# Distributing data to endpoints



The Kaa configuration subsystem supplies endpoints with the structured configuration data of custom complexity that is managed through the Kaa server. The configuration data itself is structured according to the configuration schema specified by the user beforehand. Both a configuration schema and configuration data can be modified on the server using Admin UI or REST API and then uploaded to relevant applications.

## Configuring

This section provides guidance on how to create a configuration schema in Kaa and use it for loading data into applications.

### Configuration schema

The structure of the data in Kaa is determined by the configuration schema which is created in the Avro format. See the Configuration section in the Design reference for more information on the schema format.

In this guide, we are going to use the following configuration schema for our example scenario.

This schema allows you to configure a global log level (the **global_log_level** field) for your project and an individual log level per each project module.

```
{
   "type":"record",
   "name":"LogConfiguration",
   "namespace":"org.kaaproject.kaa.schema.sample",
   "fields":[
      {
         "name":"global_log_level",
         "type":{
            "type":"enum",
            "name":"LogLevel",
            "namespace":"org.kaaproject.kaa.schema.sample",
            "symbols":[
               "FATAL",
               "ERROR",
               "WARNING",
               "INFO",
               "DEBUG",
               "TRACE"
            ]
         }
      },
      {
         "name":"modules",
         "type":{
            "type":"array",
            "items":{
               "type":"record",
               "name":"ModuleLogLevel",
               "namespace":"org.kaaproject.kaa.schema.sample",
               "fields":[
                  {
                     "name":"module_name",
                     "type":"string"
                  },
                  {
                     "name":"log_level",
                     "type":"org.kaaproject.kaa.schema.sample.LogLevel"
                  }
               ]
            }
         }
      }
   ]
}
```

## Upload configuration schema using UI/REST

Refer to Admin UI or REST API guides for instructions on uploading the configuration schema to the server.

## Base and override schemas

After the configuration schema has been successfully uploaded, Kaa generates a *base* and an *override* schemas as described in the Configuration section of the Design reference.

The main difference between these two schemas is that using an override schema you can set some fields as *unchanged* rather than assigning them specific values. For the unchanged field, its value will be taken from the corresponding field in the All group, if it's possible. If not (for example, if there's a new array item which is not present in the All group), Kaa will generate the default value for the unchanged field as described in the Configuration section of the Design reference.

You can obtain all these configuration schemas using REST API.

> **NOTE:** Admin UI currently does not allow you to obtain base and override schemas. ( **KAA-208** - Getting issue details... STATUS tracks this issue)

The base schema generated from the configuration schema in our example looks as follows:

```
{
   "type":"record",
   "name":"LogConfiguration",
   "namespace":"org.kaaproject.kaa.schema.sample",
   "fields":[
      {
         "name":"global_log_level",
         "type":{
            "type":"enum",
            "name":"LogLevel",
            "namespace":"org.kaaproject.kaa.schema.sample",
            "symbols":[
               "FATAL",
               "ERROR",
               "WARNING",
               "INFO",
               "DEBUG",
               "TRACE"
            ]
         }
      },
      {
         "name":"modules",
         "type":{
            "type":"array",
            "items":{
               "type":"record",
               "name":"ModuleLogLevel",
               "namespace":"org.kaaproject.kaa.schema.sample",
               "fields":[
                  {
                     "name":"module_name",
                     "type":"string"
                  },
                  {
                     "name":"log_level",
                     "type":"org.kaaproject.kaa.schema.sample.LogLevel"
                  },
                  {
                     "name":"__uuid",
                     "type":[
                        {
                           "name":"uuidT",
                           "type":"fixed",
                           "size":16,
                           "namespace":"org.kaaproject.configuration"
                        },
                        "null"
                     ]
                  }
               ]
            }
         }
      },
      {
         "name":"__uuid",
         "type":[
            "org.kaaproject.configuration.uuidT",
            "null"
         ]
      }
   ]
}
```

The override schema generated from the configuration schema in our example looks as follows:

```
{
   "type":"record",
```

```
"name":"LogConfiguration",
"namespace":"org.kaaproject.kaa.schema.sample",
"fields":[
   {
      "name":"global_log_level",
      "type":[
         {
            "type":"enum",
            "name":"LogLevel",
            "namespace":"org.kaaproject.kaa.schema.sample",
            "symbols":[
               "FATAL",
               "ERROR",
               "WARNING",
               "INFO",
               "DEBUG",
               "TRACE"
            ]
         },
         {
            "symbols":[
               "unchanged"
            ],
            "name":"unchangedT",
            "type":"enum",
            "namespace":"org.kaaproject.configuration"
         }
      ]
   },
   {
      "name":"modules",
      "type":[
         {
            "type":"array",
            "items":{
               "type":"record",
               "name":"ModuleLogLevel",
               "namespace":"org.kaaproject.kaa.schema.sample",
               "fields":[
                  {
                     "name":"module_name",
                     "type":[
                        "string",
                        "org.kaaproject.configuration.unchangedT"
                     ]
                  },
                  {
                     "name":"log_level",
                     "type":[
                        "org.kaaproject.kaa.schema.sample.LogLevel",
                        "org.kaaproject.configuration.unchangedT"
                     ]
                  },
                  {
                     "name":"__uuid",
                     "type":[
                        {
                           "name":"uuidT",
                           "type":"fixed",
                           "size":16,
                           "namespace":"org.kaaproject.configuration"
                        },
                        "null"
                     ]
                  }
               ]
            }
         },
         "org.kaaproject.configuration.unchangedT"
      ]
   },
```

```
      {
         "name":"__uuid",
         "type":[
            "org.kaaproject.configuration.uuidT",
            "null"
         ]
      }
   ]
}
```

## Load default data into All group

As soon as both schemas are generated, Kaa loads the default data into the All group using the base schema. The default data is constructed according to the field types as described in the Configuration section of the Design reference. In our example, the generated data looks as follows:

```
{
   "global_log_level":"FATAL",
   "modules":[

   ],
   "__uuid":{
      "org.kaaproject.configuration.uuidT":"[6\u0001°SJ<¼(\u000Fõé\u0015\u0018"
   }
}
```

## Edit configuration data for All group

Use the base schema when you edit the configuration data for the All group.

To edit the configuration data for the All group, proceed as follows:

1. Obtain the existing configuration data using the Admin UI (select the All endpoint group under the application, then select the endpoint group configuration) or REST API.
2. In our example, let's set the **global_log_level** field to "TRACE" .
   This global log level will become default for all endpoints.

   **IMPORTANT**: the **__uuid** field is reserved for Kaa needs. Never edit its value by yourself.

   The resulting configuration data looks as follows:

   ```
   {
      "global_log_level":"TRACE",
      "modules":[

      ],
      "__uuid":{
         "org.kaaproject.configuration.uuidT":"[6\u0001°SJ<¼(\u000Fõé\u0015\u0018"
      }
   }
   ```

3. To save the changes, click **Save** in Admin UI or use REST API.
4. To apply the saved changes to endpoints, activate the new configuration by clicking **Activate** in Admin UI or using REST API.

## Load configuration data into specific group

To create an individual configuration for a specific endpoint group, use the override schema.

In the following configuration example for the *Android Froyo endpoints* group, we've set the **global_log_level** field to "unchanged" (that is, the default value from the All group will be used) and assigned the "DEBUG" level to the **org.kaaproject.kaa.schema.sample.FroyoSpecificModule** module.

```
{
   "global_log_level":{
      "org.kaaproject.configuration.unchangedT":"unchanged"
   },
   "modules":{
      "array":[
         {
            "module_name":{
               "string":"org.kaaproject.kaa.schema.sample.FroyoSpecificModule"
            },
            "log_level":{
               "org.kaaproject.kaa.schema.sample.LogLevel":"DEBUG"
            },
            "__uuid":null
         }
      ]
   },
   "__uuid":null
}
```

In the following configuration example for the *Android endpoints* group, we've assigned the 'INFO" level to the **org.kaaproject.kaa.schema.sample.AndroidModule** module.

```
{
   "global_log_level":{
      "org.kaaproject.configuration.unchangedT":"unchanged"
   },
   "modules":{
      "array":[
         {
            "module_name":{
               "string":"org.kaaproject.kaa.schema.sample.AndroidModule"
            },
            "log_level":{
               "org.kaaproject.kaa.schema.sample.LogLevel":"INFO"
            },
            "__uuid":null
         }
      ]
   },
   "__uuid":null
}
```

Please note that we've left the null value for all __**uuid** fields.

To upload a new configuration for each group, use Admin UI or REST API. Don't forget to activate each configuration as described in the *Edit configuration data for All group* section.

## Edit configuration data for specific group

Editing process for a specific group is similar to editing for the All group described in the *Edit configuration data for All group* section.

1. Obtain the current configuration data using the Admin UI (select the endpoint group under the application, then select the endpoint group configuration) or REST API.
   For the *Android Froyo endpoints* group, the result is presented in the following code block. It includes the data we've added in the *Load configuration data into specific group* section, as well as the __**uuid** values automatically generated by Kaa after we've uploaded our configuration to the server.

   **NOTE**: Actual __**uuid** values may differ from those provided in this sample.

```
{
   "global_log_level":{
      "org.kaaproject.configuration.unchangedT":"unchanged"
   },
   "modules":{
      "array":[
         {
            "module_name":{
               "string":"org.kaaproject.kaa.schema.sample.FroyoSpecificModule"
            },
            "log_level":{
               "org.kaaproject.kaa.schema.sample.LogLevel":"DEBUG"
            },
            "__uuid":{
               "org.kaaproject.configuration.uuidT":"=ºÞNEµ½ ¯]\"Ê\u001A"
            }
         }
      ]
   },
   "__uuid":{
      "org.kaaproject.configuration.uuidT":"×dëîCrG®auòë\u001D\u0017"
   }
}
```


2. In our example, let's assign the "WARNING" value to the **org.kaaproject.kaa.schema.sample.FroyoSpecificModule** module.


```
{
   "global_log_level":{
      "org.kaaproject.configuration.unchangedT":"unchanged"
   },
   "modules":{
      "array":[
         {
            "module_name":{
               "string":"org.kaaproject.kaa.schema.sample.FroyoSpecificModule"
            },
            "log_level":{
               "org.kaaproject.kaa.schema.sample.LogLevel":"WARNING"
            },
            "__uuid":{
               "org.kaaproject.configuration.uuidT":"=ºÞNEµ½ ¯]\"Ê\u001A"
            }
         }
      ]
   },
   "__uuid":{
      "org.kaaproject.configuration.uuidT":"×dëîCrG®auòë\u001D\u0017"
   }
}
```


3. Load this data back to the server and activate the new configuration.

```
 {
   "global_log_level":{
      "org.kaaproject.configuration.unchangedT":"unchanged"
   },
   "modules":{
      "array":[
         {
            "module_name":{
               "string":"org.kaaproject.kaa.schema.sample.FroyoSpecificModule"
            },
            "log_level":{
               "org.kaaproject.kaa.schema.sample.LogLevel":"WARNING"
            },
            "__uuid":{
               "org.kaaproject.configuration.uuidT":"=ºÞNEµ½ ¯]\"Ê\u001A"
            }
         }
      ]
   },
   "__uuid":{
      "org.kaaproject.configuration.uuidT":"×dëîCrG®auòë\u001D\u0017"
   }
}
```

## Coding

This section provides code samples which illustrate how to use Kaa SDK to work with configuration schema/data in Kaa, for instance, how to subscribe to configuration updates and implement configuration schema persistence.

### Configuration traversal

You can upload the current configuration data to the client and use this data for specific purposes.

The following example illustrates how to upload the configuration data to the client, use it for retrieving log levels of each project module, and check whether a specific module is loggable against the provided log level.

```
    public enum LogLevel {
       FATAL,
       ERROR,
       WARNING,
       INFO,
       DEBUG,
       TRACE
   }
```

```
public class Logger {
    private LogLevel rootLogLevel = LogLevel.INFO;
    private Map<String, LogLevel> modulesLevels = new HashMap<>();

    public void initFromConfiguration(Kaa kaa) {

        KaaClient client = kaa.getClient();
        CommonRecord configuration = client.getConfigurationManager().getConfiguration();
        CommonValue globalLogLevelValue = configuration.getField("global_log_level");
        String logLevelSymbol = globalLogLevelValue.getEnum().getSymbol();
        rootLogLevel = LogLevel.valueOf(logLevelSymbol);

        modulesLevels.clear();

        CommonValue modulesValue = configuration.getField("modules");
        List<CommonValue> modulesList = modulesValue.getArray().getList();
        for (CommonValue moduleValue : modulesList) {
            CommonRecord moduleRecord = moduleValue.getRecord();
            String moduleName = moduleRecord.getField("module_name").getString().toString();
            String moduleLogLevelSymbol = moduleRecord.getField("log_level").getEnum().getSymbol();
            LogLevel moduleLogLevel = LogLevel.valueOf(moduleLogLevelSymbol);
            modulesLevels.put(moduleName, moduleLogLevel);
        }
    }
    public boolean isLoggable(Class<?> clazz, LogLevel logLevel) {
        String packageName = clazz.getPackage().getName();
        LogLevel moduleLogLevel = modulesLevels.get(packageName);
        if (moduleLogLevel == null) {
            moduleLogLevel = rootLogLevel;
        }
        return logLevel.ordinal() <= moduleLogLevel.ordinal();
    }
}
```

## Subscribe to configuration updates

You can subscribe your own callback on any configuration update, as illustrated by the following code example.

The onConfigurationUpdated method will be invoked on any configuration update that was received from the server. The arg0 argument contains the full latest configuration data.

```
       KaaDesktop kaa = new KaaDesktop();
       kaa.getClient().getConfigurationManager().subscribeForConfigurationUpdates(new ConfigurationReceiver()
{


            @Override
            public void onConfigurationUpdated(CommonRecord arg0) {
                System.out.println("Configuration updated");
            }

        });
```

## Subscribe to configuration updates for specific objects

You can change the client's configuration subscription to receive updates only on specific parts of configuration.

To use the partial configuration subscription, proceed as follows:

1. Implement the root configuration receiver using the DeltaReceiver (java) interface.
   The LogLevelHandler class is a mapping to the LogConfiguration record from the schema. It consists of the global log level value and modules.

```
   public class LogLevelHandler implements DeltaReceiver {

        private final Map<DeltaHandlerId, ModuleLogLevelReceiver> modules = new
HashMap<DeltaHandlerId, ModuleLogLevelReceiver>();
        private final DeltaManager deltaManager;
        private LogLevel globalLogLevel;

        public LogLevelHandler(DeltaManager manager) {
            this.deltaManager = manager;
        }

        @Override
        public synchronized void loadDelta(ConfigurationDelta delta) {
            if (delta.hasChanged("global_log_level")) {
                // "global_log_level" field was changed
                // applying the new value
                DeltaType globalLevelDeltaType = delta.getDeltaType("global_log_level");
                globalLogLevel = LogLevel.valueOf((String) globalLevelDeltaType.getNewValue());
            }
            if (delta.hasChanged("modules")) {
                DeltaType modulesDeltaType = delta.getDeltaType("modules");
                if (modulesDeltaType.isReset()) {
                    // Array object was cleared
                    // unsubscribing each element from updates
                    for (DeltaHandlerId id : modules.keySet()) {
                        deltaManager.unsubscribeFromDeltaUpdates(id);
                    }
                    modules.clear();
                } else {
                    List<DeltaHandlerId> removedDeltas = modulesDeltaType.getRemovedItems();
                    if (removedDeltas != null) {
                        for (DeltaHandlerId id : removedDeltas) {
                            // Configuration object was removed
                            // unsubscribing it from updates
                            deltaManager.unsubscribeFromDeltaUpdates(id);
                            modules.remove(id);
                        }
                    }
                    List<Object> addedDeltas = modulesDeltaType.getAddedItems();
                    if (addedDeltas != null) {
                        for (Object item : addedDeltas) {
                            // The new object was added
                            ConfigurationDelta itemDelta = (ConfigurationDelta) item;
                            ModuleLogLevelReceiver newReceiver = new ModuleLogLevelReceiver();
                            // Applying delta for the new object
                            newReceiver.loadDelta(itemDelta);
                            // Subscribing the new object for partial updates
                            deltaManager.subscribeForDeltaUpdates(itemDelta.getHandlerId(),
newReceiver);
                            modules.put(itemDelta.getHandlerId(), newReceiver);

                        }
                    }
                }
            }
        }

        public synchronized LogLevel getGlobalLogLevel() {
            return globalLogLevel;
        }

        public synchronized LogLevel getModuleLogLevel(String moduleName) {
            for (ModuleLogLevelReceiver receiver : modules.values()) {
                if (receiver.getModuleName().equals(moduleName)) {
                    return receiver.getLogLevel();
                }
            }
            return null;
        }
    }
```

2. Implement the ModuleLogLevelReceiver class which represents the ModuleLogLevel record from the schema.

   Objects of this class can process partial configuration updates for the individual item in the array.

```
public class ModuleLogLevelReceiver implements DeltaReceiver {
    private String moduleName;
    private LogLevel logLevel;

    @Override
    public void loadDelta(ConfigurationDelta delta) {
        if (delta.hasChanged("module_name")) {
            // "module_name" field was changed
            // applying the new value
            moduleName = (String) delta.getDeltaType("module_name").getNewValue();
        }
        if (delta.hasChanged("log_level")) {
            // "log_level" field was changed
            // applying the new value
            DeltaType logLevelType = delta.getDeltaType("log_level");
            logLevel = LogLevel.valueOf((String) logLevelType.getNewValue());
        }
    }

    public String getModuleName() {
        return moduleName;
    }

    public LogLevel getLogLevel() {
        return logLevel;
    }
}
```

3. To start using the partial subscription mechanism, register the root receiver in Kaa.

```
    KaaDesktop kaa = new KaaDesktop();
    KaaClient client = kaa.getClient();
    DeltaManager deltaManager = client.getDeltaManager();
    deltaManager.registerRootReceiver(new LogLevelHandler(deltaManager));
```

## Configuration schema persistence

During the work, the Kaa client can receive configuration schema updates. By default, the schema is not persisted by the client. You can create your own mechanism of schema persistence by creating the instance of SchemaStorage class and passing it as a parameter to the setSchemaStorage method of the SchemaPersistetManager class.

The following code example illustrates how to persist the schema by storing it to a file.

```
        KaaDesktop kaa = new KaaDesktop();
        KaaClient client = kaa.getClient();
        client.getSchemaPersistenceManager().setSchemaStorage(new SchemaStorage() {

            @Override
            public void saveSchema(ByteBuffer buffer) {
                try {
                    FileOutputStream fileStream = new FileOutputStream(new File("/path/to/schema"));
                    fileStream.write(buffer.array());
                    fileStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public ByteBuffer loadSchema() {
                try {
                    byte[] schema = Files.readAllBytes(Paths.get("/path/to/schema"));
                    return ByteBuffer.wrap(schema);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return null;
            }
        });
```

## Configuration data persistence

Similarly to the configuration schema persistence, you can persist the configuration data too. For this purpose, create the instance of ConfigurationStorage class and pass it as a parameter to the setConfigurationStorage method of the ConfigurationPersistenceManager class.

The following code example illustrates how to persist the configuration data by storing it to a file.

```
        KaaDesktop kaa = new KaaDesktop();
        KaaClient client = kaa.getClient();
                client.getConfigurationPersistenceManager().setConfigurationStorage(new
ConfigurationStorage() {

                        @Override
            public void saveConfiguration(ByteBuffer buffer) {
                try {
                    FileOutputStream fileStream = new FileOutputStream(new File("/path/to/configuration"));
                    fileStream.write(buffer.array());
                    fileStream.close();
                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            @Override
            public ByteBuffer loadConfiguration() {
                try {
                    byte[] data = Files.readAllBytes(Paths.get("/path/to/configuration"));
                    return ByteBuffer.wrap(data);
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return null;
            }
        });
```