Datenmigrationshandbuch

ARIS Risk & Compliance Manager
Version 9.7- Service Release 3

April 2015

Product Documentation

Dieses Dokument gilt für ARIS Risk & Compliance Manager ab Version 9.7. Hierin enthaltene Beschreibungen unterliegen Änderungen und Ergänzungen, die in nachfolgenden Release Notes oder Neuausgaben bekanntgegeben werden.

Inhalt

1 Einführung

Seit der Version 4.0 von ARIS Risk & Compliance Manager beinhaltet der Server ein Migration-Framework zum inkrementellen Migrieren von Daten aus früheren Versionen. Dieses Dokument bietet Ihnen eine Einführung in Handhabung, Arbeitsweisen und Erweiterungsmöglichkeiten dieses Frameworks. Das Dokument richtet sich an alle Entwickler die ARIS Risk & Compliance Manager an die Gegebenheiten beim Kunden anpassen und die Datenmigration betreuen.

Falls es eine aktualisierte Version dieses Dokuments gibt, finden Sie diese hier: http://aris.softwareag.com/ARISDownloadCenter/ADCDocumentationServer (http://aris.softwareag.com/ARISDownloadCenter/ADCDocumentationServer)

2 Das Migration-Framework von ARCM

Der Server beinhaltet einen Mechanismus zum inkrementellen Migrieren von Daten aus früheren Versionen. Dabei können Daten in allen vom ARIS Risk & Compliance Manager unterstützten Datenbanksystemen (Oracle, MSSQL und Derby) migriert werden. Dazu steht ein Portfolio aus High-Level API-Funktionen zur Verfügung. Innerhalb selbst zu schreibenden oder zu ändernden Migrationsschritten können diese Funktionen dazu verwendet werden, Tabellen und Felder hinzuzufügen und Daten an neue Gegebenheiten anzupassen.

Das Framework kann durch Java-Klassen erweitert werden. Diese müssen textuell vorliegen. Es ist kein Kompilat erforderlich. Die interne Migrationslogik ermittelt anhand eines individuell anpassbaren XML-Migrationsplans die auszuführenden Schritte, um Datenstrukturen und Daten an den aktuell gestarteten Server anzupassen.

Warnung

Um Datenverlust und irreversible Veränderung Ihrer Daten vorzubeugen, empfehlen wir, eine vollständige Sicherung Ihrer Daten zu generieren. Verwenden Sie dazu die administrativen Werkzeuge ihres Datenbanksystems.

2.1 Was das Framework nicht leistet

Das interne Migrationsframework bearbeitet nur Daten und ihre Strukturen. Dies ist ausreichend, um den Standard von ARIS Risk & Compliance Manager zu migrieren. Es kann jedoch keine interne Logik wie Rules oder Workflow angepasst werden. Dies bedeutet für kundenspezifisch angepasste Versionen, dass neben der hier angesprochenen Datenmigration eventuell auch die Logik angepasst werden muss.

3 Die Migration starten

Um die Migration zu starten, setzen Sie im Konfigurationsfile **runtimeconfig.xml** den Parameter **dbms.autoStartMigration** auf **true**.

Beispiel

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

Ist dieser Parameter auf **true** gesetzt, wird beim Start des Servers geprüft, ob in der verbundenen Datenbank die Schemaversion des ARCM-Benutzers mit der aktuellen Schemaversion des ARCM-Servers übereinstimmt. Stellt der Server bei dieser Prüfung fest, dass die Version des Servers nicht mit der Version der Datenbank übereinstimmt, wird versucht eine passende Migrationsstrategie anhand des Migrationsplans (Seite 4) zu generieren.

Solange der Parameter auf **true** gesetzt ist, wird auf dem Startbildschirm des Systems und im Header eine Testsystem-Einblendung angezeigt. Indem Sie den Parameter ausschalten, wird diese Einblendung entfernt.

Nun kann eine produktive Umgebung durch einen Datenbankexport und -import generiert werden. Generieren Sie dazu ein neues Schema auf der Zieldatenbank und lassen Sie durch den Start des Servers von ARIS Risk & Compliance Manager die notwendigen Tabellen generieren. Ein Datenbankimport kann auch in ein anderes DBMS (Datenbankmanagementsystem) importiert werden. So ist es zum Beispiel möglich, eine Datenbank die unter Oracle migriert wurde, auf einem MSSQLServer produktiv zu schalten.

4 Der Migrationsplan

Der Migrationsplan ist eine XML-Datei mit dem Namen **migrationPlan.xml**. Hier wird ein Versionsübergang einem Verzeichnis zugewiesen, in dem die entsprechende Migrationslogik zu finden ist.

4.1 Format

Ein Versionsübergang ist in dem Tag **migration** definiert, das folgende Attribute besitzt:

- **Name**: Beinhaltet den Namen eines Versionsübergangs. Dieser Name wird bei der Ausführung des Versionsübergangs in die Tabelle **A_SCHEMAPROPERTY_TBL** eingetragen. Anhand dieser Information kann zu einem späteren Zeitpunkt die Migrationshistorie der Datenbank nachverfolgt werden.
- **Source**: Beinhaltet die Startversion die in der Tabelle **A_SCHEMAPROPERTY_TBL** des verbundenen Datenbanksystems abgelegt ist. Ist die Startversion unbestimmt, so kann hier der Wert **start** eingetragen werden. In diesem Fall wird bei der Migration, wenn keine passende Quelle zu einem Schema gefunden wurde, mit dem als **start** markierten Versionsübergang im Plan begonnen.
- **Target**: Beinhaltet die Zielversion die nach dem aktuellen Versionsübergang erreicht werden soll.
- **Approach**: Gibt aus, ob sich der aktuelle Versionsübergang auf den risikobasierten oder kontrollbasierten Ansatz bezieht. Mögliche Werte sind **rba** und **cba**.
- **Implementierung**: Der Ordner der die Migrationslogik zu diesem Versionsübergang enthält. Der Pfad zu diesem Ordner setzt sich aus zwei Komponenten zusammen. Der erste Teil ist der allgemeine **SourceFolder [installation Directory] / jsp / WEB-INF / config / migration**. Der zweite Teil wird aus den Java-Packages gebildet, die einen Pfad zu dem Basispaket **com.idsscheer.webapps.arcm.dl.datamigration** bilden. Unter diesem Basispaket wird nun der Ordner **implementation** abgelegt. Dieser Pfad liegt außerhalb der ARCM-Bibliothek und kann daher mit eigenen Klassen und Ressourcen erweitert werden, die nicht kompiliert werden müssen.
- **Fix**: Ist dieses optionale Attribut auf den Wert **true** gesetzt wird der angegebene Versionsübergang als Hotfix auf einer bestehenden Version ausgeführt. Die Version der Datenbank bleibt unverändert. Hier muss für die Attribute **source** und **target** die gleichen Version beinhalten.

Beispiel

```
<?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 Das XML-Schema des Migrationsplans

Das dokumentierte Schema der Datei **migrationPlan.xml** ist als Ressource in die Bibliothek **arcm_datalayer_migration_migsteps.jar** eingebunden und liegt im Paket **com.idsscheer.webapps.arcm.dl.datamigration.xsd.** Wenn Sie die Datei **migrationPlan.xml** in einer Entwicklungsumgebung wie **IDEA** oder **ECLIPSE** bearbeiten, können Sie aufgrund dieses Schemas die kontextsensitive Autovervollständigung, Syntaxprüfung und Hilfe nutzen. Diese Datei darf nicht verändert werden. Ein geändertes Schema kann von dem Migrations-Framework nicht verarbeitet werden.

4.3 Die Lage des Migrationsplans

Der Migrationsplan ist im Ordner **migrationPlan.xml** als Ressource in die Bibliothek **arcm_datalayer_migration_migsteps.jar** eingebunden und liegt im Paket **com.idsscheer.webapps.arcm.dl.datamigration**. In der Datei **migrationPlan.xml** sind Änderungen und Erweiterungen möglich, sofern sie sich eine neue Bibliothek **arcm_datalayer_migration_migsteps.jar** mit den geänderten Klasse erzeugen lassen.

Im CTK benötigen Sie die Sourcen der Standardmigration um kundenspezifische Anpassungen vornehmen zu können. Im Anschluss wird eine Bibliothek aus diesen geänderten Sourcen erzeugt, die die Standardbibliothek ersetzt.

Im Benutzerhandbuch **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)** finden Sie im Abschnitt **Migration** Informationen darüber, wie eine Migration im CTK an kundenspezifische Datenbanken angepasst werden kann.

5 Die Architektur

Im Source-Folder werden Java-Klassen als Migrationsschritte **(Seite 4)** abgelegt, die das Interface **IMigrationStep** implementieren und die abstrakte Klasse **BaseMigrationStep** erweitern. Die Methode **::execute(…)** von **IMigrationStep** bekommt eine Instanz von IMapping aus dem Migration-Framework. Um die Migration durchzuführen, können nun in der Execute-Methode verschiedene HighLevel Funktionen aus IMapping und IMigrationStep genutzt werden. Die Methoden der Interfaces **IMapping** und **IMigrationStep** sind stabil und mit Javadoc-Hilfen versehen.

Abbildung 1: Mapping-Interface

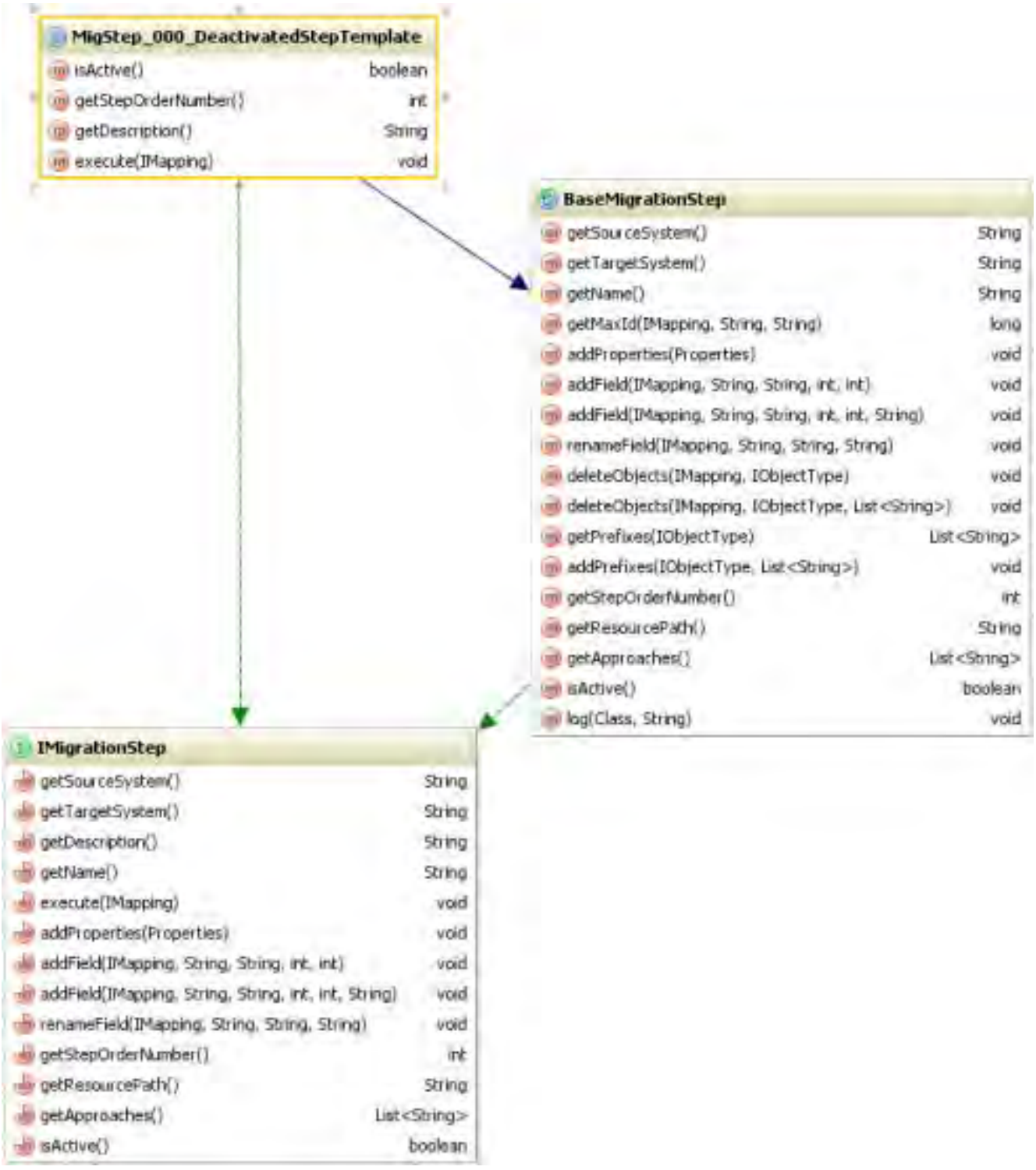

Abbildung 2: Migration-Interface

5.1 Der Baukasten

Alle öffentlichen Methoden aus den Interfaces **IMigrationStep** und **IMapping** können wie in einem Baukasten kombiniert werden, um das gewünschte Ergebnis zu erzielen.

5.1.1 IMigrationStep

In dem Ordner **implementation** (Seite 4), der alle Logik und Daten zu einem Versionsübergangs enthält, befindet sich der Unterordner **migSteps**. Dieser Ordner enthält alle für den Versionsübergang notwendigen Migrationsschritte, die das Interface **IMigrationStep** implementieren. Das Interface **IMigrationStep** liefert Hilfsfunktionen die datenbankspezifische Abhängigkeiten kapseln und unabhängig sind von datenbankspezifischen Gegebenheiten. Folgende Methoden sind nicht durch die abstrakte Oberklasse **BaseMigrationStep** implementiert und müssen im konkreten Migrationsschritt implementiert werden.

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

  Liefert die Beschreibung des Schritts als String.
- **IMigrationStep::execute(IMapping)**

  Führt den Schritt aus. Alle öffentlichen Methoden von **IMapping** können verwendet werden.

Folgende Methoden sind durch die abstrakte Oberklasse **BaseMigrationStep** implementiert und sollten im konkreten Migrationsschritt überschrieben werden.

- **IMigrationStep::getStepOrderNumber()**

  Bestimmt die Reihenfolge (Priorität) der auszuführenden Schritte. Die Standardimplementierung liefert immer die Order-Number **0**. Überschreiben Sie diese Funktion und tragen Sie eine Nummer größer **0** ein, entsprechend der Position an welcher dieser Schritt ausgeführt werden soll. Je kleiner die Nummer desto höher die Priorität.
- **IMigrationStep::isActive() Boolean**

  Bestimmt, ob dieser Schritt ausgeführt werden soll (**return true**) oder nicht (**return false**). Die Standardimplementierung liefert true. Der Schritt ist somit aktiv und wird ausgeführt. Überschreiben Sie diese Funktion, wenn Sie während der Entwicklung einen Schritt temporär ausschalten wollen. Manche Schritte, wie das Generieren von Datenbankindices, sollten deaktiviert werden, sobald es eine Folgeversion gibt. Indices sollten dann nur in dem letzten Versionsübergang generiert werden.

Das Interface stellt noch weitere nützliche Funktionen zur Bearbeitung von Daten und Datenstrukturen zur Verfügung. Diese sind mit Javadoc dokumentiert.

5.1.2 Step-Template

Das deaktivierte Step-Template befindet sich in den Standardmigrationsordnern und bietet eine Vorlage mit grundlegenden Beispielen für das Bearbeiten des Schemas und der Daten. Wenn Sie einen neuen Schritt anlegen wollen, kopieren Sie dieses Template und passen Sie den Klassennamen, den Namen des Constructors und den Eintrag **package** entsprechend an.

Achten Sie auch darauf, alle Literale an die aktuellen Gegebenheiten anzupassen. Editieren Sie nicht das Template, da es noch als Vorlage für weitere Schritte dient.

Warnung

Setzen Sie das Template in der Funktion **isActive()** niemals auf **true**. Der Schritt würde bei einem Start der Migration ausgeführt werden und Ihre Daten können beschädigt werden.

5.1.3 IMapping

Das Interface **IMapping** liefert Hilfsfunktionen die datenbankspezifische Eigenschaften kapseln. Mit Hilfe dieser Funktionen können Sie zum Beispiel Tabellen oder Felder hinzufügen und Daten bearbeiten. Einige dieser Funktionen führen die gewünschten Operationen sofort aus und schließen danach die benötigten Datenbankressourcen automatisch. Einige Funktionen liefern auch direkt Datenbankressourcen zurück. In diesem Fall müssen Sie darauf achten, diese Ressourcen selbst in einem Finally-Block zu schließen.

Schließen Sie Datenbankressourcen wie **Connection** oder **ResultSet**, die Sie aus dem Interface **IMapping** erhalten, wenn sie nicht mehr benötigt werden. Alle Methoden im Interface **IMapping** wirken auf die Datenbankverbindung, die in dem Konfigurationsfile **runtimeconfig.xml** in dem Abschnitt **Datalayer** angegeben ist. Das Interface stellt noch weitere nützliche Funktionen zur Bearbeitung von Daten und Datenstrukturen zur Verfügung. Diese sind mit Javadoc dokumentiert.

6 MigrationObject

Das MigrationObject ist eine Hilfsstruktur um Daten konsistent in Schema Tabellen für ARIS Risk & Compliance Manager zu schreiben. Erzeugen und schreiben Sie die Objekte sequentiell, um Konflikte mit dem internen ID-Management zu vermeiden. Diese Hilfsstruktur erzeugt nicht die Tabellen, sondern füllt Daten semantisch korrekt ein.

Es ist möglich ein MigrationObject mit dem Operator **new** zu erzeugen.

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

Das neu erzeugte Objekt bietet eine API, um Attribute mit ihren Werten zu pflegen.

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

Auch Relationen können so gepflegt werden:

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

Mit der Funktion **::write()** kann das Objekt in die Datenbank geschrieben werden.

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

Um einen vollständigen Überblick über die API dieser Klasse zu erhalten, prüfen Sie die **javadoc** der Datei **MigrationObject.java**.

7 Automatisches Aktualisieren der Schemaversion

In älteren Versionen des Migrationsframeworks war es erforderlich einen eigenen MigrationStep zu schreiben, um das Feld **currentSchemaId** in der Tabelle **A_SCHEMAPROPERTY_TBL** zu pflegen. Mit der aktuellen Version des Frameworks entfällt diese Notwendigkeit. Das entsprechende Feld wird nun vom Framework automatisch während der Datenmigration gepflegt.

8 Anpassungen der Datenmigration im CTK

Die Beschreibung wie eine Migration im CTK an kundenspezifische Datenbanken angepasst werden kann, finden Sie im Benutzerhandbuch **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)** im Abschnitt **Migration**.

9 Das Logging

Das Logging während der Migration folgt den Einstellungen in den **log4j.properties**. Hier stellen Sie vor der Migration die Pakete **arcm** und **dl.framework** auf **debug**.

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

Die Ausgabe der Migration wird dann auf der Konsole und in der Ausgabedatei vorgenommen, die in der Konfigurationsdatei **log4j.properties** eingestellt sind.

Um Performancenachteile auszuschließen, vergessen Sie nicht diese Änderungen im produktiven System rückgängig zu machen.

Prüfen Sie die resultierende Logdatei sorgfältig auf Fehlermeldungen bevor Sie die Datei exportieren und in das Produktivsystem importieren.