eHealth Framework

How to enable Multi-Schema Support

Imprint

InterComponentWare
Altrottstraße 31
69190 Walldorf
Tel.: +49 (0) 6227 385 0
Fax.: +49 (0) 6227 385 199

Document version: preliminary
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.10.3
Last Change: 23.03.2010
Editorial Staff: BAS Technology

Notice

Note:

The current version of this document has a draft status and various chapters are still in review.

The document is collaboratively built with the use of the Darwin-Information-Typing-Architecture (DITA) and has therefore a draft status concerning styles and layout. The necessary adaptations are currently also in a developmental stage.

Contents

1 Overview

Modules can be used in different applications while an application may be installed several times as separate disparate instances. In order to separate the persistent data of two or more application instances and to prevent conflicts, it is often required to provide a separate database instance for every single instance of an application.

This imposes huge costs, in particular, when a license has to be purchased per database. To control and to reduce the total cost of ownership for hosting applications, the eHF supports separated module and application persistence spaces by modifying the schema names. This feature, called multi-schema support, allows for the use of a single database instance for many applications simultaneously. Licensing costs are therefore reduced proportional to the number of applications installed on a single database instance.

Multi-schema support needs to be enabled on both, the module as well as the assembly level. This how-to describes the necessary steps to enable multi-schema support for modules and assemblies.

2 Enabling Multi-Schema Support in a Module

This chapter describes the required steps to enable your module for multi-schema support. For most eHF Generator based modules in fact **no adaption is required** apart of a simple rebuild. The generator will add the necessary configuration to your module automatically.

However, there are cases where even for generated modules modifications are required. Please have a look at Table 1, which steps are necessary for enabling multi-schema support.

Section	Description	Condition
2.1	Change the schema name	The schema name exceeds 22 characters.
2.2	Change the database configuration	This task is only required in case you have a custom version of the file `connection.properties` in the directory `src/main/config/db`.
2.3	Change the Spring configuration	This only applies to module, which provide a local Hibernate session factory.
2.4	Make DAOs multi-schema-aware	The module contains DAOs which use plain SQL to access the database.

Table1. Task Overview

2.1 Change the Module Schema Name

This section only applies in case your module has a schema name that exceeds 22 characters. Some databases impose a restrictions on the length of schema names (for example, in an Oracle database the schema name is restricted to 30 characters). We anticipate prefixes and suffixes of three characters and a separator character respectively. This means that we recommend as a maximum length for schema names:

```
30 - 2 x (3 + 1) characters = 22 characters
```

In case your module schema name exceeds 22 characters and needs to be shortened, please go through the following steps:

1. Change the schema name to comply to the 22 characters constraint

   Locate the property `module.schema.name` and replace the value with a shorter choice. The property is located in the `module.configuration.properties` file, which is usually to be found on the root level of your module project.

   In case you have customized domain object these may carry the old schema name within the JPA and Hibernate annotation. Make sure you replace them accordingly.

   In case your module is not generated you may not find this property in a central place and you probably need to do a search and replace on your modules codebase.

   Your module schema name configuration carries the new schema name. you can ensure that all occurrences are found by running your module tests.

Note: Sometimes it is not possible to rename the schema. For example, when your module is already deployed in production and would require intense upgrade measures. Then, you should preserve the current schema name as is, but document the implied restrictions on prefixing and suffixing for your module. We also recommend to contact the eHF development team to discuss other options for your particular case. The restriction of your module will unfortunately limit all applications using your module.

2.2 Change the Module Database Configuration

This task is only required if there exists a custom version of the file `src/main/config/db/connection.properties`.

The `connection.properties` in `src/main/config/db` provides database setup configuration specialized for your module. For generated modules this file is created by the eHF Generator. For manually written persistent modules this file has to be added in case it does not exist. For generated modules the file can also be supplied in order to provide custom settings, but this is rather rare and is only used for modules which are enabled to use a separate data source (known cases in the scope of eHF are the modules eHF Encryption and eHF Audit).

1. Add properties to your local `connection.property file`
   A custom `connection.properties` usually looks as follows:

   ```
   connection.schema.name=@@module.schema.name@@
   connection.dialect=@@encryption.connection.dialect@@
   connection.schema.prefix=@@connection.schema.prefix@@
   connection.schema.suffix=@@connection.schema.suffix@@
   connection.schema.password.key=connection.<module-name>.pass
   ```

   Please supply missing properties (listed above) in this file accordingly.

   With the supplied information in `connection.properties` the installation procedure on assembly level has all required meta-information to install the schema for your module with appropriate prefix and/or suffix.

2.3 Change the Module Spring Configuration

This step only applies to modules that provide a local Hibernate session factory. This case is rather seldom and normally only valid for modules, which operate on a separate data source.

If your module has a local session factory, additional properties are required to enable the session factory for multi-schema support. Please note that this how-to assumes naming conventions recommended by eHF. You can use any other names for the properties. However, the following ones are the recommended settings and are intended to create flexibility while preserving a homogeneous naming scheme across modules.

1. Add properties to your local module property file.
   A session factory is configured in a property file that is located in the directory `src/main/config`. Usually it named according to the eHF naming conventions `<prefix>-<module-name>.properties` (for example, the module eHF Audit provides the property file `ehf-audit.properties`) and should supply the following content:

   ```
   <module-name>.connection.driver=@@encryption.connection.driver@@
   ```

```
<module-name>.connection.url=@@<module-name>.connection.url@@
<module-name>.connection.username=@@<module-name>.connection.username@@
<module-name>.connection.dialect=@@<module-name>.connection.dialect@@
[...]
```

Add the following two properties to the above configuration file:

```
<module-name>.connection.schema.prefix=connection.schema.prefix@@
<module-name>.connection.schema.suffix=@@<module-name>.connection.schema.
suffix@@
```

Your module is now able to get a schema suffix and/or prefix configured.

2. Configure your session factory to use the pre- and/or suffix properties

   You should be able to locate the session factory bean in your spring context (do a full-text search for `sessionFactory` or `SessionFactoryBean`).

   Usually this bean is configured using the `hibernateProperties` property. In case this configuration does not exist add it or extend it as follows:

```
<property name="hibernateProperties">
  <props>
    [...]
    <prop key="hibernate.ehf.schema.prefix">
      ${<module-name>.connection.schema.prefix}
    </prop>
    <prop key="hibernate.ehf.schema.suffix">
      ${<module-name>.connection.schema.suffix}
    </prop>
  </props>
</property>
```

   The session factory bean is now aware of the prefix and/or suffix to the schema.

3. Replace the Hibernate configuration class
   In order to apply the schema prefix and suffix to the session factory the `configurationClass` property of the the session factory bean `configurationClass` has to be replaced as follows:

```
<property name="configurationClass"
    value="com.icw.ehf.commons.hibernate.AnnotationConfiguration"/>
```

   Please note that multi-schema support is currently only provided for annotation-based mappings. In case your module uses other means to establish the session factory mapping meta data a custom implementation of the configuration class must be provided. Please approach the eHF development team to address your particular case.

   The session factory is now using schema prefix and/or suffix and is able to use this configuration at runtime to extend the original module schema name.

2.4 Make DAOs Multi-Schema-Aware

The steps described in this section are only required in case your module contains DAOs which use plain SQL to access the database. This SQL statements include the schema name and therefore require additional treatment. Other Hibernate query mechanisms such as criteria or Hibernate Query Language (HQL) do not require any modifications as they access the meta information from the session factory.

In case your module uses plain SQL to access the database execute the following steps to enable multi-schema support.

1. Isolate the full table and infer it from the environment

Depending on the way your code is organized you should be able to isolate the full table name (including the schema and the table name seperated by a '.') in your SQL statements. For inferring the the pre- and/or suffixed schema name from the environment eHF provides two approaches:

In case your module uses Hibernate to access the database and the SQL approach is only used for special cases - like e.g. streaming support - you can infer the full table name from the session factory. In order to do this your need to inject the session factory into your DAO (please use standard eHF and Spring means to accomplish this). Once you have the session factory injected you can use the class

```
com.icw.ehf.commons.hibernate.SessionFactoryUtils
```

to infer the full table name (including the configured schema prefix and suffix) as follows:

```
String fullTableName = SessionFactoryUtils.
    extractTableNameForClass(yourPersistentType, sessionFactory);
```

If your module does not use a session factory you may use the class

```
com.icw.ehf.commons.dao.SchemaConfiguration
```

and configure a bean in the internal spring context of your module. The bean definition for this looks as follows:

```
<bean id="schemaConfiguration"
    class="com.icw.ehf.commons.dao.SchemaConfiguration">
        <constructor-arg type="java.lang.String"
            value="${<module-name>.connection.schema.prefix}"/>
        <constructor-arg type="java.lang.String"
            value="${<module-name>.connection.schema.suffix}"/>
</bean>
```

Having injected this bean into your DAO you can now simply invoke

```
String extendedSchemaName = modifiySchemaName(originalSchema);
```

and finally compose your SQL statement based on this information. In order to supply the values for the prefix and suffix you have to use a `PropertyPlaceholderConfigurer` bean and supply the specified configuration file on module and on assembly level.

Your DAOs are now multi-schema enabled. In order to test this you can isolate the SQL statement building code and use simple unit testing. Further testing can unfortunately only be done at installation time.

3 Enabling Multi-Schema Support in an Assembly

This chapter illustrates how multi-schema support can be enabled on assembly level for a product. Please note that you first have to enable all modules of the assembly for multi-schema deployments.

3.1 Change the Assembly Property Configuration

In your assembly you provide a production close default configuration for all properties in your application. These properties can be adjusted to the needs of the customer at installation time. The eHF provides capabilities to separate different concerns into separate property files and use an inclusion mechanism from the central `configuration.properties` file to make them available to the installation procedure. The organization of these properties are subject to the assembly an can be organized as desired by the assembly developers. In order to allow the customer to define a prefix and/or suffix to the schemas, you have to add a prefix and suffix property to your assembly.

1. Add properties to your assembly configuration.

   Locate the best place for specifying the prefix and the suffix property. Probably it is best to put this configuration next to the other connection or database properties.

   When you have found the place to add products defaults for the new properties. Here we specify that no prefix or suffix are used as a default:

   ```
   connection.schema.prefix=
   connection.schema.suffix=
   ```

   Your assembly now indicates that it supports multi-schema deployments. You supplied the defaults, which you find most suitable for your product. Please ensure that your installation manual is updated accordingly.

3.2 Change the Assembly Spring Configuration

Currently, with eHF 2.9, an assembly containing persistent modules supplies an assembly-level session factory bean. This session factory bean requires additional configuration to enable multi-schema support.

1. Configuring your session factory to use the pre- and/or suffix properties

   You should be able to locate the session factory bean in your spring context (do a full-text search for `sessionFactory` or `SessionFactoryBean`).

   Usually this bean is configured using the `hibernateProperties` property. In case this configuration does not exist add it or extend it as follows:

   ```
   <property name="hibernateProperties">
    <props>
     [...]
     <prop key="hibernate.ehf.schema.prefix">@@connection.schema.prefix@@</prop>
     <prop key="hibernate.ehf.schema.suffix">@@connection.schema.suffix@@</prop>

    </props>
   </property>
   ```

   Depending on how you apply the properties you may also use a different token to identify property place holders (e.g ${a} instead of @@a@@).

   The session factory bean is now aware of pre- and suffixes to the schema.

2. Replace the Hibernate configuration class
   In order to apply the schema pre-and suffixes the `configurationClass` property of the session factory bean `configurationClass` has to be replaced as follows:

   ```
   <property name="configurationClass"
       value="com.icw.ehf.commons.hibernate.AnnotationConfiguration"/>
   ```

   Please note that multi-schema support is currently only provided for annotation-based mappings. In case your assembly uses other means to establish the session factory mapping meta data a custom implementation of the configuration class must be provided. Please approach the eHF development team to address your particular case.

   The session factory is now using schema pre- and suffixes and is able to use this configuration at runtime.

3.3 Change the Module Property Files

Your assembly may use modules, which provide their own session factory. In order to keep these module internal session factories configurable on assembly-level, these modules normally require module-specific property files being supplied on assembly level. Please note that from the existing eHF modules only eHF Encryption and eHF Audit require such an assembly-located property file.

The assembly-located property files of persistent modules have to be extended to supply schema pre- and suffixing configuration.

1. Add properties to the assembly-located module property files
   Locate the aforementioned property files in your `src/main/config` folder. For eHF Audit and eHF Encryption these property files are named `ehf-audit.properties` and `ehf-document`. These property files should already contain the following content:

   ```
   <module-name>.connection.driver=@@audit.connection.driver@@
   <module-name>.connection.url=@@audit.connection.url@@
   <module-name>.connection.username=@@audit.connection.username@@
   [...]
   ```

   Supply the following properties to each of these files:

   ```
   <module-name>.connection.schema.prefix=@@connection.schema.prefix@@
   <module-name>.connection.schema.suffix=@@connection.schema.suffix@@
   ```

   As a result all consumed modules with a separate session factory are supplied with the prefix and suffix settings.

3.4 Testing Multi-Schema Support

Prerequisite for the following steps is that your assembly is enabled for multi-schema support.

The enabled assembly is supposed to be tested concering the multi-schema support. For this purpose you can use a simply hypersonic SQL database (HSQLDB) as supplied by the eHF build environment.

1. Ensure a clean database is running

   Locate the installation folder of the HSQLDB. In case you have the ICW-IDE installed you find the HSQLDB in the IDE installation folder `database/hsqldb`.

Clean the database by deleting the files starting the testdb in the `bin` folder. If the HSQLDB is already running make sure you stop it as a running process may have locks on the files.

As a result the HSQLDB is cleaned.

2. Start the database

   Now run the `server-start.bat` or `startup.sh` located in the `bin` folder.

   A fresh and empty HSQLDB database process is started. If you connect to the database (`username=sa`, `password='' (empty)`) you can see that the database contains no schemas yet.

3. Build a distributable artifact of your assembly

   Run then command `maven clean ehf:release` in your assembly project.

   A distributable artifact of your assembly is build. The `target` folder contains the archive as well as an unarchived folder with the content.

4. Install the assembly
   Change directories into the unarchived folder (e.g. `target/<assembly-name>-<version>-bin`) and include the following properties into the `configuration.properties` file.

```
connection.schema.prefix=ABC
connection.schema.suffix=001
```

   You may of course choose you favorite prefix and suffix for testing.

   Please furthermore ensure that all properties are supplied that enable to connect to the local HSQLDB. You may copy these settings from the `configuration.properties` file in your assembly.

   Execute the command `ant -f install/install.xml install:all` on unix based systems or `ant -f install\install.xml install:all` on windows based systems.

   The installation procedure is executed. The procedure implicitly creates all the schemas in the HSQDB database and executes the bootstrap and import steps as configured in your assembly.

5. Validate the right schemas have been created

   Have a closer look into the HSQLDB database again. After refreshing the list of schemas you should now be able to see all schemas of the modules included in your assembly being prefixed and suffixed according to your configuration. E.g. in case you used the above suggested extensions and your assembly contains the eHF Code System module. You should be able to see a schema named `ABC_EHF_CODESYSTEM_001`.

   In case all schemas are prefixed as expected your test on this level (also covering the bootstrap and import) is successful. The next level is a test deployment of the prefixed version with all your functional tests against a production type of database like oracle.

4 Troubleshooting

I tried to install my assembly against a clean database, but the schema creation step fails.

Ensure all the modules your are using in the assembly are multi-schema enabled. This error normally indicates that a module does not meet all criteria to support multi-schema deployments.

I tried to install my assembly against a clean database, but the bootstrap/import step fails.
There are various possibilities for causes of problems during the import. It reaches from not or incompletely enabled modules to missing configuration elements on assembly level. In the following we only list two general general cases. To resolve such issues it is recommended to have a deeper look into your assembly again and potentially run again through the tasks described in this howto. In the following we try to cover selected cases:

1. Ensure that all the schemas are prefixed as expected. In case one schema is not pre- or suffixed you may have included a module, which is not yet ready for multi-schema deployments. In particular the database configuration of this module may be incomplete.
2. Another cause for the issue could be that you missed a module configuration properties file located in your assembly. In this case the schema is correctly modified with pre- and suffix, but the configuration does not really reach the module. Ensure your assembly setup is complete and continue investigations in the direction of the module that causes the trouble (see stacktrace during bootstrap/import).