of the ... ";
        reference "RFC 5815, Section 8
          (ipfixMeteringProcessCacheDiscontinuityTime).";
      }
    }


    ct:complex-type ImmediateCache {
      if-feature cacheModeImmediate;
      ct:extends Cache;
    }

    ct:complex-type NonImmediateCache {
      ct:abstract true;
      ct:extends Cache;
      leaf maxFlows {
        type uint32;
        units flows;
        description "This parameter configures the maximum number of
          Flows in the Cache ... ";
      }
```

```
       leaf activeFlows {
         type yang:gauge32;
         units flows;
         config false;
         description "The number of Flows currently active in this
           Cache.";
         reference "RFC 5815, Section 8
           (ipfixMeteringProcessCacheActiveFlows).";
       }
       leaf unusedCacheEntries {
         type yang:gauge32;
         units flows;
         config false;
         description "The number of unused Cache entries in this
           Cache.";
         reference "RFC 5815, Section 8
           (ipfixMeteringProcessCacheUnusedCacheEntries).";
       }
     }

     ct:complex-type NonPermanentCache {
       ct:abstract true;
       ct:extends NonImmediateCache;
       leaf activeTimeout {
         type uint32;
         units milliseconds;
         description "This parameter configures the time in
           milliseconds after which ... ";
       }
       leaf inactiveTimeout {
         type uint32;
         units milliseconds;
         description "This parameter configures the time in
           milliseconds after which ... ";
       }
     }

     ct:complex-type NaturalCache {
       if-feature cacheModeNatural;
       ct:extends NonPermanentCache;
     }

     ct:complex-type TimeoutCache {
       if-feature cacheModeTimeout;
       ct:extends NonPermanentCache;
     }

     ct:complex-type PermanentCache {
```

```
      if-feature cacheModePermanent;
      ct:extends NonImmediateCache;
      leaf exportInterval {
        type uint32;
        units milliseconds;
        description "This parameter configures the interval for
          periodical export of Flow Records in milliseconds.
          If not configured by the user, the Monitoring Device sets
          this parameter.";
       }
    }

    ct:complex-type ExportDestination {
      ct:abstract true;
      description "Abstract export destination.";
      key name;
      leaf name {
        type nameType;
        description "Key of an export destination.";
      }
    }

    ct:complex-type IpDestination {
      ct:abstract true;
      ct:extends ExportDestination;
      description "IP export destination.";
      leaf ipfixVersion {
         type uint16;
         default 10;
         description "IPFIX version number.";
       }
       leaf destinationPort {
         type inet:port-number;
         description "If not configured by the user, the Monitoring
           Device uses the default port number for IPFIX, which is
           4739 without Transport Layer Security, and 4740 if Transport
           Layer Security is activated.";
       }
       choice indexOrName {
         description "Index or name of the interface ... ";
         reference "RFC 2863.";
         leaf ifIndex {
           type uint32;
           description "Index of an interface as stored in the ifTable
             of IF-MIB.";
           reference "RFC 2863.";
         }
         leaf ifName {
```

```
          type string;
          description "Name of an interface as stored in the ifTable
            of IF-MIB.";
          reference "RFC 2863.";
        }
      }
      leaf sendBufferSize {
        type uint32;
        units bytes;
        description "Size of the socket send buffer.
          If not configured by the user, this parameter is set by
          the Monitoring Device.";
      }
      leaf rateLimit {
        type uint32;
        units "bytes per second";
        description "Maximum number of bytes per second ... ";
        reference "RFC 5476, Section 6.3";
      }
      container transportLayerSecurity {
        presence "If transportLayerSecurity is present, DTLS is
          enabled if the transport protocol is SCTP or UDP, and TLS
          is enabled if the transport protocol is TCP.";
        description "Transport Layer Security configuration.";
        uses transportLayerSecurityParameters;
      }
      container transportSession {
        config false;
        description "State parameters of the Transport Session
          directed to the given destination.";
        uses transportSessionParameters;
      }
    }

     ct:complex-type SctpExporter {
       ct:extends IpDestination;
       description "SCTP exporter.";
       leaf-list sourceIPAddress {
         type inet:ip-address;
         description "List of source IP addresses used ... ";
         reference "RFC 4960, Section 6.4
           (Multi-Homed SCTP Endpoints).";
       }
       leaf-list destinationIPAddress {
         type inet:ip-address;
         min-elements 1;
         description "One or multiple IP addresses ... ";
         reference "RFC 4960, Section 6.4
```

```
          (Multi-Homed SCTP Endpoints).";
      }
      leaf timedReliability {
        type uint32;
        units milliseconds;
        default 0;
        description "Lifetime in milliseconds ... ";
        reference "RFC 3758; RFC 4960.";
      }
    }

    ct:complex-type UdpExporter {
      ct:extends IpDestination;
      if-feature udpTransport;
      description "UDP parameters.";
      leaf sourceIPAddress {
        type inet:ip-address;
        description "Source IP address used by the Exporting
           Process ...";
      }
      leaf destinationIPAddress {
        type inet:ip-address;
        mandatory true;
        description "IP address of the Collection Process to which
          IPFIX Messages are sent.";
      }
      leaf maxPacketSize {
        type uint16;
        units octets;
        description "This parameter specifies the maximum size of
          IP packets ... ";
      }
      leaf templateRefreshTimeout {
        type uint32;
        units seconds;
        default 600;
        description "Sets time after which Templates are resent in the
          UDP Transport Session. ... ";
        reference "RFC 5101, Section 10.3.6; RFC 5815, Section 8
          (ipfixTransportSessionTemplateRefreshTimeout).";
      }
      leaf optionsTemplateRefreshTimeout {
        type uint32;
        units seconds;
        default 600;
        description "Sets time after which Options Templates are
          resent in the UDP Transport Session. ... ";
        reference "RFC 5101, Section 10.3.6; RFC 5815, Section 8
```

```
          (ipfixTransportSessionOptionsTemplateRefreshTimeout).";
      }
      leaf templateRefreshPacket {
        type uint32;
        units "IPFIX Messages";
        description "Sets number of IPFIX Messages after which
          Templates are resent in the UDP Transport Session. ... ";
        reference "RFC 5101, Section 10.3.6; RFC 5815, Section 8
          (ipfixTransportSessionTemplateRefreshPacket).";
      }
      leaf optionsTemplateRefreshPacket {
        type uint32;
        units "IPFIX Messages";
        description "Sets number of IPFIX Messages after which
          Options Templates are resent in the UDP Transport Session
          protocol. ... ";
        reference "RFC 5101, Section 10.3.6; RFC 5815, Section 8
          (ipfixTransportSessionOptionsTemplateRefreshPacket).";
      }
    }

     ct:complex-type TcpExporter {
       ct:extends IpDestination;
       if-feature tcpTransport;
       description "TCP exporter";
       leaf sourceIPAddress {
         type inet:ip-address;
         description "Source IP address used by the Exporting
           Process...";
       }
       leaf destinationIPAddress {
         type inet:ip-address;
         mandatory true;
         description "IP address of the Collection Process to which
           IPFIX Messages are sent.";
       }
     }

     ct:complex-type FileWriter {
       ct:extends ExportDestination;
       if-feature fileWriter;
       description "File Writer.";
       leaf ipfixVersion {
         type uint16;
         default 10;
         description "IPFIX version number.";
       }
       leaf file {
```

```
        type inet:uri;
        mandatory true;
        description "URI specifying the location of the file.";
      }
      leaf bytes {
        type yang:counter64;
        units octets;
        config false;
        description "The number of bytes written by the File
          Writer...";
      }
      leaf messages {
        type yang:counter64;
        units "IPFIX Messages";
        config false;
        description "The number of IPFIX Messages written by the File
          Writer. ... ";
      }
      leaf discardedMessages {
        type yang:counter64;
        units "IPFIX Messages";
        config false;
        description "The number of IPFIX Messages that could not be
          written by the File Writer ... ";
      }
      leaf records {
        type yang:counter64;
        units "Data Records";
        config false;
        description "The number of Data Records written by the File
          Writer. ... ";
      }
      leaf templates {
        type yang:counter32;
        units "Templates";
        config false;
        description "The number of Template Records (excluding
          Options Template Records) written by the File Writer.
          ... ";
      }
      leaf optionsTemplates {
        type yang:counter32;
        units "Options Templates";
        config false;
        description "The number of Options Template Records written
          by the File Writer. ... ";
      }
      leaf fileWriterDiscontinuityTime {
```

```
        type yang:date-and-time;
        config false;
        description "Timestamp of the most recent occasion at which
          one or more File Writer counters suffered a discontinuity.
          ... ";
      }
      list template {
        config false;
        description "This list contains the Templates and Options
          Templates that have been written by the File Reader. ... ";
        uses templateParameters;
      }
    }

    ct:complex-type ExportingProcess {
      if-feature exporter;
      description "Exporting Process of the Monitoring Device.";
      key name;
      leaf name {
        type nameType;
        description "Key of this list.";
      }
      leaf exportMode {
        type identityref {
          base "exportMode";
        }
        default parallel;
        description "This parameter determines to which configured
          destination(s) the incoming Data Records are exported.";
      }
      ct:instance-list destination {
        ct:instance-type ExportDestination;
        min-elements 1;
        description "Export destinations.";
      }
      list options {
        key name;
        description "List of options reported by the Exporting
          Process.";
        leaf name {
          type nameType;
          description "Key of this list.";
        }
        leaf optionsType {
          type identityref {
            base "optionsType";
          }
          mandatory true;
```

```
          description "Type of the exported options data.";
        }
        leaf optionsTimeout {
          type uint32;
          units milliseconds;
          description "Time interval for periodic export of the options
            data. ... ";
        }
      }
    }

    ct:complex-type CollectingProcess {
      description "A Collecting Process.";
      key name;
      leaf name {
        type nameType;
        description "Key of a collecing process.";
      }
      ct:instance-list sctpCollector {
        ct:instance-type SctpCollector;
        description "List of SCTP receivers (sockets) on which the
          Collecting Process receives IPFIX Messages.";
      }
      ct:instance-list udpCollector {
        if-feature udpTransport;
        ct:instance-type UdpCollector;
        description "List of UDP receivers (sockets) on which the
          Collecting Process receives IPFIX Messages.";
      }
      ct:instance-list tcpCollector {
        if-feature tcpTransport;
        ct:instance-type TcpCollector;
        description "List of TCP receivers (sockets) on which the
          Collecting Process receives IPFIX Messages.";
      }
      ct:instance-list fileReader {
        if-feature fileReader;
        ct:instance-type FileReader;
        description "List of File Readers from which the Collecting
          Process reads IPFIX Messages.";
      }
      leaf-list exportingProcess {
        type instance-identifier { ct:instance-type ExportingProcess; }
        description "Export of received records without any
          modifications.  Records are processed by all Exporting
          Processes in the list.";
      }
    }
```

```
ct:complex-type Collector {
    ct:abstract true;
    description "Abstract collector.";
    key name;
    leaf name {
      type nameType;
      description "Key of collectors";
    }
}

ct:complex-type IpCollector {
  ct:abstract true;
  ct:extends Collector;
  description "Collector for IP transport protocols.";
  leaf localPort {
    type inet:port-number;
    description "If not configured, the Monitoring Device uses the
      default port number for IPFIX, which is 4739 without
      Transport Layer Security, and 4740 if Transport Layer
      Security is activated.";
  }
  container transportLayerSecurity {
    presence "If transportLayerSecurity is present, DTLS is enabled
      if the transport protocol is SCTP or UDP, and TLS is enabled
      if the transport protocol is TCP.";
    description "Transport Layer Security configuration.";
    uses transportLayerSecurityParameters;
  }
  list transportSession {
    config false;
    description "This list contains the currently established
      Transport Sessions terminating at the given socket.";
    uses transportSessionParameters;
  }
}

ct:complex-type SctpCollector {
  ct:extends IpCollector;
  description "Collector listening on an SCTP socket";
  leaf-list localIPAddress {
    type inet:ip-address;
    description "List of local IP addresses ... ";
    reference "RFC 4960, Section 6.4
      (Multi-Homed SCTP Endpoints).";
  }
}

ct:complex-type UdpCollector {
```

```
      ct:extends IpCollector;
      description "Parameters of a listening UDP socket at a
        Collecting Process.";
      leaf-list localIPAddress {
        type inet:ip-address;
        description "List of local IP addresses on which the Collecting
          Process listens for IPFIX Messages.";
      }
      leaf templateLifeTime {
        type uint32;
        units seconds;
        default 1800;
        description "Sets the lifetime of Templates for all UDP
          Transport Sessions ... ";
        reference "RFC 5101, Section 10.3.7; RFC 5815, Section 8
          (ipfixTransportSessionTemplateRefreshTimeout).";
      }
      leaf optionsTemplateLifeTime {
        type uint32;
        units seconds;
        default 1800;
        description "Sets the lifetime of Options Templates for all
          UDP Transport Sessions terminating at this UDP socket.
          ... ";
        reference "RFC 5101, Section 10.3.7; RFC 5815, Section 8
          (ipfixTransportSessionOptionsTemplateRefreshTimeout).";
      }
      leaf templateLifePacket {
        type uint32;
        units "IPFIX Messages";
        description "If this parameter is configured, Templates
          defined in a UDP Transport Session become invalid if ...";
        reference "RFC 5101, Section 10.3.7; RFC 5815, Section 8
          (ipfixTransportSessionTemplateRefreshPacket).";
      }
      leaf optionsTemplateLifePacket {
        type uint32;
        units "IPFIX Messages";
        description "If this parameter is configured, Options
          Templates defined in a UDP Transport Session become
          invalid if ...";
        reference "RFC 5101, Section 10.3.7; RFC 5815, Section 8
          (ipfixTransportSessionOptionsTemplateRefreshPacket).";
      }
    }

    ct:complex-type TcpCollector {
      ct:extends IpCollector;
```

```
      description "Collector listening on a TCP socket.";
      leaf-list localIPAddress {
        type inet:ip-address;
        description "List of local IP addresses on which the Collecting
          Process listens for IPFIX Messages.";
      }
    }

    ct:complex-type FileReader {
      ct:extends Collector;
      description "File Reading collector.";
      leaf file {
        type inet:uri;
        mandatory true;
        description "URI specifying the location of the file.";
      }
      leaf bytes {
        type yang:counter64;
        units octets;
        config false;
        description "The number of bytes read by the File Reader.
          ... ";
      }
      leaf messages {
        type yang:counter64;
        units "IPFIX Messages";
        config false;
        description "The number of IPFIX Messages read by the File
          Reader. ... ";
      }
      leaf records {
        type yang:counter64;
        units "Data Records";
        config false;
        description "The number of Data Records read by the File
          Reader. ... ";
      }
      leaf templates {
        type yang:counter32;
        units "Templates";
        config false;
        description "The number of Template Records (excluding
          Options Template Records) read by the File Reader. ...";
      }
      leaf optionsTemplates {
        type yang:counter32;
        units "Options Templates";
        config false;
```

```
        description "The number of Options Template Records read by
          the File Reader. ... ";
      }
      leaf fileReaderDiscontinuityTime {
        type yang:date-and-time;
        config false;
        description "Timestamp of the most recent occasion ... ";
      }
      list template {
        config false;
        description "This list contains the Templates and Options
          Templates that have been read by the File Reader.
          Withdrawn or invalidated (Options) Templates MUST be removed
          from this list.";
        uses templateParameters;
      }
    }

    ct:complex-type SelectionProcess {
        description "Selection Process";
        key name;
        leaf name {
          type nameType;
          description "Key of a selection process.";
        }
        ct:instance-list selector {
          ct:instance-type Selector;
          min-elements 1;
          ordered-by user;
          description "List of Selectors that define the action of the
            Selection Process on a single packet.  The Selectors are
            serially invoked in the same order as they appear in this
            list.";
        }
        list selectionSequence {
          config false;
          description "This list contains the Selection Sequence IDs
            which are assigned by the Monitoring Device ... ";
          reference "RFC 5476.";
          leaf observationDomainId {
            type uint32;
            description "Observation Domain ID for which the
              Selection Sequence ID is assigned.";
          }
          leaf selectionSequenceId {
            type uint64;
            description "Selection Sequence ID used in the Selection
              Sequence (Statistics) Report Interpretation.";
```

```
          }
        }
        leaf cache {
          type instance-identifier { ct:instance-type Cache; }
          description "Cache which receives the output of the
            Selection Process.";
        }
      }

    /*****************************************************************
    * Groupings
    *****************************************************************/

    grouping transportLayerSecurityParameters {
      description "Transport layer security parameters.";
      leaf-list localCertificationAuthorityDN {
        type string;
        description "Distinguished names of certification authorities
          whose certificates may be used to identify the local
          endpoint.";
      }
      leaf-list localSubjectDN {
        type string;
        description "Distinguished names that may be used in the
          certificates to identify the local endpoint.";
      }
      leaf-list localSubjectFQDN {
        type inet:domain-name;
        description "Fully qualified domain names that may be used to
          in the certificates to identify the local endpoint.";
      }
      leaf-list remoteCertificationAuthorityDN {
        type string;
        description "Distinguished names of certification authorities
          whose certificates are accepted to authorize remote
          endpoints.";
      }
      leaf-list remoteSubjectDN {
        type string;
        description "Distinguished names that are accepted in
          certificates to authorize remote endpoints.";
      }
      leaf-list remoteSubjectFQDN {
        type inet:domain-name;
        description "Fully qualified domain names that are accepted in
          certificates to authorize remote endpoints.";
      }
    }
```

```
    grouping templateParameters {
      description "State parameters of a Template used by an Exporting
        Process or received by a Collecting Process ... ";
      reference "RFC 5101; RFC 5815, Section 8 (ipfixTemplateEntry,
        ipfixTemplateDefinitionEntry, ipfixTemplateStatsEntry)";
      leaf observationDomainId {
        type uint32;
        description "The ID of the Observation Domain for which this
          Template is defined.";
        reference "RFC 5815, Section 8
          (ipfixTemplateObservationDomainId).";
      }
      leaf templateId {
        type uint16 {
          range "256..65535" {
            description "Valid range of Template Ids.";
            reference "RFC 5101";
          }
        }
        description "This number indicates the Template Id in the IPFIX
          message.";
        reference "RFC 5815, Section 8 (ipfixTemplateId).";
      }
      leaf setId {
        type uint16;
        description "This number indicates the Set Id of the Template.
          ... ";
        reference "RFC 5815, Section 8 (ipfixTemplateSetId).";
      }
      leaf accessTime {
        type yang:date-and-time;
        description "Used for Exporting Processes, ... ";
        reference "RFC 5815, Section 8 (ipfixTemplateAccessTime).";
      }
      leaf templateDataRecords {
        type yang:counter64;
        description "The number of transmitted or received Data
          Records ... ";
        reference "RFC 5815, Section 8 (ipfixTemplateDataRecords).";
      }
      leaf templateDiscontinuityTime {
        type yang:date-and-time;
        description "Timestamp of the most recent occasion at which
          the counter templateDataRecords suffered a discontinuity.
          ... ";
        reference "RFC 5815, Section 8
          (ipfixTemplateDiscontinuityTime).";
      }
```

```
      list field {
        description "This list contains the (Options) Template
          fields of which the (Options) Template is defined.
          ... ";
        leaf ieId {
          type uint16 {
            range "1..32767" {
              description "Valid range of Information Element
                  identifiers.";
              reference "RFC 5102, Section 4.";
            }
          }
          description "This parameter indicates the Information
            Element Id of the field.";
          reference "RFC 5815, Section 8 (ipfixTemplateDefinitionIeId);
            RFC 5102.";
        }
        leaf ieLength {
          type uint16;
          units octets;
          description "This parameter indicates the length of the
            Information Element of the field.";
          reference "RFC 5815, Section 8
            (ipfixTemplateDefinitionIeLength); RFC 5102.";
        }
        leaf ieEnterpriseNumber {
          type uint32;
          description "This parameter indicates the IANA enterprise
            number of the authority ... ";
          reference "RFC 5815, Section 8
            (ipfixTemplateDefinitionEnterpriseNumber).";
        }
        leaf isFlowKey {
          when "../../setId = 2" {
            description "This parameter is available for non-Options
              Templates (Set Id is 2).";
          }
          type empty;
          description "If present, this is a Flow Key field.";
          reference "RFC 5815, Section 8
            (ipfixTemplateDefinitionFlags).";
        }
        leaf isScope {
          when "../../setId = 3" {
            description "This parameter is available for Options
              Templates (Set Id is 3).";
          }
          type empty;
```

```
          description "If present, this is a scope field.";
          reference "RFC 5815, Section 8
            (ipfixTemplateDefinitionFlags).";
        }
      }
    }

    grouping transportSessionParameters {
      description "State parameters of a Transport Session ... ";
      reference "RFC 5101; RFC 5815, Section 8
        (ipfixTransportSessionEntry,
         ipfixTransportSessionStatsEntry)";
      leaf ipfixVersion {
        type uint16;
        description "Used for Exporting Processes, this parameter
          contains the version number of the IPFIX protocol ... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionIpfixVersion).";
      }
      leaf sourceAddress {
        type inet:ip-address;
        description "The source address of the Exporter of the
          IPFIX Transport Session... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionSourceAddressType,
          ipfixTransportSessionSourceAddress).";
      }
      leaf destinationAddress {
        type inet:ip-address;
        description "The destination address of the Collector of
          the IPFIX Transport Session... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionDestinationAddressType,
          ipfixTransportSessionDestinationAddress).";
      }
      leaf sourcePort {
        type inet:port-number;
        description "The transport protocol port number of the
          Exporter of the IPFIX Transport Session.";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionSourcePort).";
      }
      leaf destinationPort {
        type inet:port-number;
        description "The transport protocol port number of the
          Collector of the IPFIX Transport Session... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionDestinationPort).";
```

```
      }
      leaf sctpAssocId {
        type uint32;
        description "The association id used for the SCTP session
          between the Exporter and the Collector ... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionSctpAssocId),
          RFC 3871";
      }
      leaf status {
        type transportSessionStatus;
        description "Status of the Transport Session.";
        reference "RFC 5815, Section 8 (ipfixTransportSessionStatus).";
      }
      leaf rate {
        type yang:gauge32;
        units "bytes per second";
        description "The number of bytes per second transmitted by the
          Exporting Process or received by the Collecting Process.
          This parameter is updated every second.";
        reference "RFC 5815, Section 8 (ipfixTransportSessionRate).";
      }
      leaf bytes {
        type yang:counter64;
        units bytes;
        description "The number of bytes transmitted by the
          Exporting Process or received by the Collecting
          Process ... ";
        reference "RFC 5815, Section 8 (ipfixTransportSessionBytes).";
      }
      leaf messages {
        type yang:counter64;
        units "IPFIX Messages";
        description "The number of messages transmitted by the
          Exporting Process or received by the Collecting Process... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionMessages).";
      }
      leaf discardedMessages {
        type yang:counter64;
        units "IPFIX Messages";
        description "Used for Exporting Processes, this parameter
          indicates the number of messages that could not be
          sent ...";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionDiscardedMessages).";
      }
      leaf records {
```

```
        type yang:counter64;
        units "Data Records";
        description "The number of Data Records transmitted ... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionRecords).";
      }
      leaf templates {
        type yang:counter32;
        units "Templates";
        description "The number of Templates transmitted by the
          Exporting Process or received by the Collecting Process.
          ... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionTemplates).";
      }
      leaf optionsTemplates {
        type yang:counter32;
        units "Options Templates";
        description "The number of Option Templates transmitted by the
          Exporting Process or received by the Collecting Process...";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionOptionsTemplates).";
      }
      leaf transportSessionStartTime {
        type yang:date-and-time;
        description "Timestamp of the start of the given Transport
          Session... ";
      }
      leaf transportSessionDiscontinuityTime {
        type yang:date-and-time;
        description "Timestamp of the most recent occasion at which
          one or more of the Transport Session counters suffered a
          discontinuity... ";
        reference "RFC 5815, Section 8
          (ipfixTransportSessionDiscontinuityTime).";
      }
      list template {
        description "This list contains the Templates and Options
          Templates that are transmitted by the Exporting Process
          or received by the Collecting Process.
          Withdrawn or invalidated (Options) Templates MUST be removed
          from this list.";
        uses templateParameters;
      }
    }

    /*****************************************************************
    * Main container
```

```
     ****************************************************************/

     container ipfix {
       description "Top-level node of the IPFIX/PSAMP configuration
         data model.";
       ct:instance-list collectingProcess {
         if-feature collector;
         ct:instance-type CollectingProcess;
       }

       ct:instance-list observationPoint {
         if-feature meter;
         ct:instance-type ObservationPoint;
       }

       ct:instance-list selectionProcess {
         if-feature meter;
         ct:instance-type SelectionProcess;
       }

       ct:instance-list cache {
         if-feature meter;
         description "Cache of the Monitoring Device.";
         ct:instance-type Cache;
       }

       ct:instance-list exportingProcess {
         if-feature exporter;
         description "Exporting Process of the Monitoring Device.";
         ct:instance-type ExportingProcess;
       }

     }
}
<CODE ENDS>
```

Authors' Addresses

Bernd Linowski
TCS/Nokia Siemens Networks
Heltorfer Strasse 1
Duesseldorf 40472
Germany

EMail: bernd.linowski.ext@nsn.com

Mehmet Ersue
Nokia Siemens Networks
St.-Martin-Strasse 76
Munich 81541
Germany

EMail: mehmet.ersue@nsn.com

Siarhei Kuryla
360 Treasury Systems
Grueneburgweg 16-18
Frankfurt am Main 60322
Germany

EMail: s.kuryla@gmail.com