Configuration

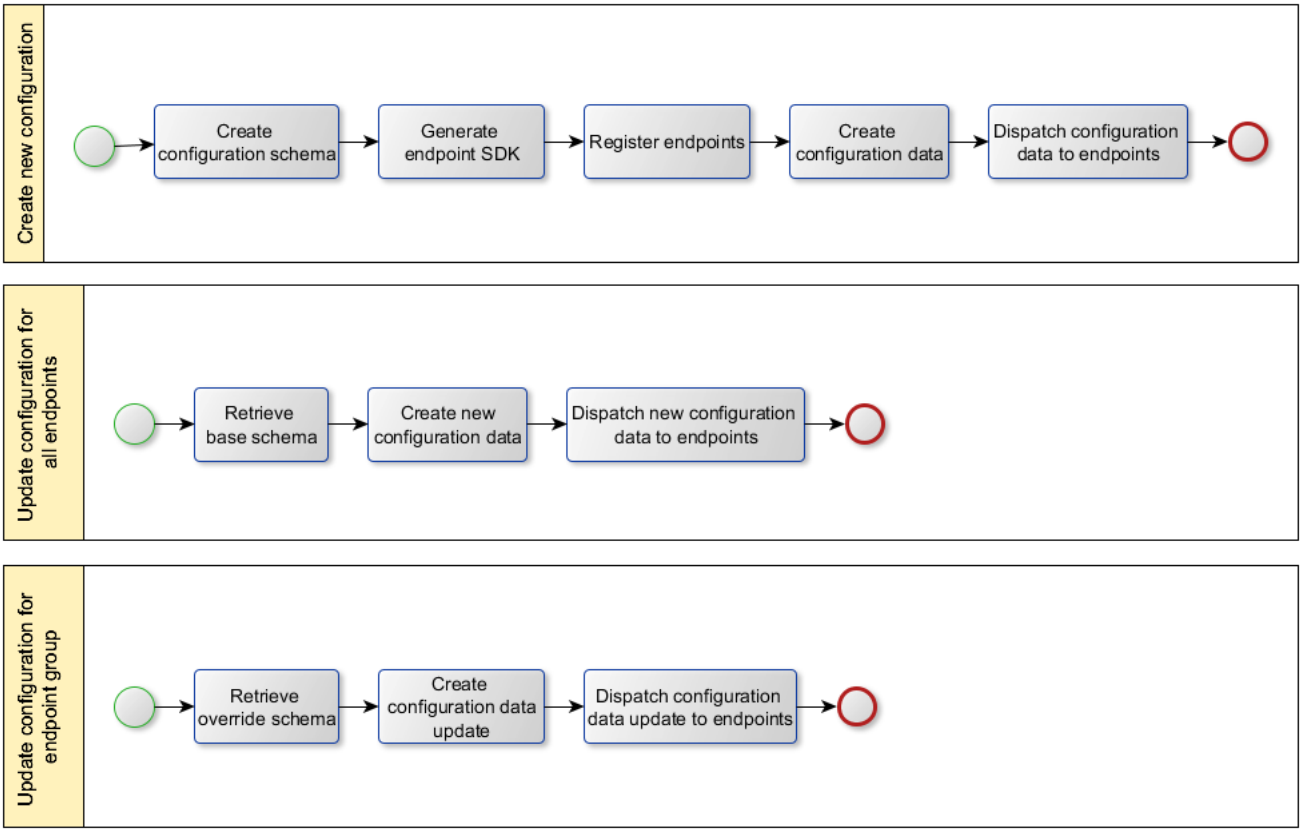
The Kaa Configuration subsystem is responsible for configuring endpoints by supplying them with the structured data of user-defined complexity that is managed via the Kaa server. The fact that Kaa operates with the uniformly structured data deserves a special emphasis. By knowing the data structure of the application, Kaa provides a number of very useful features, such as:

- Incremental data updates for endpoints
- Endpoint-specific data view that is based on the endpoint membership in endpoint groups
- Automatic generation of the data object model in the endpoint SDK
- Automatic generation of the default configuration
- Enforcement of the data integrity and validity on the server's northbound integration interface

The structure of the configuration data is determined by the customizable *configuration schema,* which is created under the application using Admin UI or REST API. It is the responsibility of the Kaa developer to construct the configuration schema for the application and make the Kaa client interpret the data supplied by the endpoint SDK. The Kaa administrator, in turn, can provision the configuration schema into the Kaa server and set the configuration data accordingly.

Once a new configuration schema is loaded into the Kaa application, the Control server automatically assigns a version number to the schema, generates three corresponding derivative schemas (a *base schema*, *override schema*, and *protocol schema*), and populates configuration data in the "all" group with the default values.

Configuration management overview



Configuration schema

A *configuration schema* is a user-defined specification of the application data model that Kaa Configuration subsystem uses to configure endpoints registered under the application. In other words, the configuration schema defines in which format the actual configuration data should be entered by the user/developer and then transferred to the endpoints.

The format of the configuration schema is similar to the profile schema and based on the Apache Avro schema. The Kaa Configuration subsystem supports all of the Avro primitive types: null, boolean, int, long, float, double, bytes, string, and most of the complex types: record, enum, array, union, and fixed.

NOTE:
The Avro map type (a set of <key, value> pairs) is not currently supported.
It is possible to define an array of unions, but Kaa expects all of the entities to be of the same type.

The following examples illustrate basic constituents of the configuration schema and their usage.

- The root object type in Kaa is always the record.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        ...
    ]
}
```

- The **name** and **namespace** attributes are both mandatory for the record type. They are used for the record referencing in derivative schemas.
- The **optional** field attribute (boolean, false by default) determines whether or not the field in the record is optional. Internally, Kaa translates optional fields into union fields with the null type at the top of the list (which automatically makes them default to null - see the Records auto-generation section for more details).
  In case of the optional union field, Kaa automatically puts null at the top of the types list in the union definition.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "optionalBytesField",
            "type": "bytes",
            "optional": true
        },
        {
            "name": "optionalSuit",
            "optional": true,
            "type": {
                "name": "suitT",
                "namespace": "org.kaaproject.sample",
                "type": "enum",
                "symbols" : ["spades", "hearts", "diamonds", "clubs"]
            }
        }
    ]
}
```

- The **by_default** field attribute (the value is interpreted according to the field type; it has no default value) determines the default value for the field and is used for generation of the default record. The **by_default** attribute must be present for all mandatory primitive record fields, except for null.

```
 {
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "stringField",
            "type": "string",
            "by_default": "default string value"
        },
        {
            "name": "intField",
            "type": "int",
            "by_default": 0
        }
    ]
}
```

The following table specifies the **by_default** attribute format for every supported primitive type.

Type	Format	Example
boolean	true/false	"by_default": true
int	numeric value from (-2^31+1) to (2^31-1)	"by_default": 55
long	numeric value from (-2^63+1) to (2^63-1)	"by_default": 2147483648
float	floating point value	"by_default": 1.432
double	floating point value	"by_default": 1.432
bytes	json array of byte values	"by_default": [1, 2, 55, 254, 4]
string	simple string format	"by_default": "abcdef"

- The **addressable** record type field attribute (boolean, true by default) determines whether or not the record supports partial updates (deltas). If this attribute is true, Kaa automatically adds a UUID field (__uuid) to the record when producing derivative schemas, which is done for the addressing purposes. For this reason, __uuid is a reserved field name in Kaa. Note that the root record in the configuration schema is always addressable and thus ignores the statement *"addressable": false*.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "nestedRecord",
            "type": {
                "name": "nonAddressableRecordT",
                "namespace": "org.kaaproject.sample",
                "type": "record",
                "addressable": false,
                "fields": [
                    {
                        "name": "booleanField",
                        "type": "boolean",
                        "by_default": false
                    },
                    {
                        "name": "intField",
                        "type": "int",
                        "by_default": "0"
                    }
                ]
            }
        }
    ]
}
```

- The **overrideStrategy** array type field attribute (string, "replace" by default) determines how to merge arrays in the configuration across the endpoint groups. Accepted values are "replace" and "append".

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "nestedRecord",
            "type": {
                "name": "nonAddressableRecordT",
                "namespace": "org.kaaproject.sample",
                "type": "record",
                "addressable": false,
                "fields": [
                    {
                        "name": "booleanField",
                        "type": "boolean",
                        "by_default": false
                    },
                    {
                        "name": "intField",
                        "type": "int",
                        "by_default": "0"
                    }
                ]
            }
        },
        {
            "name": "arrayAppended",
            "overrideStrategy": "append",
            "type": {
                "type": "array",
                "items": "org.kaaproject.sample.nonAddressableRecordT"
            }
        },
        {
            "name": "arrayOverride",
            "type": {
                "type": "array",
                "items": "float"
            }
        }
    ]
}
```

Addressing

A field in Kaa configuration is addressable if the record containing this field is addressable (the **addressable** attribute of the record is true). Addressable field values can be updated via the group-specific configuration mechanism.

The field address is formed by appending the field name to the containing record address and using the slash ("/") as a separator. The address of the root record is always "/".

Consider the following record schema example.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "intField",
            "type": "int",
            "by_default": 12345
        },
        {
            "name": "nestedRecord",
            "type": {
                "name": "nestedRecordT",
                "namespace": "org.kaaproject.sample",
                "type": "record",
                "fields": [
                    {
                        "name": "enumField",
                        "type": {
                            "name": "hashT",
                            "namespace": "org.kaaproject.sample",
                            "type": "fixed",
                            "size": 16
                        }
                    },
                    {
                        "name": "arrayField",
                        "type": {
                            "type": "array",
                            "items": "float"
                        }
                    }
                ]
            }
        },
        {
            "name": "arrayOfRecords",
            "type": {
                "type": "array",
                "items": "org.kaaproject.sample.nestedRecordT"
            }
        }
    ]
}
```

The following fields are addressable in the provided schema.

```
/intField
/nestedRecord
/nestedRecord/enumField
/nestedRecord/arrayField
/arrayOfRecords
```

However, the fields within the instances of org.kaaproject.sample.nestedRecordT contained in /arrayOfRecords are not addressable, because it is not possible to address records within arrays in the current Kaa version.

Records auto-generation

For every configuration schema, Kaa constructs the default configuration data records. During this process of the default configuration data generation, each record in the schema is analyzed one-by-one in the depth-first, top-to-bottom order, as follows:

- Union fields assume the first type listed in the union definition. The default value is generated according to the rules specific to the type encountered.
  NOTE: any optional fields (those having attribute "optional": true in the schema) are in fact unions with the first type being null. Therefore, optional fields default to the empty value.
- For a field of any primitive type (except for null), Kaa expects the **by_default** attribute to be present and provide the default field value. A schema missing such an attribute for a mandatory primitive record field generates an exception and gets rejected by Kaa.
- Non-optional record type fields are generated by applying the same record generation algorithm.
- Enum fields assume the first value listed in the type definition.
- Arrays are generated empty by default.
- Fixed type fields are generated filled in with zeros.
- __uuid fields of the org.kaaproject.configuration.uuidT type are generated with a valid UUID assigned to the value sub-field.

Consider the following record schema example.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "unionField",
            "type": ["string", "int", "null"],
            "by_default": "default string value"
        },
        {
            "name": "optionalUnionField",
            "type": ["string", "int", "null"],
            "optional": true
        },
        {
            "name": "optionalBoolean",
            "type": "boolean",
            "optional": true
        },
        {
            "name": "intField",
            "type": "int",
            "by_default": 12345
        },
        {
            "name": "mandatoryNestedRecord",
            "type": {
                "name": "nestedRecordT",
                "namespace": "org.kaaproject.sample",
                "type": "record",
                "fields": [
                    {
                        "name": "enumField",
                        "type": {
                            "name": "suitT",
                            "namespace": "org.kaaproject.sample",
                            "type": "enum",
                            "symbols": ["spades", "hearts", "diamonds", "clubs"]
                        }
                    },
                    {
```

```
                    "name": "arrayField",
                    "type": {
                        "type": "array",
                        "items": "float"
                    }
                },
                {
                    "name": "enumField",
                    "type": {
                        "name": "hashT",
                        "namespace": "org.kaaproject.sample",
                        "type": "fixed",
                        "size": 16
                    }
                }
            ]
        }
```

```
        }
    ]
}
```

This schema would default to the following record.

```
{
    "unionField": "default string value",
    "optionalUnionField": null,
    "optionalBoolean": null,
    "intField": 12345,
    "mandatoryNestedRecord": {
        "enumField": "spades",
        "arrayField": [],
        "enumField": [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    }
}
```

The Kaa server stores the auto-generated default records in a cache and re-uses them in further operation.

Schema-specific configuration management

Kaa allows simultaneously updating configuration data for all endpoints under the application which use the same configuration schema. For this purpose, Kaa uses a *base schema* which is derived from the configuration schema. The base schema assigns an address to each addressable record from the configuration schema and in this way enables updating the record values.

For each Kaa application, there is a default "all" endpoint group to which all endpoints registered under the application belong. The "all" group contains base schemas for all configuration schemas under the application as well as the corresponding default configuration data sets. The "all" group has weight 0. When a new configuration schema version is loaded into the application, the server automatically creates the corresponding base schema and default configuration data for the "all" group (by applying the default records generation algorithm described in the previous section to the root record in the base schema). This configuration data may be changed either via the Web UI or through the integration interface as follows: the user retrieves the corresponding base schema, specifies new configuration data that corresponds to the base schema format, and submits this data to the server.

After the new configuration is submitted, the server overwrites the current configuration data which is stored in the "all" group for the base schema in question. Finally, Kaa delivers the up-to-date configuration data to all endpoints that support the corresponding configuration schema.

The fact that the "all" group has weight 0 results in the following: if apart from the schema-specific update the user introduces the group-specific update too, any conflicts in values between these two updates will be resolved by giving preference to the group-specific update (because 0 is the lowest priority value).

Base schema

The base schema is obtained by transforming the configuration schema, as follows:

- Optional fields are transformed into unions with the null type at the top of the type list.
- The __uuid field of the union type ["org.kaaproject.configuration.uuidT", "null"] is added to every record having addressable = true. In the following example, the org.kaaproject.configuration.uuidT type is declared in the nestedRecordT record type, and reused in rootT.

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "mandatoryNestedRecord",
            "type": {
                "name": "nestedRecordT",
                "namespace": "org.kaaproject.sample",
                "type": "record",
                "addressable": false,
                "fields": [
                    {
                        "name": "booleanField",
                        "type": "boolean",
                        "by_default": false
                    },
                    {
                        "name": "__uuid",
                        "type": [
                            {
                                "name": "uuidT",
                                "type": "fixed",
                                "size": 16,
                                "namespace": "org.kaaproject.configuration"
                            },
                            "null"
                        ]
                    }
                ]
            }
        },
        {
            "name": "__uuid",
            "type": [
                "org.kaaproject.configuration.uuidT",
                "null"
            ]
        }
    ]
}
```

Updating configuration data

The Kaa Control server exposes API for loading configuration data into the "all" group. To introduce the configuration update, the new data set must be supplied in the Avro binary or Avro JSON format and correspond to the base schema version specified in the API call. As a result, all the endpoints under the application will assume the new configuration.

The following rules apply during the update:

- When loading the new data set, Kaa persists the existing UUIDs for all addressable records, thus ignoring the corresponding values supplied with the new configuration.
- In case of unions that assumes that the record types in the old configuration and in the new configuration match. If they don't, Kaa generates a new UUID value for the replacement record.
- In case of non-addressable records (those directly or indirectly contained in the arrays), Kaa performs record instances matching by comparing the UUID values in the old and the new configuration.

UUID validation

Kaa performs UUID validation to prevent corruption of configuration data when the API user or Kaa administrator loads a new version of the configuration data to the server. In this case, Kaa inspects the UUIDs of the newly introduced configuration data and validates them against the UUIDs in the previous version of the configuration data.

UUID validation is performed in the following two cases: updating an existing object or creating a new object.

Updating an existing object (there is a UUID in the previous version of the configuration)

The following rules apply for the object update.

- If the resolved UUID remains the same, the new configuration is stored with the existing UUID.
- If the resolved UUID is null or unknown and the object is a record, Kaa does a search throughout the existing base schema to find the object in question and obtain its existing UUID; after the search is complete, the new configuration is stored with the existing UUID.
- If the resolved UUID is null or unknown and the object is an array, Kaa generates a new UUID for the new configuration.

Creating a new object (there is no UUID in previous version of the configuration)

The following rules apply for object creation.

- If the resolved UUID is null, Kaa generates a new UUID for the object and stores it in the database.
- If the resolved UUID is equal to some other UUID which is already in use or to unknown value, Kaa generates a new UUID for the object and stores it in the database.

Group-specific configuration management

Kaa allows updating configuration data for a specific endpoint group under the application, which is a more targeted approach than that described in Schema-specific configuration management for the "all" group. To implement such a modification, Kaa distributes *configuration override data* to the endpoint group. The structure of the configuration override data is specified by the *override schema*, which is derived from the configuration schema used by the target endpoint group. As different from the base schema, the override schema allows updating fields selectively, leaving some of them without any change if necessary (using the org.kaaproject.configuration.unchangedT type).

Override schema

The override schema is obtained by transforming the configuration schema and looks very similar to the base schema. The difference is that when constructing an override schema, Kaa adds org.kaaproject.configuration.unchangedT to every field type definition (converting them into union types, if necessary). Thus, all mandatory fields become union types with either their original type (if there is any data change) or the org.kaaproject.configuration.unchangedT type (if there is no data change). When org.kaaproject.configuration.unchangedT appears in the data, it indicates that the corresponding field value remains without the change during the update.

As an example, consider how mandatory and optional fields are converted from the configuration schema to the override schema.

Configuration schema	Override schema

Mandatory field:

```
{
    "name":
"stringField",
    "type": "string",
    "by_default":
"default string value"
}
```

becomes:

```
{
    "name": "stringField",
    "type": [
        "string",
        {
            "name": "unchangedT",
            "namespace":
"org.kaaproject.configuration",
            "type": "enum",
            "symbols": ["unchanged"]
        }
    ],
    "by_default": "default string value"
}
```

Optional field:

```
{
    "name":
"optionalBytesField",
    "type": [
        "null",
        "bytes"
    ],
    "optional": true
}
```

becomes:

```
{
    "name": "optionalBytesField",
    "type": [
        "null",
        "bytes",
        "org.kaaproject.configuration.unchangedT"
    ],
    "optional": true
}
```

Loading the override data into the endpoint groups is similar to loading base data into group "all". Among other parameters, the group ID must be passed to the API call to indicate the group to which the new data should be applied. The loading algorithm processes the record UUID values identically to how it is done for the "all" group, persisting the UUID values that already existed in the previous version of the data in the processed group. Loading is done for the group in question independently of any other groups and without any cross-group data lookups.

Override algorithm

To define and apply the resulting configuration update for the endpoint, the Operations server proceeds as follows:

1. Evaluates the endpoint group membership according to the endpoint profile.
2. Merges all configuration data sets assigned to the groups the endpoint belongs to, starting with the one that has the lowest weight (which is always group "all" with the 0 weight).
   In case of the conflicting field values, the field is set with the the value from the group with the highest weight.

NOTE:
In case of the arrays, the **overrideStrategy** field in the configuration schema defines the way in which the arrays are merged.
Record UUID fields never change from the values in the lowest weight group they were first encountered in.
Both profile schema and configuration schema versions the endpoint supports are taken into account during the merge.

User-specific configuration management

The user-specific configuration management allows updating configuration data for the specific user under the application. The user-specific configuration management implements the same approach as in Group-specific configuration management, based on the override schema and override algorithm.

NOTE:

Since each endpoint belongs to only one user at a time, the override algorithm does not support data set merges as in the group-specific configuration management.
The override algorithm applies user-specific overrides only after all group-specific overrides have been applied.

Endpoint data synchronization

After calculating the up-to-date configuration for the endpoint, the Operations server constructs a delta update based on the knowledge of the prior configuration of that endpoint. In certain cases, the delta update can contain a complete endpoint configuration.

Having received the delta update, the endpoint merges it with the existing configuration data, notifies the client application about the update, and persists the resulting configuration. The data storage location is set as an abstraction in the endpoint, therefore the concrete location for persisting the data is defined in the client implementation.

By delivering configuration deltas instead of full configuration sets each time the endpoint needs to be reconfigured, the Kaa server significantly reduces a traffic load in the available data channels.

Data consistency is ensured by the hash comparison between the endpoint and the server.

Protocol schema

A *protocol schema* determines the structure of the data updates that Kaa server sends to the endpoints and possible update actions (change, add, reset, and leave unchanged). The protocol schema is automatically generated by the Control server for each configuration schema (when the configuration schema is loaded by the user).

The updates may carry either a full set of data (full sync) or a partial update (configuration delta). Full sync is used in exceptional situations (e.g., a hash mismatch), with the corresponding protocol schema being very similar to the base schema.

The schema generator performs the following transformations to convert the configuration schema to the protocol schema.

- The __uuid field of the org.kaaproject.configuration.uuidT type is added to every addressable record. Having a UUID ensures that the record becomes addressable and the Kaa server can send partial updates on such records.
- Optional fields are transformed into unions with either the original field type, null type or org.kaaproject.configuration.unchangedT type. (*TODO* is this only for optional or for any unions with null?)
  The org.kaaproject.configuration.unchangedT type is used for a simple enum that indicates that there is no change for the field in the current update. It has the following structure.

```
{
    "name": "unchangedT",
    "namespace": "org.kaaproject.configuration",
    "type": "enum",
    "symbols": ["unchanged"]
}
```

- Mandatory fields are transformed into unions with either the original field type or the org.kaaproject.configuration.unchangedT type.
- Array fields are transformed into unions, as follows:
  - an array of:
    - the original array item type to add new items to the array
    - the org.kaaproject.configuration.uuidT type if the original item type is a UUID-addressable record
  - the org.kaaproject.configuration.unchangedT type to indicate no changes to the array contents
  - the org.kaaproject.configuration.resetT type to purge the array contents. Similarly to org.kaaproject.configuration.unchangedT, org.kaaproject.configuration.resetT is a simple enum, as follows:

```
{
    "name": "resetT",
    "namespace": "org.kaaproject.configuration",
    "type": "enum",
    "symbols": ["reset"]
}
```

- The root record schema is wrapped into an array of org.kaaproject.configuration.deltaT records. The org.kaaproject.configuration.deltaT record contains the only one field, which is called delta. The delta field is a union of the

transformed data root record schema and every UUID-addressable record type encountered in the root record schema. Such a structure allows encoding delta updates to every UUID-addressable record in the schema, including the root record.

```
{
    "type": "array"
    "items": {
        "name": "deltaT",
        "namespace": "org.kaaproject.configuration"
        "type": "record",
        "fields": [
            {
                "name": "delta",
                "type": [
                    <root record schema>,
                    <list of addressable record types>
                ]
            }
        ],
    },
}
```

As an example, consider transformation of the following record.

Configuration schema	Protocol schema (transformed)

	<pre>{
	    "type": "array",
	    "items": {
	        "name": "deltaT",
	        "namespace": "org.kaaproject.c
	        "type": "record",
	        "fields": [
	            {
	                "name": "delta",
	                "type": [
	                    {
	                        "name": "rootT
	                        "namespace": "
	                        "type": "recor
	                        "fields": [
	                            {
	                                "name"
	                                "type"
	                                    {</pre>

```
{
    "name": "rootT",
    "namespace": "org.kaaproject.sample",
    "type": "record",
    "fields": [
        {
            "name": "arrayOfRecords",
            "type": {
                "type": "array",
                "items": {
                    "name":
"addressableRecordT",
                    "namespace":
"org.kaaproject.sample",
                    "type": "record",
                    "fields": [
                        {
                            "name":
"booleanField",
                            "type":
"boolean",
                            "by_default":
false
                        }
                    ]
                }
            }
        },
        {
            "name": "arrayOfPrimitives",
            "type": {
                "type": "array",
                "items": {
                    "name":
"primitiveRecordT",
                    "namespace":
"org.kaaproject.sample",
                    "type": "record",
                    "addressable": false,
                    "fields": [
                        {
                            "name":
"intField",
                            "type":
"int",
                            "optional":
true
                        }
                    ]
                }
            }
        }
    ]
}
```

```
"org.kaaproject.configuration",

"org.kaaproject.configuration",

                                            },
                                            {

                                            },
                                            "o
                                        ]
                                    },
                                    {
                                        "name"
                                        "type"
                                            {

                                            },
                                            "o
                                            "o
                                        ]
                                    },
                                    {
```

```
                        "name"
                        "type"
                    }
                ]
            },
            "org.kaaproject.sa
        ]
    }
```

```
            ]
        }
    }
```

Note:

- The __uuid field was added to org.kaaproject.sample.addressableRecordT, but not to org.kaaproject.sample.primitiveRecordT.
- There is a reference to org.kaaproject.sample.addressableRecordT in the union definition for the delta field of the org.kaaproject.configuration.deltaT type.
- org.kaaproject.configuration.uuidT is used in a union to indicate the removal of UUID-addressable records in /arrayOfRecords, but not in /arrayOfPrimitives.

Delta calculation algorithm

The delta calculation algorithm is executed by the Operations server to discover and represent via the protocol schema the difference between the newly required configuration and the current one in regard to the specific endpoint. This difference is represented according to the corresponding protocol schema and then supplied to the endpoint as the *delta configuration data* (other accepted terms for which are delta, delta update, delta record). Note that both configurations should conform to the same configuration schema version.

To keep track of the current endpoint configuration, the Kaa server maintains a hash look-up table that contains the configuration SHA-1 hash, as well as the configuration data itself. Whenever the Operations server calculates a configuration with the unique hash, it adds that configuration to the table which is persisted in the database and shared across the Operations servers. The endpoint submits the hash of the last known configuration to Operations server for the table lookup.

Having found the current endpoint configuration via the hash look-up, the Operations server starts comparing the current configuration set with the new one to detect any discrepancies between the two. Once all the differences are found, the Operations server looks up the nearest UUID-addressable record in the addressing hierarchy and creates a delta record for it. After that, the algorithm proceeds to comparing the corresponding records in the two data sets and filling in the delta record. In case some field has the same value in both configuration sets, this field is assigned the unchanged type value ("unchangedT") in the delta record to indicate so. Otherwise, the value from the new configuration set is copied into the delta record.

The described traversal method works for all addressable fields in the configuration. However, items in the arrays are not addressable, therefore Kaa employs special handling logic for calculating deltas for the array fields, as follows:

- If the array items are UUID-addressable, the algorithm looks for the items with matching UUIDs in the two configuration sets and creates a separate delta for these items.
  If the new configuration set misses some of the items, the algorithm proceeds as follows according to the item type:
  - for primitive items, the "reset" value is used in the first delta update to erase all the array items, whereupon the second delta update delivers all the items which should be kept and adds them to the previously emptied array.
  - for records, the UUIDs of the missing records are added to the "remove" list for this field in the protocol schema. If all of the old records are absent in the new configuration, then, instead of listing all UUIDs in the "remove" list, the algorithm uses the "reset" value as for primitive types.

  If the new configuration set includes some of the items that the current configuration set does not contain, the items that does not exist in the current configuration are added to the delta update for appending to the array on the client side (this is valid for both primitive types and records).
- If array items are not UUID-addressable and the algorithm detects any difference between the old and new array contents, the old contents gets reset, and the new contents are sent together with the configuration delta.
- If array items have different types in the old and new configurations, the server resets the array first and then populates another delta message with the most recent items.

As an example, consider the following configuration schema.

```
{
    "name": "testT",
    "type": "record",
    "addressable": true,
    "fields": [
        {
            "optional": true,
            "name": "testField1",
            "type": "string"
        },
        {
            "name": "testField2",
            "type": {
                "addressable": false,
                "type": "record",
                "name": "testRecordT",
                "namespace": "org.kaa.config",
                "fields": [
                    {
                        "name": "testField3",
                        "type": {
                            "type": "array",
                            "items": {
                                "type": "record",
                                "name": "testRecordItemT",
                                "namespace": "org.kaa.config",
                                "fields": [
                                    {
                                        "name": "testField4",
                                        "type": "int"
                                    }
                                ]
                            }
                        }
                    }
                ]
            }
        },
        {
            "optional": true,
            "name": "testField5",
            "type": "int"
        }
    ],
    "namespace": "org.kaa.config"
}
```

In the scope of the example, let's assume that the Operations server needs to calculate the delta between the following two configuration sets.

Current configuration (on the endpoint)

```
{
    "testField1": {
        "string": "abc"
    },
    "testField2": {
        "testField3": [
            {
                "testField4": 1,
                "__uuid": {
                    "org.kaaproject.configuration.uuidT":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u00
                }
            },
            {
                "testField4": 2,
                "__uuid": {
                    "org.kaaproject.configuration.uuidT":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u00
                }
            },
            {
                "testField4": 3,
                "__uuid": {
                    "org.kaaproject.configuration.uuidT":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u00
                }
            }
        ]
    },
    "testField5": {
        "int": 123
    },
    "__uuid": {
        "org.kaaproject.configuration.uuidT":
"\u0001\u0002\u0003\u0004\u0005\u0006\u0007\b\t\n\u000b\f\r\u000e\u000f\u0010"
    }
}
```

Compared to the current endpoint configuration, the new configuration on the server has the following differences.

- The **testField5** field is set to the default value.
- The element with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3] has a new value for the **testField4** field.
- The element with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] was removed.
- A new element with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4] was added.

The calculated delta update will be an array with three entries, as follows:

```
[
    {
        "delta": {
            "org.kaa.config.testRecordItemT": {
                "testField4": {
                    "int": 36
                },
                "__uuid":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u000
0\u0000\u0003"
```

```
            }
        }
    },
    {
        "delta": {
            "org.kaa.config.testT": {
                "testField1": {
                    "org.kaaproject.configuration.unchangedT": "unchanged"
                },
                "testField2": {
                    "org.kaa.config.testRecordT": {
                        "testField3": {
                            "array": [
                                {
                                    "org.kaaproject.configuration.uuidT":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u000
0\u0000\u0001"
                                }
                            ]
                        }
                    }
                },
                "testField5": {
                    "org.kaaproject.configuration.unchangedT": "unchanged"
                },
                "__uuid":
"\u0001\u0002\u0003\u0004\u0005\u0006\u0007\b\t\n\u000b\f\r\u000e\u000f\u0010"
            }
        }
    },
    {
        "delta": {
            "org.kaa.config.testT": {
                "testField1": {
                    "org.kaaproject.configuration.unchangedT": "unchanged"
                },
                "testField2": {
                    "org.kaa.config.testRecordT": {
                        "testField3": {
                            "array": [
                                {
                                    "org.kaa.config.testRecordItemT": {
                                        "testField4": {
                                            "int": 4
                                        },
                                        "__uuid":
"\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u000
0\u0000\u0004"
                                    }
                                }
                            ]
                        }
                    }
                },
                "testField5": null,
                "__uuid":
"\u0001\u0002\u0003\u0004\u0005\u0006\u0007\b\t\n\u000b\f\r\u000e\u000f\u0010"
            }
```

```
        }
    }
]
```

The first delta item is a partial update for the object with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3], where the **testField4** field value was changed from 3 to 36. The second delta item is an update for the **testField3** field to remove the item with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] (**NOTE**: the **testField1** and **testField5** fields were set to "unchanged"). The last delta item sets the **testField5** field to null and adds a new array item with UUID [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4].

Schema versioning

In the current Kaa version, any configuration schema updates require updates of the client application.

Therefore, to enable the server compatibility with the older clients as the configuration schema evolves, Kaa servers are capable of maintaining more than one configuration schema version. For this purpose, a new, sequentially incremented version number is automatically assigned to every new configuration schema loaded into the Kaa server. Configuration data is managed independently for every schema version.

The endpoints report their supported configuration data schema version in the service profile. The Kaa server, in response, supplies them with the data updates that correspond to the reported version.
