Data Migration Guide

ARIS Risk & Compliance Manager
Version 97

October 2014

Product Documentation

Contents

1 Introduction 1

2 The ARCM migration framework 2

2.1 What the framework cannot do 2

3 Start the migration 3

4 The migration plan 4

4.1 Format 4
4.2 The XML schema of the migration plan 5
4.3 The location of the migration plan 5

5 The architecture 6

5.1 The construction set 8

5.1.1 IMigrationStep 9
5.1.2 Step template 9
5.1.3 IMapping 10

6 MigrationObject 11

7 Automatic update of the schema version 12

8 Adjustments to the data migration in CTK 13

9 Logging 14

1 Introduction

As of version 4.0 of ARIS Risk & Compliance Manager the server has a migration framework for the incremental migration of data from previous versions. This document provides you with an introduction into the handling, operation and extension possibilities of this framework. It is oriented towards all developers who adapt ARIS Risk & Compliance Manager to specific customer requirements and are in charge of data migration.

If an updated version of this document is available, you will find it here: http://aris.softwareag.com/ARISDownloadCenter/ADCDocumentationServer (http://aris.softwareag.com/ARISDownloadCenter/ADCDocumentationServer)

2 The ARCM migration framework

The server has a mechanism for the incremental migration of data from previous versions. With it data can be migrated in all of the database systems (Oracle, MSSQL, and Derby) supported by ARIS Risk & Compliance Manager. In addition, a portfolio of high-level API functions is available. These functions can be used within migration steps that you can write or change yourself to add tables and fields and to adapt data to new requirements.

The framework can be extended with Java classes. These must be available in text form. A compilation is not required. The internal migration logic determines the steps required to adapt data structures and data on the currently started server using an individually customizable XML migration plan.

Warning

In order to prevent data loss and irreversible changes to your data we recommend that you perform a full backup of all of your data. To do so, please use the administrative tool from your database system.

2.1 What the framework cannot do

The internal migration framework only processes data and its structures. This is sufficient for migrating the standard version of ARIS Risk & Compliance Manager. However, it cannot adapt internal logic such as rules or workflows. This means for customer-specific adapted versions that in addition to the data migration mentioned here, the logic may have to be adapted.

3 Start the migration

To start the migration, set the **dbms.autoStartMigration** parameter to **true** in the **runtimeconfig.xml** configuration file.

Example

```
<!--if automatic migration should run during server startup set this parameter to true-->
<parameter name="dbms.autoStartMigration" value="true"/>
```

If this parameter is set to **true**, then it is checked at server start whether the schema version of the ARCM user in the connected database coincides with the current schema version of the ARCM server. If, during this check, the server detects that the server version does not coincide with the database version it tries to generate an appropriate migration strategy using the migration plan (Page 4).

As long as the parameter is set to **true** a test system overlay is displayed on the start screen of the system and in the header. This overlay is removed by switching off the parameter.

Now a productive environment can be generated via a database export and import. To do so, generate a new schema on the target database and have the required tables generated by the starting the ARIS Risk & Compliance Manager server. A database import can also be imported into another DBMS (database management system). It is for example, possible to go live on an MSSQL server with a database that was migrated under Oracle.

4 The migration plan

The migration plan is an XML file with the name **migrationPlan.xml**. Here a version transition is assigned to a directory in which the corresponding migration logic is located.

4.1 Format

A version transition with the following attributes is defined in the **migration** tag:

- **Name**: contains the name of the version transition. This name is entered into the **A_SCHEMAPROPERTY_TBL** table when the version transition is carried out. Later, the migration history of the database can be tracked with this information.
- **Source**: contains the start version located in the **A_SCHEMAPROPERTY_TBL** table of the connected database system. If the start version is not specified the **start** value can be entered here. In this case, if an appropriate source to a schema is not found, the migration is started with the version transition marked as **start** in the plan.
- **Target**: contains the target version to be reached after the current version transition.
- **Approach**: outputs whether the current version transition refers to the risk-based or the control-based approach. Possible values are **rba** and **cba**.
- **Implementation**: the folder that contains the migration logic for this version transition. The path to this folder is composed of two components. The first part is the general **SourceFolder [installation Directory] / jsp / WEB-INF / config / migration**. The second part is composed of the Java packages that create a path to the basis package **com.idsscheer.webapps.arcm.dl.datamigration**. The **implementation** folder is now saved under this basis package. This path lies outside the ARCM library and can therefore be extended with your own classes and resources that do not need to be compiled.
- **Fix**: If this optional attribute is set to the value **true** the specified version transition will be executed as a hotfix on an existing version. The database version remains unchanged. Here the attributes **source** and **target** must contain the same version.

Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<migrationPlan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="./xsd/migrationPlan.xsd">
   <migration name="start_to_4.0_RBA" source="start"
target="arcm_4.0.0_rba_standard" approach="rba" implementation="mig31SR4To40"/>
   <migration name="start_to_4.0_CBA" source="start"
target="arcm_4.0.0_cba_standard" approach="cba" implementation="mig31SR4To40"/>
   <migration source="arcm_4.0.0_rba_standard" target="arcm_4.1.0_rba_standard"
approach="rba" implementation="mig40To41" />
   <migration source="arcm_4.0.0_cba_standard" target="arcm_4.1.0_cba_standard"
approach="cba" implementation="mig40To41" />
</migrationPlan>
```

4.2 The XML schema of the migration plan

The documented schema for the file **migrationPlan.xml** is integrated as a resource in the library **arcm_datalayer_migration_migsteps.jar** and is located in the package **com.idsscheer.webapps.arcm.dl.datamigration.xsd.** This schema allows you to use the context-sensitive auto-complete, syntax check and help if you edit the **migrationPlan.xml** file in a development environment such as **IDEA** or **ECLIPSE**. This file cannot be changed. A modified schema cannot be processed by the migration framework.

4.3 The location of the migration plan

The migration plan is integrated in the folder **migrationPlan.xml** as a resource in the library **arcm_datalayer_migration_migsteps.jar** and is located in the package **com.idsscheer.webapps.arcm.dl.datamigration**. Changes and extensions are possible in the file **migrationPlan.xml**, as long as you generate a new library **arcm_datalayer_migration_migsteps.jar** with the modified class.

In CTK you need the sources of the standard migration in order to carry out custom adjustments. Subsequent to this, a library is created from these modified sources, which replaces the standard library.

The user guide **http://iwiki.eur.ad.sag:8080/display/PUB/ARCM+9.5+Eclipse+CTK (http://iwiki.eur.ad.sag:8080/display/PUB/ARCM+9.5+Eclipse+CTK)** contains information in the section **Migration** about how a migration in CTK can be adjusted to customer-specific databases.

5 The architecture

Java classes are saved in the source folder as migration steps (Page 4) that implement the **IMigrationStep** interface and extend the abstract **BaseMigrationStep** class. The method **::execute(...)** from **IMigrationStep** receives an instance from IMapping from the migration framework. To carry out the migration, various HighLevel functions from IMapping and IMigrationStep can now be used in the Execute method. The methods of the **IMapping** and **IMigrationStep** interfaces are stable and have Java documentation helps.

Figure 1: Mapping interface

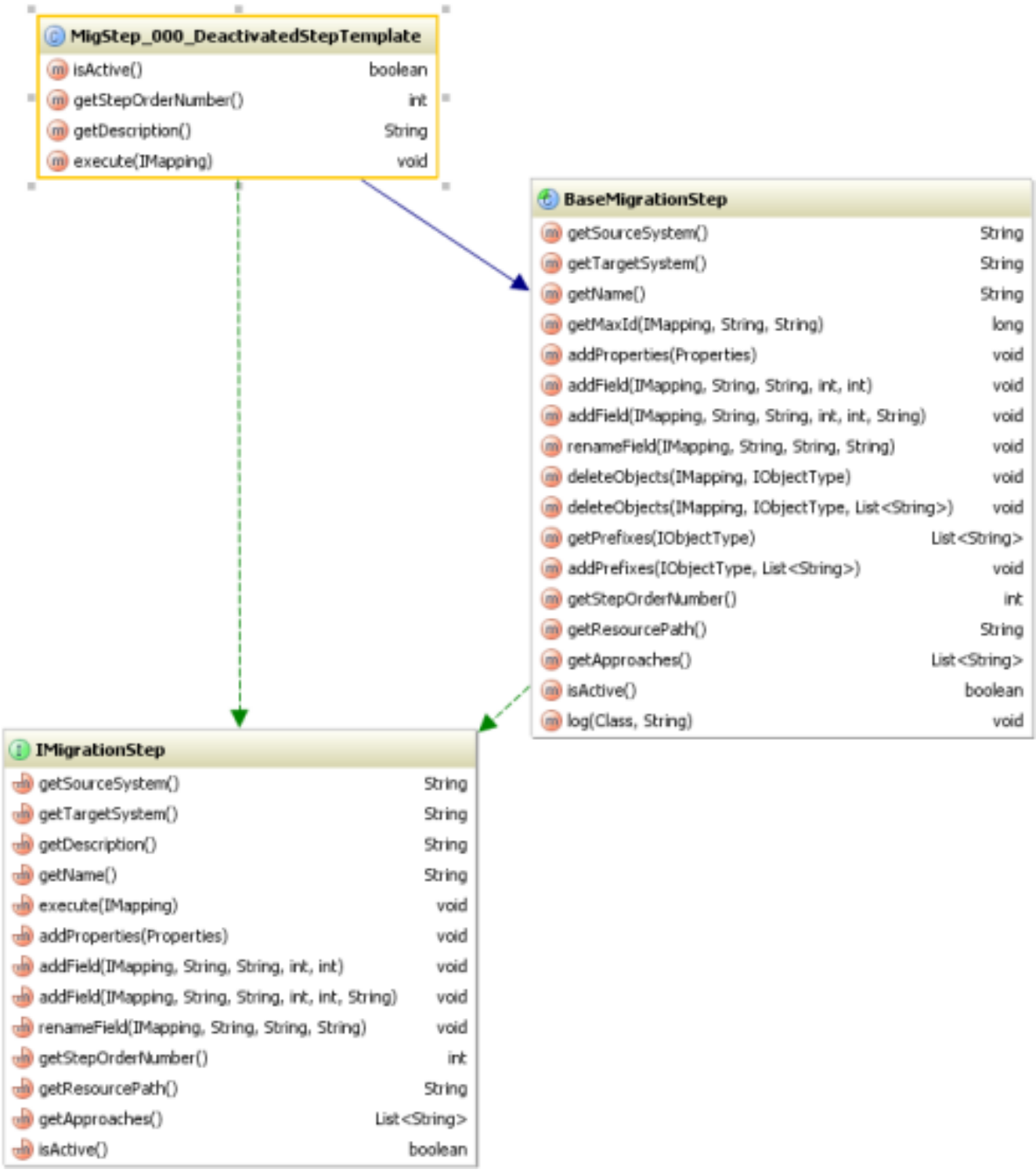

Figure 2: Migration interface

5.1 The construction set

All public methods from the **IMigrationStep** and **IMapping** interfaces can be combined just like in a construction set to achieve the desired results.

5.1.1 IMigrationStep

The **migSteps** subfolder is located in the **implementation** (Page 4) folder that contains all of the logic and data regarding a version transition. This folder contains all of the migration steps required for the version transition, which implement the **IMigrationStep** interface. The **IMigrationStep** interface provides help functions that package database-specific dependencies and that are not dependent on database-specific requirements. The following methods are not implemented by the abstract superior class **BaseMigrationStep** and must be implemented in the specific migration step.

- **IMigrationStep::getDescription() String**

  Provides the description of the step as a string.

- **IMigrationStep::execute(IMapping)**

  Carries out the step. All public methods of **IMapping** can be used.

The following methods are not implemented by the abstract superior class **BaseMigrationStep** and should be overwritten in the specific migration step.

- **IMigrationStep::getStepOrderNumber()**

  Determines the order (priority) of the steps to be performed. The default implementation always returns the order number **0**. Overwrite this function and enter a number larger than **0**, according to the position at which this step should be carried out. The lower the number the higher the priority.

- **IMigrationStep::isActive() Boolean**

  Specifies whether this step should be carried out (**return true**) or not (**return false**). The default implementation returns true. The step is thus active and is carried out. Overwrite this function if you wish to temporarily turn off a step during development. Some steps, such as the generation of database indices, should be deactivated as soon as a follow-up version is available. Indices should then be generated in the last version transition only.

The interface provides other useful functions for processing data and data structures. These are documented in the Java documentation.

5.1.2 Step template

The deactivated step template is located in the standard migration folder and provides a template with basic examples for processing the schema and the data. If you wish to create a new step, copy this template and adjust the class name, the name of the constructor and the **package** entry accordingly.

Also, make sure that you adjust all literals to the current requirements. Do not edit the template because it will be used as a template for further steps.

Warning

Never set the template to **true** in the **isActive()** function. The step would then be carried out at the start of a migration and your data could be damaged.

5.1.3 IMapping

The **IMapping** interface provides help functions that package data-specific properties. With these functions you can for example, add tables or fields and process data. A few of these functions perform the desired operations immediately and then close the required database resources automatically. A few functions also provide database resources directly. In this case, you must be sure to close these resources yourself in a Finally-block.

Close database resources such as **Connection** or **ResultSet**, which you receive from the **IMapping** interface, when they are no longer required. All methods in the **IMapping** interface affect the database connection specified in the **runtimeconfig.xml** configuration file in the **Datalayer** section. The interface provides other useful functions for processing data and data structures. These are documented in the Java documentation.

6 MigrationObject

The MigrationObject is a help structure with which to write data consistently in schema tables for ARIS Risk & Compliance Manager. Generate and write the objects sequentially in order to prevent conflicts with the internal ID management. This help structure does not generate the tables, but rather fills in the data semantically correct.

It is possible to generate a MigrationObject with the operator **new**.

```
MigrationObject migObject = new MigrationObject("POLICYREVIEWTASK", mapping, this,
UUID.randomUUID().toString(),
OVIDFactory.getOVID(SystemGUID.INTERNAL_SYSTEM_USER.getObjID()));
```

The newly generated object provides an API that can be used to maintain attributes with their values.

```
migObject.setAttribute("reviewRelevant", IMapping.TYPE_NUMBER, "0");
```

Relations can also be maintained like this:

```
migObject.setRelationAttribute("POLICYREVIEWTASK", "owner_group",
IMapping.TYPE_RELATION_1_1, ownerGroupID, 5520, 0, null);
```

With the **::write()** function the object can be written in the database.

```
migObject.write();
```

In order to receive a complete overview of the API from this class, check the **javadoc** of the file **MigrationObject.java**.

7 Automatic update of the schema version

In older versions of the migration framework it was necessary to write your own MigrationStep to maintain the **currentSchemaId** field in the **A_SCHEMAPROPERTY_TBL** table. This is no longer necessary with the current version of the framework. The corresponding field is now maintained automatically by the framework during the data migration.

8 Adjustments to the data migration in CTK

You can find the description as to how a migration can be adjusted to customer specific databases in CTL in the user guide **http://iwiki.eur.ad.sag:8080/display/PUB/ARCM+9.5+Eclipse+CTK (http://iwiki.eur.ad.sag:8080/display/PUB/ARCM+9.5+Eclipse+CTK)** in the section **Migration**.

9 Logging

Logging during migration takes place according to the settings in **log4j.properties**. Set the **arcm** and **dl.framework** packages to **debug** before migration.

```
log4j.logger.com.idsscheer.webapps.arcm=DEBUG
log4j.logger.com.idsscheer.webapps.arcm.dl.framework=DEBUG
```

The migration output is displayed on the console and in the output file that are set in the **log4j.properties** configuration file.

To prevent poor performance do not forget to undo these changes in the productive system.

Check the resulting log file carefully for error messages before you export the file and then import it into the productive system.