

Release Notes

eHealth Framework 2.9_preview_20091106

Imprint

InterComponentWare
Industriestraße 41
69190 Walldorf
Tel.: +49 (0) 6627 385 0
Fax.: +49 (0) 6627 385 199

© Copyright 2009 InterComponentWare AG. All rights reserved.

Document version: 1.0
Document Language: en (US)
Security Level: Public
Document Status: Approved
Product Name: eHealth Framework
Product Version: 2.9
Last Change: 26.10.2009
Editorial Staff: BAS Technology and eHF Contributors

Preface

The eHealth Framework (eHF) represents the technology platform of ICW and is continuously improved and enhanced.

The eHF Release Notes summarize all changes and bug fixes in between different releases in order to enable users, partners and customers to benefit from the enhancement and support the adaption to changes.

Please note that the eHF documentation is publicly accessible from the ICW Developer Network (IDN). You can navigate to the IDN by following this link to the [references section](#) on page 46 of this document.

All logos are registered trademarks of InterComponentWare AG. The product names mentioned in this document are either the trademarks or registered trademarks of the respective owners and are used for identification purposes only.

This document and all related software components are protected by copyright © 2009 InterComponentWare AG.

All rights reserved.

Contents

1 Overview	1
2 Developer Notes	2
2.1 Encryption Infrastructure Changes (Sprint 30)	2
2.2 Encryption Infrastructure Changes (Sprint 29)	3
2.3 Value Objects (Sprint 29)	5
2.4 Code Meta Data Validation (Sprint 29)	5
2.5 Migration to IPF 2.0-m1 and Apache Camel 2.0.0	6
2.6 Extension to SessionFactory Configuration	8
3 Development Environment	9
3.1 eHF Build Plug-in	9
3.2 eHF Install Scripts	9
3.2.1 Initialization for Encryption.....	9
3.2.2 Automation of Database Privileges.....	10
4 Application Platform	12
4.1 Commons Module	12
4.1.1 Domain Objects and Persistence.....	12
4.1.2 Support for Code Validation.....	13
4.1.3 Code System Meta Data Format Changes and Code Meta Data Validation.....	14
4.1.4 Change in length determination algorithm for encrypted columns.....	16
4.1.5 Refactoring the Date/Time API.....	16
4.1.6 Crud Services at Adapter Level and Service Level for eHF Service.....	17
4.1.7 Signature of persistChild method in ObjectManagerAwareCrudService changed.....	17
4.1.8 Changes in Error Processing Behavior	18
4.2 Commons Entity Module	19
4.2.1 Implementation of the Entity-Provider SPI.....	19
4.2.2 Implementation of the Entity-Converter SPI.....	20
4.2.3 Implementation of the Entity-Resolver SPI.....	20
4.2.4 Implementation of the Entity-Manager SPI.....	20
4.2.5 Implementation of the Entity-Relationship-Provider SPI.....	20
4.2.6 Using the API of the Commons Entity Module.....	21
4.2.7 Migrating to the API of the Commons Entity Module.....	22
4.3 Entity Relationship Module	23
4.4 User Management Module	23
4.4.1 Implementation of the Entity Participant Converter Interface.....	23
4.4.2 Implementation of the Entity Provider Interface.....	23
4.4.3 Implementation of the Entity Resolver Interface.....	23
4.4.4 Deprecation of the payment data.....	23
4.4.5 User secret history.....	24
4.5 Encryption Support	24
4.6 Encryption Support and Harmonization of Personal Data	26

4.7 Handling for DocumentIdentifier of DocumentContentContext	26
4.8 Pseudonymization and Harmonization	26
4.9 Commons-Code-System Module	27
4.9.1 Refactoring of the Codesystem-API.....	27
4.10 eHF Assembly	29
4.10.1 Change of Configuration Structure.....	29
4.10.2 Automation of Database Privileges.....	29
4.10.3 Migration of Upgrade and Encryption scripts to the modules.....	29
5 Fixed Change Requests	32
5.1 Sprint 29 and 30	32
5.1.1 High Priority Change Requests	32
5.1.2 Medium Priority Change Requests	32
5.1.3 Low Priority Change Requests	32
5.2 Sprint 24 to 28	32
5.2.1 High Priority Change Requests	32
5.2.2 Medium Priority Change Requests	33
5.2.3 Low Priority Change Requests	34
6 Fixed Issues	35
6.1 Sprint 29 and 30	35
6.1.1 Fixed High Priority Issues.....	35
6.1.2 Fixed Medium Priority Issues.....	35
6.1.3 Fixed Low Priority Issues.....	36
6.2 Sprint 24 to 28	36
6.2.1 Fixed High Priority Issues.....	36
6.2.2 Fixed Medium Priority Issues.....	37
6.2.3 Fixed Low Priority Issues.....	41
7 Known Issues	42
7.1 Known High Priority Issues	42
7.2 Known Medium Priority Issues	42
7.3 Known Low Priority Issues	43
8 Third-Party Components	45
9 References	46

1 Overview

Since the last weeks the dominating development focus clearly lied on the realization of an encryption infrastructure for application level encryption and pseudonymization. This major topic is covered in the corresponding module sub-sections of the application platform chapter. Further an additional paragraph summarizes aspects applying to all pseudonymization-related modules. Also a set of additional howto documentation on this matter is provided.

Please note that the eHF Key Tools and the Medicine Cabinet are covered within the encryption module section, since they are just playing a supporting role for initialization and reference module respectively.

Further the build plug-in, the automation of data base privileges and the installation scripts were also affected and are described in the development environment section.

The content of this document is organized in alignment to the components of the eHealth Framework (eHF). On the highest level the framework can be decomposed into the *Development Environment* and the *Application Platform* .

The *Development Environment* provides tools for designing, building, deploying and running eHF-based applications. The *Application Platform* provides reusable software modules for building applications in the eHealth domain.

The first chapters illustrate the enhancements of the eHF modules. Changes are also documented in order to communicate benefits and upgrade support for the current release of the eHF.

Please note that modules, which remain unchanged are not listed in this document.

Change Requests are summarized in chapter four.

The *Fixed Issues* chapter lists all issues that have been resolved and closed in eHF since the last release.

Analogically the *Known Issues* section depicts all items that are currently open and in progress.

Third Party Components shows all third-party dependencies of eHF and especially highlights version upgrades of third-party components.

The *References Chapter* lists other documentation that is relevant in this context.

The target audience of this document are engineers and architects developing with eHF.

2 Developer Notes

Many customers of eHF are required to work with the latest snapshots or technical tags of eHF, being forced to consume the latest feature set. Working with software in transition has the disadvantage that the software may go through intermediary development stages. In order to harness the impact on the early adopters we collect developer notes in this chapter that includes notices and change descriptions that help to solve issues and also explain the background of the changes.

2.1 Encryption Infrastructure Changes (Sprint 30)

Target Mapping

Application-level encryption is controlled by a configuration file on assembly level named `ehf-target-mapping.xml` (or `ehf-target-mapping-nullcipher.xml`). In this configuration file the strength of the encryption can be defined for the application.

The target mapping is using module aliases to provide module-specific configuration (e.g. 'audit' for the eHF Audit module). Those aliases however are not considered unique and interchangeable. Nevertheless, encryption configuration has definitely no interchangeable character and therefore those aliases are replaced by implementation specific module identifiers.

From Sprint 30 on modules are identified in the target mapping file using the module package name. This change has an impact on those assemblies, which already in use and configure the eHF encryption support. Inside the target mapping configuration the aliases are no longer used in the module ID, but have to be replaced by their module package names. The table below shows a mapping between the aliases and their package names.

Module Alias	Module Package Name
audit	com.icw.ehf.audit
compositon	com.icw.ehf.composition
document	com.icw.ehf.documentt
recordMedical	com.icw.ehf.record.medical
usermgnt	com.icw.ehf.usermgnt

Table1. Module alias to package names mapping

The following target mapping fragment shows the current configuration for eHF Record Medical with the module package name used as module ID.

```
<module module-id="com.icw.ehf.record.medical">
  <config-element classification="identifier">
    <parameters key-pool="pool_identifier" iv-type="scope"/>
  </config-element>
  <config-element classification="scope">
    <parameters key-pool="pool_scope"/>
  </config-element>
  <config-element classification="freetext">
    <parameters key-pool="pool_freetext"/>
  </config-element>
  <config-element classification="date">
    <parameters key-pool="pool_date"/>
  </config-element>
  <config-element classification="confidential">
```

```

        <parameters key-pool="pool_confidential" iv-type="scope"/>
    </config-element>
    <config-element classification="strictly-confidential">
        <parameters key-pool="pool_strictly_confidential" />
    </config-element>
    <key-pool id="pool_identifier"/>
    <key-pool id="pool_scope"/>
    <key-pool id="pool_freetext"/>
    <key-pool id="pool_confidential"/>
    <key-pool id="pool_strictly_confidential"/>
    <key-pool id="pool_date" engine-id="LocalOPE"/>
</module>

```

For testing purposes an ehf-target-mapping-nullcipher.xml can be provided that basically deactivates encryption on application level (please note that in various cases the results are nevertheless BASE64 encoded or underwent other format transitions). The following shows a simple default configuration:

```

<?xml version="1.0" encoding="UTF-8"?>
<target-mapping xmlns="http://www.intercomponentware.com/schema/ehf-encryption">

    <template>
        <config-element classification="scope">
            <parameters key-pool="global_pool_scope" />
        </config-element>
        <config-element classification="identifier">
            <parameters key-pool="global_pool_identifier" />
        </config-element>
        <config-element classification="date">
            <parameters key-pool="global_pool_date" />
        </config-element>
        <config-element classification="confidential">
            <parameters key-pool="global_pool_confidential" />
        </config-element>
        <config-element classification="strictly-confidential">
            <parameters key-pool="global_pool_strictly_confidential" />
        </config-element>
        <config-element classification="freetext">
            <parameters key-pool="global_pool_freetext" />
        </config-element>
        <key-pool id="global_pool_scope" />
        <key-pool id="global_pool_identifier" />
        <key-pool id="global_pool_confidential" />
        <key-pool id="global_pool_strictly_confidential" />
        <key-pool id="global_pool_freetext" />
        <key-pool id="global_pool_date" engine-id="LocalOPE" />
    </template>

    <properties>
        <property name="algorithm" value="null" />
    </properties>

</target-mapping>

```

Please note that updated default configurations can be supplied by the eHF development team.

2.2 Encryption Infrastructure Changes (Sprint 29)

Module Configuration

The initial approach to configure the `CryptoService` for each request (`StateHolder`) has caused too much trouble. As well the encryption of eHF Audit has revealed some major flaws that we could not easily overcome. Therefore the approach was changed. In the current implementation the `CryptoService` has to be imported by modules, which require encryption. The generator takes care of this, as long as you have not customized

your module context. In case that you experience a missing `CryptoService` bean configuration, you will have to include the following line into your module context:

```
<ehf:import>
  [...]
  <ehf:ref-bean
    alias="cryptoService"
    interface="com.icw.ehf.commons.encryption.api.CryptoService" />
</ehf:import>
```

In case you use a different name space, you have to adjust this line accordingly.

Furthermore you have to configure a bean named `cryptoService` in your test spring contexts. This may look as follows:

```
<bean id="cryptoService"
  class="com.icw.ehf.commons.encryption.mock.CryptoServiceMock" />
```

For more details on configuring this mock, please consult the encryption howtos. Normally this configuration satisfies your needs.

Please note that for JUnit test support (which are not spring-context-based) we still cover the fallback support as described in the encryption howtos. In this case you simply place a spring context file with the above mock configuration in your project (`src/test/resources/META-INF/ehf-commons-encryption/ehf-cryptoservice-context.xml`).

Assembly Configuration

Please note that this modification has made the `cryptoServiceHandlerInterceptor` configuration redundant. If you have the following configuration in your assembly spring context

```
<bean id="cryptoServiceHandlerInterceptor"
  class="com.icw.ehf.commons.encryption.web.CryptoServiceHandlerInterceptor">
  <property name="cryptoService" ref="encryptionModuleService" />
</bean>

<bean id="requestContextFilter"
  class="com.icw.ehf.commons.context.web.RequestContextFilter">

  <property name="interceptors">
    <list>
      <ref bean="cryptoServiceHandlerInterceptor" />
    </list>
  </property>
</bean>
```

you have to reduce it to

```
<bean id="requestContextFilter"
  class="com.icw.ehf.commons.context.web.RequestContextFilter" />
```

as the `CryptoServiceHandlerInterceptor` is obsolete and the class has been removed.

Furthermore it is necessary to add the following bean to the system spring context since eHF Audit was encrypted.

```
<bean id="auditModuleAttributeModulationConfig"
  class="com.icw.ehf.audit.domain.ModuleAttributeModulation"
  factory-method="getAttributeModulation">

  <property name="cryptoService" ref="cryptoService" />
```

```
</bean>
```

This is currently necessary, due to the way audit events are currently processed on assembly level. In the future we hope that we can live without this configuration (and respect the audit public API).

The last step to activate encryption is to configure an alias to connect encrypted modules with the encryption module:

```
<alias alias="cryptoService" name="encryptionModuleService" />
```

If you experience situations where the `cryptoService` bean cannot be found, it is either the missing alias or a missing bean import into a module context of an encrypted module.

2.3 Value Objects (Sprint 29)

One-to-many entity-to-value-object Compositions

In the ongoing discussion in eHF on value objects, we have applied some modifications to resolve issue BAS-1106. At first this has no particular impact on an eHF consumer as the change is completely transparent.

The fix of this issue has many components. One major component is the introduction of the internal/external representation on entities and to extend this capability into compositions with value objects. The duplication of objects in the database was mainly caused by a hibernate issue. This issue or better limitation is now addressed by treating `null` attributes differently in such one-to-many entity-to-value-object compositions. Internally the `null` representation is converted to an empty string (blank to make it distinguishable to an Oracle database).

Additionally a versioned template was introduced to further support this. The versioned template introduces additional `not-null` constraints on the columns, composing the value object in the database. Migrating to this versioned template also means to introduce these constraints on the respective columns and to replace `null` with a blank in the existing data (upgrade). The migration to the versioned template is described in the eHF Reference Documentation and can be performed on demand. New module (generated from the eHF project templates) will start with the latest set of templates.

In this context hibernate shows an other undesired behavior. It basically marks the collection of value objects as dirty, if the `equals` and `hashCode` methods are not implemented as hibernate desires. Currently this can be only overcome by customizing the domain objects and implementing these methods manually. Please note that we currently collect ideas for solutions to this problem. We currently have deep discussions on the behavior of value objects in general. Depending on the outcome of these discussions we may be able to further narrow or eliminate this last issue, which has no functional impact, but an expected performance penalty.

2.4 Code Meta Data Validation (Sprint 29)

Bean `codeValidationService`

The `codeValidatorService` bean has been introduced by the generator for modules, which use coded attributes. Modules with a customized module context needs to import the bean in the module context:

```
<ehf:ref-bean alias="codeValidatorService"
  interface="com.icw.ehf.commons.codesystem.CodeValidatorService" />
```

For local JUnit it may be required to configure an appropriate bean as a mock:

```
<bean id="codeValidatorService"
      class="com.icw.ehf.commons.codesystem.test.CodeValidatorServiceMock" />
```

Depending on the structure of the test context this mock may be required in multiple locations.

Class CodeMetaDataValidator

The class

```
com.icw.ehf.commons.codesystem.metadata.validator.strategy.CodeMetaDataValidator
```

has been replaced with

```
com.icw.ehf.commons.codesystem.metadata.validator.CodeMetaDataValidator
```

in conjunction with the improvement and revision of the complete code handling and validation. The new CodeMetaDataValidator takes a codeValidatorService bean as constructor parameter:

```
<constructor-arg>
  <ref bean="codeValidatorService" />
</constructor-arg>
```

Usually the according spring configuration is generated. In case you are using the old class your project has to be migrated as the old functionality is no longer supported.

Class AnnotatedCodeMetaDataValidator

Analog to the CodeMetaDataValidator the

```
com.icw.ehf.core.metadata.validator.AnnotatedCodeMetaDataValidator
```

also requires a codeValidatorService bean as constructor parameter. This change may be necessary in some custom context files.

2.5 Migration to IPF 2.0-m1 and Apache Camel 2.0.0

Namespace URL changed

We have migrated to IPF 2.0-m1 and Apache Camel 2.0.0 that requires a change in the ehf-ipf-context.xml file, since the Spring namespace of Camel has changed to <http://camel.apache.org/schema/spring>.

```
<beans xmlns="http://www.springframework.org/schema/beans"
      ...
      xmlns:camel="http://camel.apache.org/schema/spring" xmlns:util="http://www.
springframework.org/schema/util"
      xsi:schemaLocation="
      ...
      http://camel.apache.org/schema/spring
      http://camel.apache.org/schema/spring/camel-spring.xsd">
```



Note: Please bear in mind to change the namespace definition and the schema location.

Namespace usage changed

The schema change also has some effects on the namespace usage. Please verify that your `camel.route.fragment` uses

```
<camel:routeBuilder ref="[nameOfYourRouteBuilder]" />
```

instead of

```
<camel:routeBuilderRef ref="[nameOfYourRouteBuilder]" />
```

Additionally, the default behaviour of the `camelContext` definition seems to be different. With Camel 1.x, the definition

```
<camel:camelContext id="camelContext" />
```

caused a package scan to find all route builders. In Camel 2.x, this is no longer the case. In addition, using the package scan feature is discouraged.

Please change your Camel Context definition and add your route builder explicitly.

```
<camel:camelContext id="camelContext">
  <camel:routeBuilder ref="[nameOfYourRouteBuilder]" />
  <camel:routeBuilder ref="[nameOfYourOtherRouteBuilder]" />
  ...
</camel:camelContext>
```

ehf-commons-camel component modified

The ehf-commons-camel module was also adapted to work with the new API changes in Apache Camel. If some modules are using the ehf-service or ehf-resource component they also have to adapt to the new Camel API.

Dependency changes

Updates of existing dependencies (includes dependencies which are necessary to use IPF XDS component):

groupid	artifactId	version
org.apache.camel	camel-core	2.0.0
org.apache.camel	camel-http	2.0.0
org.apache.camel	camel-spring	2.0.0
org.apache.camel	camel-restlet	2.0.0
org.openehealth.ipf	commons-core	2.0-m1
org.openehealth.ipf	commons-ihe-xds-core	2.0-m1
org.openehealth.ipf	commons-ihe-xds-iti41	2.0-m1
org.openehealth.ipf	platform-camel-core	2.0-m1
org.openehealth.ipf	platform-camel-ihe-xds-iti41	2.0-m1
org.openehealth.ipf	platform-camel-ihe-xds-core	2.0-m1

Additionally the following dependencies have to be added:

groupid	artifactId	version
org.openehealth.ipf	commons-ihe-atna	2.0-m1
javax.ws.rs	jsr311-api	1.0

Migration directly to IPF 2.0-m2 should be possible without difficulty but is not tested yet.

2.6 Extension to SessionFactory Configuration

Extension to SessionFactory Configuration

Through changes made to the eHF Document module, involving the implementation of Hibernate listeners to populate the Participant objects when reading Documents, the SessionFactory configuration in the assembly needs to be extended.

The **sessionFactory** bean definition contained in the `src/main/config/ehf-system-context.xml` Spring configuration file of the assembly needs to be extended with the following property:

```
<property name="eventListeners">
  <map>
    <entry key="post-load">
      <list>
        @@@hibernate.sessionfactory.postLoadEventListeners.fragment@@@
      </list>
    </entry>
  </map>
</property>
```

3 Development Environment

The eHealth Framework provides both a runtime environment for health care solutions and an application development platform that can be used for the development of new as well as existing modules. The eHealth Framework supports and accompanies the full software development life cycle to produce lean and high quality health care applications.

Developers can turn to the provided development support and use it to enhance their productivity. The eHealth Framework supports agile and iterative development methodologies. When this approach is applied consistently, it will result in the creation of robust and sophisticated health care applications. The modules that are provided by the eHealth Framework have been developed using the eHealth Framework development environment.

3.1 eHF Build Plug-in

The build plug-in was extended to support the initialization of additional artifacts for the application. Using the properties

```
# properties controlling initialization
configuration.initialize.pattern=
configuration.initialize.policy=${basedir}/ehf-assembly/bootstrap/bootstrap.policy

configuration.processor.java.xms=768m
configuration.processor.java.xmx=768m
configuration.processor.java.xss=256k
```

the initialization can be controlled and adapted. This feature was mainly introduced in the context of application-level encryption and pseudonymization to setup a so-called key package (consisting of a key store, checksums and other encryption related information required later in the installation or deployment process).

3.2 eHF Install Scripts

3.2.1 Initialization for Encryption

According to the build plug-in also the ant installation scripts were extended to support the initialization of the application for encryption purposes.

The step can be executed using the ant target `configure:initialize`:

```
ant -f install/install.xml configure:initialize
```

from the installation process. Please note that this target is implicitly invoked from the target `configure:all`.

Details for the individual products must be described in the product installation manual.

The installation step is controlled by the properties (here with default values)

```
# properties controlling initialization
configuration.initialize.pattern=
configuration.initialize.policy=${basedir}/ehf-assembly/bootstrap/bootstrap.policy

configuration.processor.java.xms=768m
configuration.processor.java.xmx=768m
```

```
configuration.processor.java.xss=256k
```

Please note that it is normally sufficient to place only the first two properties in your product specific configuration file. Here an example was taken from the eHF reference assembly:

```
# encryption initialization
configure.initialize.pattern=classpath:/META-INF/ehf-assembly-initialize.xml
configure.initialize.policy=assembly/deploy.policy
```

Please note that all artifacts required in the classpath at initialization time must be included in the lib folder of your release zip. This may require to mark an additional dependency in your project.xml as <category>lib</category>. Here an example:

```
<dependency>
  <groupId>springframework</groupId>
  <artifactId>spring</artifactId>
  <version>2.5.6</version>
  <properties>
    <war.bundle>true</war.bundle>
    <category>lib</category>
  </properties>
</dependency>
```

In case an artifact is missing, the installation process will report that the required classes cannot be found.

3.2.2 Automation of Database Privileges

New eHF Installation

The Database privileges can now be automated by using a new ant target `ant -f install/install.xml database:privileges`

This should be run after the `database:prepare` and before the `database:bootstrap` goals.

The process will not check for errors, so it is the Database Administrators responsibility to ensure the process runs all the SQL commands correctly.

There are two parameters that must be set in the `configuration.password.properties` These are as follows:

```
# username and password for creating the database privileges
main.database.username=ehfuser
main.database.password=ehfuser
```

eHF Upgrade and Encryption (TDE)

In the situation of Upgrade and Encryption the database user needs extra DBA privilege rights to complete the process. For this purpose, two other ant targets are provided:

- `ant -f install/install.xml upgrade:user:` upgrades the database user with DBA role
- `ant -f install/install.xml upgrade:privileges:` grants the privileges and revokes the DBA role

The complete steps for Upgrade including database privileges is as follows:

- `ant -f install/install.xml configure:all`
- `ant -f install/install.xml upgrade:configure`
- `ant -f install/install.xml upgrade:prepare-remote-host`
- `ant -f install/install.xml upgrade:user`
- `ant -f install/install.xml upgrade:execute-tasks`
- `ant -f install/install.xml upgrade:finalize-remote-host`
- `ant -f install/install.xml upgrade:privileges`



Important: The following order of upgrade steps is important when using privileges as shown in the steps above:

1. `upgrade:user`
 2. `upgrade:privileges`
-



Note: The possibility still exists to use the older approach of running the `make_grant` scripts manually. In the case of upgrade, the DBA must grant the user the DBA oracle role *before* starting upgrade.

4 Application Platform

The application platform consists of ready-to-use, modular building blocks. These range from fundamental core, infrastructure, and security modules to high-level, data-centric health care-specific application modules. They provide application programming interface (API) functionality, which can be used by health care applications or custom add-on modules. Modules can expose web service interfaces so that they can participate in an orchestrated service-oriented architecture (SOA) infrastructure.

Module Usage for Production

The following list shows the modules being shipped with this eHF release but which must not be used in production:

- eHF Reference
- eHF Reference Web Service
- eHF Account
- eHF Entity Relationship
- eHF Medicine Cabinet
- eHF Commons Context
- eHF Context

The following modules may be used for productive usage but the web services must not be exposed:

- eHF Audit

4.1 Commons Module

This section summarizes all developments and enhancements which have taken place in the Commons module.

4.1.1 Domain Objects and Persistence

In preparation for the implementation of encryption and pseudonymization a change was introduced enabling the isolation of the external from the internal (or persistent) state of a domain object. The change helps us to be very explicit when dealing with the persistent entities.

The change enables

- prevention of implicit data writes from hibernate without sacrificing layering. This gives us the chance to remove some workarounds where it is necessary to detach or clone objects.
- removal of workarounds for attribute-level contract-relevance and expert-entry checks. Currently it is required to open a second hibernate session to access the persistent state.
- provision of a hook for ADD and REMOVE checks on aggregations. At the moment this is only possible on method level. But this is too restrictive.
- provision of a hook for encryption and decryption. The de-/encryption happens on transitions between internal and external representations (synchronization)
- elimination of a hibernate limitation that did not enable us to manage deleting orphans for *-to-many composites.

Those changes have an impact, because

- existing example queries will not work anymore: You have to perform an extra step on your example object to recover. The step is isolated in an alternative Example class: use

```
com.ehf.commons.hibernate.Example
```

instead of `org.hibernate.criterion.Example` and you are done.

- eHF is explicit now: code relying on implicit changes (changes being flushed to the database without an explicit update) will not work anymore. You are required to invoke an explicit update on things you like to persist. In the standard CRUD case this is not an issue. However, when modifying a persistent object in a piece of business logic and not calling update, the change will not become effective. Some logic is build in this fashion. Please note that being explicit is not a bad thing. The entire authorization concept, invariant checking (metadata validation) is based upon explicit triggers.
- transactions are required: due to the way the synchronization of internal and external representation works it is required that a unit tests runs within a transaction. Some unit test (those which are below) the adapter layer therefore may fail with the following error message: `LazyInitializationException`. Wrap the test code in a transaction and you are done.
- Issues with ordinary update signature: There is an issue with the former update signature. This needs further investigation. For 2.9 we consider removing the existing signature (or at least cease support for the old behavior). It may be necessary to switch to the update signature including the update parameter in some cases.
- You use `entity.save()`: The synchronization is connected to the EJB3.0 signatures. I.e. the behavior of `entity.persist()` and `entity.save()` is different (id generation; cascading behavior). *Currently we only support `entity.persist()`.*
- You use XStream to serialize Domain Objects: Using XStream on domain object level is not recommended, as you expose non-public classes (i.e. you persistence model) to the outside world. Nevertheless, it is recommended to derive objects that map your expectation and to use the XStream alias functions shown below on object or attribute-level, even though, you already have exposed non-public classes to the outside world.

```
xstream.alias("com.icw.ehf.document.domain.DocumentTypeMetaData",
DocumentTypeMetaDataStub.class);
```

Use the following XStream alias functions for configuring the parser:

```
xstream.aliasField("documentType", DocumentTypeMetaData.class,
 "_documentType");
xstream.aliasField("defaultStyleSheetId", DocumentTypeMetaData.class,
 "_defaultStyleSheetId");
xstream.aliasField("defaultPrintStyleSheetId", DocumentTypeMetaData.
class, "_defaultPrintStyleSheetId");
xstream.aliasField("defaultThumbnailId", DocumentTypeMetaData.class,
 "_defaultThumbnailId");
```

We are aware of this particular inconvenience and apologize.

4.1.2 Support for Code Validation

eHF modules already come with a default set of meta data validators like the `StringMetaDataValidator`. From eHF 2.9 onwards, the code validators will be part of the default meta data validators as well. If the model contains one or more coded attributes, the generator attaches the `com.icw.ehf.core.metadata.validator.AnnotatedCodeMetaDataValidator` to the generated context.

In consequence of this attachment, there is no further need to register the code meta data validator inside the Spring custom context, since it is already provided by the generated configuration. Modules having the code meta data validator defined in the custom context,

will still work as intended, but the code validator is registered twice - respectively executed twice - in each validation phase. To solve this unnecessary performance loss, you can easily remove the code meta data validator configuration inside the custom context.

4.1.3 Code System Meta Data Format Changes and Code Meta Data Validation

The tags for controlling coded attributes were refactored and improved. Together with the profile changes, the code meta data format was changed to better reflect the know concepts (namely code system, code set and code category).

The prior version of eHF treated any coded attribute meta data the same way in XML. The new meta data XML tags for coded attributes now distinguish between different code concepts. A code system is represented by the code system's OID and the optional version:

```
<code>
  <attribute>codedAttribute</attribute>
  <codeSystem oid="..." version="..." />
  ...
</code>
```

A code set is referenced through its name and the optional version information:

```
<code>
  <attribute>codedAttribute</attribute>
  <codeSet name="..." version="..." />
  ...
</code>
```

And a code category assignment for a coded attribute only contains the code category name:

```
<code>
  <attribute>codedAttribute</attribute>
  <codeCategory name="..." />
  ...
</code>
```

This change in the meta data XML has some implications on existing eHF-based modules. The meta data files are produced by the eHF Generator and a module using the meta data only indirectly through meta data validation should only experience minor adaptations.

Usage of commons beanutils

The `MetaDataProvider` implementation which obtains meta data from XML files is now based on commons beanutils. Every module which uses meta data and meta data validation should therefore add the following dependency to its `project.xml` file, if not yet there:

```
<dependency>
  <groupId>commons-beanutils</groupId>
  <artifactId>commons-beanutils</artifactId>
  <version>1.7.0</version>
</dependency>
```

Removal of codeSubsystem tag

The tag `codeSubsystem` was removed from the stereotype `ehf-attribute` and is no longer supported by eHF. If your UML model still uses this tag, we recommend to switch to one of the supported code concepts: code system or define a code set, containing only the codes of this subsystem and switch to the concept: code set.

Change in code meta data Java types

In Java, we have attribute meta data containers which hold the information which is read from the XML files. In prior eHF versions, we always used the same container named `com.icw.ehf.commons.metadata.transfer.CodeMetaData`. Since we

now are able to distinguish between different code concepts, we marked this class as abstract and provide concrete implementations (namely `com.icw.ehf.commons.metadata.transfer.CodeSystemMetaData`, `com.icw.ehf.commons.metadata.transfer.CodeSetMetaData` and `com.icw.ehf.commons.metadata.transfer.CodeCategory`). The concrete types now only carry the necessary information for a given code concept.

If you work with the `com.icw.ehf.commons.metadata.transfer.CodeMetaData` type directly in your business code or unit tests, it is likely that you will get compile errors. Instead of working with the generic `com.icw.ehf.commons.metadata.transfer.CodeMetaData`, you are forced to use the more concrete implementation according to the used code concept.

Change in code meta data validator

Together with the meta data changes, the code meta data validator was adapted as well. It now benefits from being able to distinguish the different code concepts by type. However, in the case of the code set validation, the code meta data validator has less information available. In the old version, the mapping from code set to underlying code system was produced into the XML meta data files by the eHF Generator. In the new version, an assigned code set is referenced by its name and version only. Due to this fact, the code meta data validator currently does not perform an existence check for incoming codes which are bound to a code set.

Furthermore, the eHF Generator is now able to detect whether a module contains coded attributes. If there are coded attributed, the eHF Generator will automatically add the code meta data validator (of type `com.icw.ehf.core.metadata.validator.AnnotatedCodeMetaDataAdapter`) to the list of registered meta data validators in the generated Spring context file. While now the eHF Generator is in control of the code meta data validator registration, there is no longer a need, to register a code meta data validator a second time through the custom context. Custom configuration like the following is highly recommended to be removed from your custom context:

```
<ehf:inject target-ref="<moduleName>MetaDataAdapter">
  <ehf:propRef path="additionalValidator" ref="codeValidator"/>
</ehf:inject>

<bean id="codeValidator"
  class="com.icw.ehf.core.metadata.validator.AnnotatedCodeMetaDataAdapter">
  <property name="codeSystemResolver">
    <ref bean="codeResolver" />
  </property>
</bean>
```

Leaving the registration of the second code meta data validator in your custom context leads to doubled code validation and implies avoidable performance penalties.

Extensive model checks for coded attributes

The new eHF profile comes with additional model validation constraints. So, the eHF Generator will now complain if a coded attribute is modeled without being assigned to a code concept. If your model contains such a constellation, the module build will fail with the corresponding error message. In this case, you have to adapt your model so that all new constraints do comply.

Potential adaptation for meta data enhancers

If your module uses meta data enhancers to introduce additional code meta data, you will experience some problems. The meta data enhancer works on old code meta data which is as well stored in XML. In order to successfully read the meta data XML, you have to update the XML content to the new code meta data format (see section 1 for how the format looks like for the different code concepts)

XSLT for web service backwards compatibility

Because of the change in the code meta data, the `loadMetaData` web service operation gets backwards incompatible. The module `ehf-commons` provides an XSLT (`src/main/resources/META-INF/compatibility/ehf-commons-v2.8.0-v2.9.0-response.xslt`) which restores compatibility. So, if your module exposes the `loadMetaData` operation on web service level and it has to be backwards compatible, you have to add the following statement into your backwards compatibility transformation:

```
<xsl:include href="/META-INF/compatibility/ehf-commons-v2.8.0-v2.9.0-response.xslt"/>
```

Removal of codesystem mapping file

The previous version of the eHF Generator required a file named `codesystem-mapping.properties` where OIDs are mapped to names of the code systems used in the model. With the new tag set, the mapping file is obsolete because we deal with OIDs in the model directly. Hence, the obsolete `codesystem-mapping.properties` file can be removed from your modules together with the declaration

```
<codeSystemMappingFile value="${basedir}/src/main/model/codesystem-mapping.properties"/>
```

in your `workflow.oaw`

4.1.4 Change in length determination algorithm for encrypted columns

For encrypted columns, the eHF Generator calculates the column length which goes into JPA annotations in the eHF domain objects.

The algorithm for determining the maximum size of encrypted columns was improved so that the new version calculates more restricted and shorter column lengths. This directly affects the module's schema and already written upgrade scripts. Please revise your upgrade scripts for all modules which use pseudonymization in order to not run into schema derailments. In most cases, the column alter statements need to be updated to reflect the newly estimated column length for an encrypted property.

4.1.5 Refactoring the Date/Time API

Some refactoring was done in the date/time API (`com.icw.ehf.commons.time` package). It was mainly package changes and renaming of some classes. Here is the mapping between old and new classes:

- `com.icw.ehf.commons.time.DateTimeConverter` --> `com.icw.ehf.commons.time.format.DateTimeFormatter`
- `com.icw.ehf.commons.time.AbstractDateTimeConverter` --> `com.icw.ehf.commons.time.format.AbstractDateTimeFormatter`
- `com.icw.ehf.commons.time.JodaDateTimeConverter` --> `com.icw.ehf.commons.time.format.JodaDateTimeFormatter`
- `com.icw.ehf.commons.time.MutableJodaDateTimeConverter` --> `com.icw.ehf.commons.time.format.MutableJodaDateTimeFormatter`
- `com.icw.ehf.commons.time.ISOExtendedParser` --> `com.icw.ehf.commons.time.parse.ISOExtendedParser`
- `com.icw.ehf.commons.time.ISOGenericParser` --> `com.icw.ehf.commons.time.parse.ISOGenericParser`

Three new methods for formatting date/time from timestamp to user defined formats were added in the class `com.icw.ehf.commons.time.format.DateTimeFormatter`:

- `String getCustomDateTime(java.util.Date date)`
- `String getCustomDate(java.util.Date date)`
- `String getCustomTime(java.util.Date date)`

4.1.6 Crud Services at Adapter Level and Service Level for eHF Service

This change was done as bug fix for BAS-1283. CRUD services are now also available at service level when modelling ehf-service. The service dto adapter delegates to the service that in turn delegates to the dao, so that the layering is no longer violated.



Note: In order to realize this bug fix the interface `CrudService` was renamed to `DomainObjectCrudService` to differ between crud service of domain objects and services. this change may lead to compilation errors or test failures in cases, when the interface `CrudService` is directly used.

4.1.7 Signature of `persistChild` method in `ObjectManagerAwareCrudService` changed.

Parameter `boolean requiresChildObject` was added to method

```
void persistChild(final D parent, final C child, final PersistChildCallback<D, C>
callback, boolean requiresChildUpdate)
```

in the interface `com.icw.ehf.commons.service.ObjectManagerAwareCrudService` and accordingly added in the implementing class `com.icw.ehf.commons.service.AbstractDomainObjectCrudService`.

The parameter is used to determine whether child objects should be updated when they are added to the subgraph of a root object via the `add[nameOfChildObject](...)` convenience method of the root objects' CRUD service. The `add[nameOfChildObject](...)` methods, which internally call the `persistChild(...)` method, are generated by the generator for composites and aggregates in the CRUD service of the (root)object. The update of the childobject is suppressed for aggregates in general and composites in special circumstances (see note below).

Apart from fixing BAS 1246 this also improves the performance of the `addXYZ(...)` methods, as unnecessary updates are avoided.

Note:

The generator takes care for the needed changes on generated CRUD services (in `src/main/gen`). However, if you have moved CRUD services from `src/main/gen` to `src/main/java` the necessary changes won't be made by the generator and thus have to be done manually in the `addXYZ()` methods.



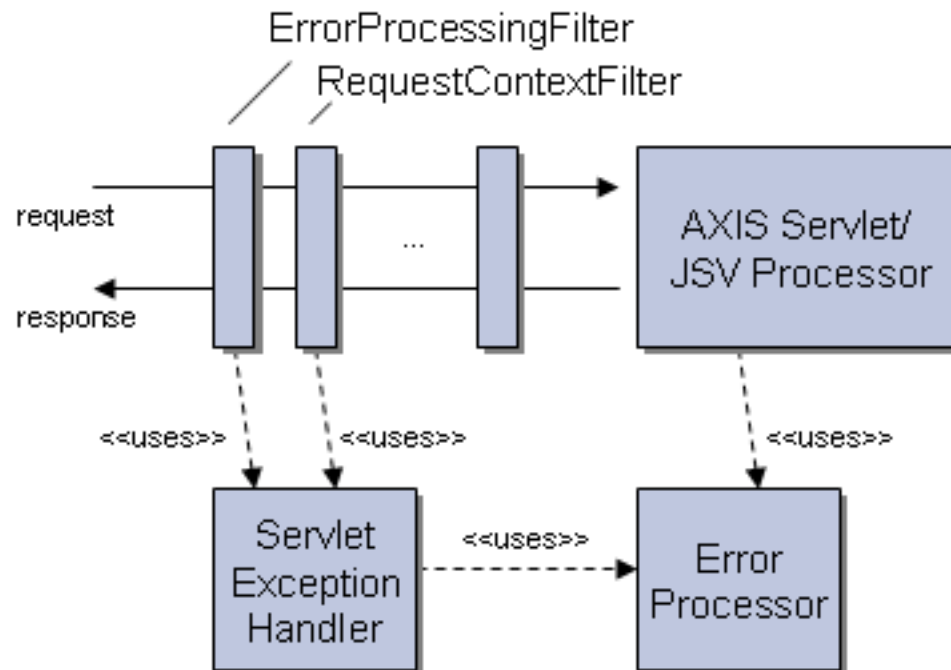
Please extend the call to `persistChild()` in these methods as follows:

- add `true` as last param in the method call if
 - the object added via this method is a composite AND
 - the objects are not modelled as a m:n relation AND
 - not modelled with 'forceJointable' AND
 - the added object IS NOT a Link.
- add `false` as last param in all other cases.

4.1.8 Changes in Error Processing Behavior

It was possible that the user received technical exceptions with full stack traces which were not handled by the Error Processor infrastructure. These exceptions usually originated somewhere in the filter chain for the http requests. This was because there was no central error sink in place handling the Exception. Common exceptions leading to the behavior were `UserDeactivatedException` or `InvalidUsernamePasswordException`.

The following figure shows the two filters in action.



The `ErrorProcessingFilter` (first filter in the chain) makes sure that no unprocessed exceptions are passed to the client. This means that all Exceptions that have not been processed before (this can happen if the Exception occurs after the standard error processing took place e.g. in the chain of filters) will be handled here using the `ServletExceptionHandler` Spring bean.

The default implementation of the `ServletExceptionHandler` handles the exception using the `Error Processor` and sets a http status code. The Result of the `Error Processor` is put in the body of the response and the the http status code in the header. If we are outside the `StateHolder Management` the `ErrorProcessor` is not available and raw error Data is emitted. The Handler applies http status codes to the response header if an exception occurs. The mapping from exception to status code can be configured in the spring context were this bean is defined. If no mapping is found the status code 500 (`SC_INTERNAL_SERVER_ERROR`) is set.

The second Filter (the `RequestContextFilter`) also uses the `ServletExceptionHandler` to resolve Errors and Exceptions from down chain. the exception handling of this filter has been adapted so that it doesn't throw exceptions to the client anymore. These formerly unhandled exceptions are no taken care of using the `ServletExceptionHandler`. Only those Exceptions that slip through here reach the `ErrorPropcessingFilter` and will be handled there.

The following code snippet is taken from the "`ehf-system-context.xml`" file, and illustrates the configuration of the various spring beans involved.

```

<bean id="requestContextFilter" class="com.icw.ehf.commons.context.web.
RequestContextFilter">
  <property name="interceptors">
    <list>
      </list>
    </property>
    <!-- Handler for any unhandled Exception -->
    <property name="servletExceptionHandler">
      <ref bean="servletExceptionHandler" />
    </property>
  </bean>

<!-- The filter that handles all unresolved Exception
could happen if the requestContextFilter fails -->
<bean id="errorProcessingFilter" class="com.icw.ehf.commons.context.web.
ErrorProcessingFilter">
  <!-- Handler for any unhandled Exception -->
  <property name="servletExceptionHandler">
    <ref bean="servletExceptionHandler" />
  </property>
</bean>

<!--
This handles error processing/mapping for all
unhandled exceptions. Is called from RequestContextFilter and
ErrorProcessorFilter.
-->
<bean id="servletExceptionHandler"
class="com.icw.ehf.commons.context.web.DefaultServletExceptionHandlerImpl">
  <!-- define Map for Exception Mapping, maps Exceptions to Http Status Codes -
->
  <property name="exceptionMapping">
    <map>
      <entry
        key="com.icw.ehf.authentication.exceptions.
UserDeactivatedException">
        <!-- HttpServletResponse.SC_UNAUTHORIZED -->
        <value>401</value>
      </entry>
      <entry
        key="com.icw.ehf.authentication.exceptions.password.
InvalidUsernamePasswordException">
        <!-- HttpServletResponse.SC_UNAUTHORIZED -->
        <value>401</value>
      </entry>
      <!-- -->
      <!-- Feel free to add more entries -->
      <!-- -->
    </map>
  </property>

  <!-- pass ErrorProcessor Bean as Property -->
  <property name="errorProcessor" ref="webserviceErrorProcessor">
  </property>
</bean>

```

4.2 Commons Entity Module

This section summarizes all developments and enhancements which have taken place in the Commons Entity module.

4.2.1 Implementation of the Entity-Provider SPI

In order to initiate an entity provider SPI, the `EntityProvider` interface has been introduced. It defines the interface in the Commons Entity Module for reading/resolving arbitrary entities. For example you can resolve an entity with the help of a given `InstanceIdentifier`. The `EntityProvider` interface publishes methods for reading only.

Currently, eHF Entity provides an implementation of `EntityProvider` through its `EntityModuleServiceDtoAdapter`. eHF User Management provides an implementation through its `UserEntityService`.

4.2.2 Implementation of the Entity-Converter SPI

In order to provide an entity converter SPI, the `EntityConverter` interface has been introduced. The entity domain model is designed in a way to abstract from the concrete usage of the module. The concrete semantic must be introduced by the module that is using the entity. For example: A user management might use the entity as storage. But the semantic that this data resembles a user is introduced by the user management, not by the entity itself. So normally you will like to convert the comparatively generic data of an entity into something more meaningful. In this example you will likely convert a person entity into a user object. With this `EntityConverter` interface we leverage reuse of converter implementations across different modules.

Currently, an implementation is realized in eHF User Management. In particular, this implementation facilitates the conversion of an user entity (i.e. an entity which is related to an authenticated user) into a participant which is a module-specific implementation of the interface `Participant` provided by eHF Core. In case the user is associated with a person, information about the person in eHF User Management is retrieved and added to the participant.

4.2.3 Implementation of the Entity-Resolver SPI

In order to provide an entity resolver SPI, the `EntityResolver` interface has been introduced. It is an interface to resolve the entity information for the currently authenticated user. It hides user management specific implementation details and provides a unified service interface for resolving user information. The `EntityResolver` interface provides methods for reading only.

Currently, eHF User Management provides an implementation of entity resolver which is based on the `UserPrincipal` of the authenticated user. The unique user identifier is used as the extension of the instance identifier.

4.2.4 Implementation of the Entity-Manager SPI

In order to initiate an entity manager SPI, the `EntityManager` interface has been introduced. This interface provides methods to create and update entities.

Currently, eHF Entity provides an implementation of entity manager through its `EntityModuleServiceDtoAdapter`.

4.2.5 Implementation of the Entity-Relationship-Provider SPI

In order to initiate an entity relationship provider SPI, the `EntityRelationshipProvider` interface has been introduced. Its purpose is to provide methods for retrieving arbitrary relationships between entities.

Currently, eHF Entity Relationship provides an implementation of entity relationship provider through its `EntityRelationshipModuleServiceDtoAdapter`.

4.2.6 Using the API of the Commons Entity Module

Programming against interfaces is a common practice to decouple a consuming class from implementation classes. Architecturally, it allows more flexibility and extensibility. For instance, by using the entity converter interface eHF Document and eHF Record Medical are no longer dependent on eHF User Management to perform user-to-participant conversions. The following shows the steps to use eHF Commons Entity API.

- Add dependency to eHF Commons Entity API: Both the providing and the consuming module need to add a dependency to eHF Commons Entity API by inserting the following lines in the respective `project.xml` file.

```
<dependency>
  <groupId>ehf</groupId>
  <artifactId>ehf-commons-entity-api</artifactId>
  <version>SNAPSHOT</version>
  <type>jar</type>
</dependency>
```

- Export implementations of eHF Commons Entity interfaces: The implementing module exports implementations of eHF Commons Entity interfaces in its `[module-name]-module-context.xml` file. For example, eHF User Management exports the beans `usermgntEntityConverter`, `usermgntEntityService` and `usermgntEntityResolver` which implement the entity converter, entity provider and entity resolver interface respectively. The beans themselves are defined in `ehf-usermgnt-runtime-context.xml` as usual (and not shown here).

```
<ehf:export>
  <ehf:bean id="usermgntEntityConverter"
    alias="usermgntEntityConverter"
    interface="com.icw.ehf.common.entity.api.EntityConverter" />
  <ehf:bean id="usermgntUserEntityService"
    alias="usermgntEntityProvider"
    interface="com.icw.ehf.common.entity.api.EntityProvider" />
  <ehf:bean id="usermgntEntityResolver"
    interface="com.icw.ehf.common.entity.api.EntityResolver" />
</ehf:export>
```

- Import implementations of eHF Commons Entity interfaces: The consuming module imports a service which is specified by the interface without an explicit dependency to a particular implementation. As mentioned in the previous step, the service may be provided by any interface implementation declared in `[module-name]-module-context.xml` files. A consuming module imports services in its `[module-name]-module-context.xml` file. For example, eHF Document imports the services specified by the entity provider, entity converter and entity resolver interface using the following lines in `ehf-document-module-context.xml`.

```
<ehf:import>
  <ehf:pattern>usermgntEntityProvider</ehf:pattern>
  <ehf:pattern>usermgntEntityConverter</ehf:pattern>
  <ehf:pattern>usermgntEntityResolver</ehf:pattern>
</ehf:import>
```

- Using implementations of eHF Commons Entity interfaces: After importing the services, the consuming module can reference to these imported services. In case of eHF Document, the imported services will be injected into the `DocumentModuleService` as declared in `document-custom-context.xml`.

```
<ehf:inject target-ref="documentModuleServiceTarget">
  <ehf:propRef path="entityProvider" ref="usermgntEntityProvider" />
  <ehf:propRef path="entityConverter" ref="usermgntEntityConverter" />
  <ehf:propRef path="entityResolver" ref="usermgntEntityResolver" />
</ehf:inject>
```

```
</ehf:inject>
```

4.2.7 Migrating to the API of the Commons Entity Module

eHF User Management provides an implementation of the entity converter interface, which converts a user entity to a participant instance. The same functionality is provided by `UserParticipantConverter` in eHF User Management. Until now, the consuming modules have an explicit dependency to this class and `PersonService` of User Management (see below the code example).

```
com.icw.ehf.document.domain.Participant dataEnterer =
    new com.icw.ehf.document.domain.Participant();
UserParticipantConverter.create(personService, dataEnterer);
```

From this release on, consuming modules shall remove the dependency to User Management by using the entity converter interface. The migration requires the following steps:

- Modify the Java source to use `EntityResolver`, `PersonProvider` and `EntityConverter` instead of `UserParticipantConverter` and `PersonService`. For example, the above code example shall be replaced by the following lines.

```
// Find the entity of the authenticated user
Entity entity = null;
InstanceIdentifier ii = entityResolver.resolveAuthenticatedUser();

// Find the instance identifier of the user entity
if (ii != null) {
    entity = entityProvider.findByIdentifier(ii);
}

// Perform the conversion
ConversionContext conversionContext = new ConversionContext(Participant.
class, dataEnterer);
dataEnterer = (Participant) entityConverter.convert(entity,
    conversionContext);
```

- Remove the dependency to `PersonService`: After adapting the Java source code, the dependency to `PersonService` can be removed from `[module-name]-module-context.xml` and `[module-name]-custom-context.xml`.
- Add mock implementations of entity resolver, provider and converter: Various unit tests in eHF Document and eHF Record Medical rely on eHF User Management to perform the user-participant conversion. After removing the dependency to User Management, these tests will now fail. Therefore, mock implementations of the entity resolver, provider and converter are required. Currently, these mock implementations are duplicated in eHF Document, Record Medical and Composition (although some slight differences exist). Depending on the modules, the mocks shall be placed to either `main` or `test` package.
- Provide bean definitions of mock implementations to the Spring context: The bean definitions of mock implementations shall be added to `[module-name]-system-context.xml` or `[module-name]-test-system-context.xml`. In case of eHF Document, the following lines have been added:

```
<bean id="usermgntEntityProvider"
    class="com.icw.ehf.document.mock.EntityProviderMock">
</bean>

<bean id="usermgntEntityConverter"
    class="com.icw.ehf.document.mock.EntityConverterMock">
</bean>

<bean id="usermgntEntityResolver"
    class="com.icw.ehf.document.mock.PrincipalEntityResolverMock">
```

4.3 Entity Relationship Module

This section summarizes all developments and enhancements which have taken place in the eHF Entity-Relationship module.

Implementation of the Commons Entity Module Interface

The eHF Entity Relationship module now implements the `com.icw.ehf.common.entity.api.EntityRelationshipProvider` interface defined by the eHF Commons Entity module. The implementation of this interface currently provides two methods for finding entity-relationship domain objects: `findByCriteria` and `findTransitive`. Please consult the JavaDoc for details.

Applications now access Entity Relationship functionality via the `EntityRelationshipProvider` interface to be compile-time independent from the Entity Relationship module.

The `EntityRelationshipProvider` interface is provided as spring bean with the id `entityRelationshipModuleService`.

4.4 User Management Module

This section summarizes all developments and enhancements which have taken place in the User Management module.

4.4.1 Implementation of the Entity Participant Converter Interface

As mentioned in the eHF Commons Entity the user management module contains the implementation of the entity converter interface. It converts a user entity into a participant. More precisely, it retrieves the information about the person which is related to the given entity, and embeds it into the target participant object.

4.4.2 Implementation of the Entity Provider Interface

As mentioned in the eHF Commons Entity the user management module contains the implementation of the entity provider interface through its `UserEntityService`. It returns a `UserEntity` to the given instance identifier.

Note, that at the time of writing the implementation of eHF User Management does not persist user entities in the database. Hence, a "virtual" `UserEntity` is constructed to wrap the given instance identifier. User Service of eHF User Management is utilized to ensure that the user exists, otherwise a null will be returned.

4.4.3 Implementation of the Entity Resolver Interface

As mentioned in the eHF Commons Entity the user management module contains a principal-based implementation of the entity resolver interface through its `PrincipalEntityResolver`. It takes the `UserPrincipal` of the current authenticated user, gets the unique identifier of the user, and constructs an instance identifier for the associated user entity.

4.4.4 Deprecation of the payment data

All payment related signatures have been deprecated due to legal constraints. Payment related data has to be stored in dedicated and certified systems. The classes `Payment`,

`BankAccount`, `CreditCard` has been deprecated. The related fields (getters and setters) in the `Person` class are also deprecated.

4.4.5 User secret history

History of the passwords is stored in `T_UserSecretHistory` table. It contains digest, type, value and create date. This structure makes possible usage of the table for history of all types of user secrets. The current implementation works only with passwords, which syntax policy is not `ONE_TIME_PASSWORD`.

The client of the module should add permissions of user profile for crud operations with the history. The permission classifier is `com.icw.ehf.usermgnt.domain.UserSecretHistory`.

Syntax policy is extended with two fields:

- `reuseCount`- the count of previous password which could not be used
- `reusePeriod`- the time period in which one password could not be used twice

4.5 Encryption Support

This section summarizes the latest developments to provide an encryption infrastructure for application-level encryption and pseudonymization.



Note:

The document *ehf-howto-activate-encryption-in-ehf-based-module-0.1.pdf* describes in detail how you activate encryption for an eHF-based module.

The document *ehf-howto-integrate-encryption-into-an-assembly-0.1.pdf* describes in detail how you activate encryption in an assembly.

The document *ehf-howto-upgrade-encrypted-module-0.1.pdf* describes in detail how you upgrade a non-encrypted database to an encrypted database.

The following list shows the modules which together provide the encryption infrastructure.

eHF Commons Encryption

eHF Commons Encryption defines the API of the encryption infrastructure and provides a number of utility classes.

The API defines the basic classes and cryptographic services.

- The basic classes are, for instance, the target and result of a cryptographic operation and encryption-related exceptions.
- Cryptographic services provide, for instance, encryption and decryption functionalities, keystores which are repositories of cryptographic keys as well as key management functionalities. Currently, eHF Commons Encryption provides a file-based implementation of keystore, which utilizes Sun's `java.security.KeyStore`.

The utility classes provides a set of converters for type conversions and a mock crypto service for testing purposes

- Cryptographic engines which perform the actual cryptographic operations take plain texts as byte sequences and return cipher texts also as byte sequences. However, application data to be encrypted is often of types such as `String`, `Integer`, etc. Hence, a

conversion of target values to byte sequences is required before the actual encryption. After a decryption, the resulting plain text byte sequence is converted to the original type of the plain text. The utility classes in eHF Commons Encryption provide the functionality of type conversions.

- The mock crypto service for testing purposes prepends a prefix `<enc>` and appends a suffix `_enc` to the plaintext. For a plaintext which is encrypted with an instance-specific key, an instance-specific suffix `{id}` is appended, whereas `id` is the identifier of the instance in the database. For a plaintext which is encrypted with a scope-specific key, a scope-specific suffix `<scope>` is appended, whereas `scope` is the scope related to the plaintext.

eHF Encryption

This module provides the implementation of the encryption and decryption services specified by the eHF Commons Encryption API. Internally, it maintains the cryptographic parameters and utilizes various cryptographic engines. The current implementation utilizes cryptographic engines provided by Sun Java Cryptography Extension (JCE) and Bouncy Castle. Additionally, a local engine has been developed to support order preserving encryption of numeric data such as time stamps.

eHF Key Tools

When you have annotated your model with classifications for relevant-relevant attributes and created the configuration file `target-mapping.xml` to map these classifications to cryptographic parameters, the following steps necessary to put pseudonymization into production.

- Initialization step: This step initializes the encryption infrastructure. Taking the configuration file `target-mapping.xml` as input, a keystore is created, required keys are derived, generated and filled into the keystore. Key references associated with these keys are created for bootstrapping the key reference table of eHF Encryption. Additionally, a file `target-mapping.mac` is created which consists of the message authentication code (MAC) of the `target-mapping.xml` file to protect its integrity. Moreover, these files are distributed to the respective locations specified by marker files such as `.keystore` for the keystore, `.keyreferences` for the key reference file, `.targetmapping` and `.mac` for the target-mapping file and its MAC.
- Bootstrap step: This step bootstraps the database of eHF Encryption, in particular the key reference table.
- Upgrade step: This step provides upgrade support for migrating a unencrypted database to an encrypted database.

eHF Key Tools is a development module. It provides support for the initialization and upgrade step. eHF Build Tools has been extended to support the initialization step. The initialization step utilizes a `KeyPackageCreationProcessor` to create a `keypackage` consisting of the following files:

- `keystore.keys`: This file serves as a repository of keys. It contains the keys required by the modules which are derived from the `target-mapping.xml` file. The keystore is protected by a keystore password which is referred to as a primary master key. A keystore can only be opened with the correct primary master key provided by a master key provider.
- `key-references.xml`: This file consists of the key references associated with the keys in the keystore. Each key in the keystore is encrypted by an individual key password. `key-references.xml` stores these passwords which are further encrypted with a secondary master key provided by a master key provider.
- `target-mapping.xml` and `target-mapping.mac`: To verify the integrity of `target-mapping.xml`, its digest is signed with the secondary master key. The signed digest of a file is referred to as its message authentication code (MAC). MAC of `target-mapping.xml` is stored in `target-mapping.mac`.

After creating the key package, the initialization step utilizes a `KeyPackageDistributionProcessor` to copy the content of the key package to the different locations specified by marker files.

The upgrade step utilizes the encryption service of eHF Encryption and performs the necessary database operations to read the unencrypted data and write the encrypted results. This step is integrated into the migration process of eHF Migration Tools. Application developers only need to provide a configuration file specifying the encryption-relevant attributes for each domain object class.

eHF Medicine Cabinet

Moreover, eHF Medicine Cabinet has been introduced as a reference module to validate the encryption infrastructure and illustrate the encryption and pseudonymization concept.

4.6 Encryption Support and Harmonization of Personal Data

Removed separate storage of XML and TXT content

Up to now binary, XML and text content were stored in separate tables in the database. Encrypted content is due to technical reasons converted into binary content.

4.7 Handling for DocumentIdentifier of DocumentContentContext

The following table describes the new handling of DocumentIdentifier (docId). The user can set the attributes of the docId in different ways. In general the system takes the attribute values as they are passed by the user. There is one exception when not passing a root attribute value but still providing the extension attribute in this case an exception is thrown. Another special case is if the user does neither provide a value for the root nor extension attribute then the system will generate valid values on its own. See the following table for more information.

Case	DocId Attribute	Input	Handling
1	root	"root"	"root"
'	extension	"ext"	"ext"
2	root	null	throws InvalidDocumentQualifierException
'	extension	"ext"	throws InvalidDocumentQualifierException
3	root	"root"	"root"
'	extension	null	null
4	root	null	[generated value]
'	extension	null	[generated value]

Table2. DocumentIdentifier handling

4.8 Pseudonymization and Harmonization

The (non-pseudonymized) schema of a eHF 2.8 database can be upgraded to the eHF 2.9 version where eHF Pseudonymization is enabled (only for Oracle databases). On the database table level, eHF Pseudonymization means (application-level) encryption of specified columns. Generally, this requires the change of the database schema because the type and/or the length of a table column could be different when it is encrypted (e.g. a column of type number becomes a varchar2, etc.).

A prerequisite for the pseudonymization upgrade of each module is the harmonization upgrade which is also provided with eHF 2.9. The harmonization upgrade migrates personal data (that is all participant data) to the eHF Entity module and stores references (that is Instance Identifiers) to the entity data in the module. To be more specific, these instance identifiers are stored in the relevant tables which have references to participants (e.g. T_MEDICATION in record-medical, T_COMPOSITION in composition etc.). This of course leads also to a change of the eHF 2.9 schema compared to the eHF 2.8 schema of a module.

The eHF 2.8 to eHF 2.9 upgrade process is controlled by the `upgrade.tasks` parameter of the `configuration.product.instance.properties` file. The `upgrade.tasks` parameter lists the individual control files of the upgrade steps in execution order.

4.9 Commons-Code-System Module

4.9.1 Refactoring of the Codesystem-API

The Code System API was completely revised. The classes and interfaces were restructured according to the `CodeQualifier` hierarchy. The `CodeQualifier` is used for identifying the the appropriate concept.

New Code Services

The goal of this re-engineering was to accentuate the service orientation in the Code System module. The approach is a turn from a data or domain specific design towards a functional, service-oriented implementation. The service concepts are now reflected in the method names:

- `CodeResolverService` for resolving and searching translated codes.
- `CodeFinderService` for finding codes and the related code concepts. This service for example offers a `find`, `findCodeSets` and `findCodeSystems` method.
- `CodeValidatorService` for validating the concepts is currently in an ongoing state. It offers an `exists` and an `elementOf` method for verifying the existence and relations of certain concepts.

The common signature for the `find`, `resolve` and `search` methods suites the following pattern:

```
CodeContainer service(CodeKeyCollection, CodeCriteria)
```

Introducing the CodeCriteria Paramter

The signature above shows the `CodeCriteria` object as a parameter to restrict or specify the result set which is returned by the `CodeKeyCollection`. The `CodeCriteria` comes with a default value configuration which can be adapted, in order to restrict the result set.

One major achievement of this `CodeCriteria` parameter is the flexibility of its configuration: with the properties it is possible to adapt the default configuration of the `CodeCriteria` values even system-wide.

Please refer to XYZ for more details on the `CodeCriteria` values.

Container-wrapped Result Set

The result of the service request is wrapped in a container, in order to deliver a comprehensive and complete response.

The returned result set contains not only the requested set of concepts - being restricted with the `CodeCriteria` parameter, but the result additionally contains the information that

there are eventually more concepts available, which were not found, because of the set CodeCriteria parameters.

The following table shows the use cases for which the API provides methods. It is a comparison of the former API and the new API.

Use Case	Old API	New API
Resolve a code	yes	yes
Resolve a code set	restricted	yes
Resolve a code system, category	yes	yes
Search a code	yes	yes
Search a code set	restricted	yes
Search a code system, category	yes	yes
Find a code	no	yes
Find a code set, system, category	no	yes
Validate a code	yes	no
Validate a code range	restricted	no
Validate a code set, system, category	yes	no
Use external terminology services	yes	no

Table3. Comparison of Use Cases for the API

Comparison of Refactored Methods

The following table shows the mapping between the old methods and their new counterparts:

Old API	New API
CodeResolver.resolveCode	CodeResolverService.resolve
CodeResolver.resolveCodes	CodeResolverService.resolve
CodeResolver.query	CodeResolverService.search
CodeResolver.queryByKey	CodeResolverService.search
CodeResolver.validateCode	CodeValidatorService.exists

Table4. Comparison of Refactored Code-Resolution Methods

Old API	New API
CodeSystemResolver.resolveCodeSystems	CodeFinderService.findCodeSets
CodeSystemResolver.resolveCodeSystem	CodeFinderService.findActiveCodeSets
CodeSystemResolver.validateCodeSystem	CodeValidatorService.elementOf
CodeSystemResolver.validateCodeSystem	CodeValidatorService.exists

Table5. Comparison of Refactored Code System - Resolution Methods

Old API	New API
CodeSetResolver.resolveCodes	CodeResolverService.resolve

Old API	New API
CodeSetResolver.resolveCodeSets	CodeFinderService.find
CodeSetResolver.validateCodeSet	CodeValidatorService.exists
CodeSetResolver.validateCodeSet	CodeValidatorService.elementOf
CodeSetResolver.validateCode	CodeValidatorService.elementOf

Table6. Comparison of Refactored Code Set - Resolution Methods

Old API	New API
CodeCategoryResolver.resolveCodes	CodeResolverService.resolve
CodeCategoryResolver.validateCodeCateg	CodeValidatorService.exists

Table7. Comparison of Refactored Code Category - Resolution Methods

More details on the changes to the Code System API can be found soon in the respective chapter of the *eHF Reference Documentation*.

4.10 eHF Assembly

This section summarizes all developments and enhancements which have taken place in the eHF assembly.

4.10.1 Change of Configuration Structure

A new `configuration` directory has been introduced. The existing directories of `upgrade`, `encryption` and `database` have been moved to this new directory. In addition, all properties files previously contained in the parent or root directory excluding `configuration.properties` and `configuration.properties.template` have been moved to this new `configuration` directory.

The new configuration directory currently looks like this:

- `configuration/database/encryption`: contains the encryption scripts
- `configuration/database/upgrade`: contains the upgrade scripts
- `configuration/database/scripts`: contains the database specific privileges scripts
- `configuration` : contains the properties files

4.10.2 Automation of Database Privileges

The Database privileges can now be automated by using a new ant goal `ant -f install/install.xml database:privileges`

This should be run after the `database:prepare` and before the `database:bootstrap` goals.

The process will not check for errors, so it is the Database Administrators responsibility to ensure the process runs all the SQL commands correctly.

4.10.3 Migration of Upgrade and Encryption scripts to the modules

As well as adding a new `configuration` directory, the upgrade and encryption scripts have also been moved to the eHF modules. These scripts are aggregated during the call of `ehf:release` from the various modules and copied to the release artifact.

The layout of the upgrade and encryption scripts remains the same within the module as was it would be in the assembly:

- `configuration/database/encryption`: contains the encryption scripts
- `configuration/database/upgrade`: contains the upgrade scripts

Upgrade scripts contained in modules should be configured in the `upgrade-ehf-[modulename].xml` also maintained in the module. For example, if codesystem has a new upgrade script for eHF-2.8-2.9, the user should create the script in the codesystem module at `configuration/database/upgrade/ehf-2.8-2.9/ehf-codesystem` and reference the script in the `configuration/database/upgrade/ehf-2.8-2.9/upgrade-ehf-codesystem.xml` file.

The `upgrade-ehf-[modulename].xml` script contained in the module can be called in three different ways from the assembly:

- Using the ANT `upgrade:execute -D[taskname] target`
- Using the ANT `upgrade:module-upgrade target`
- Adding the `upgrade-ehf-[modulename].xml` script reference to an assembly Migration XML file

Execute Module Upgrade using `upgrade:execute -D[taskname]`

This is the recommended way for production environments. In this case, all module upgrade scripts i.e. `upgrade-ehf-[modulename].xml` should be executed before other scripts such as harmonization or encryption scripts.

For example, the codesystem script `upgrade-ehf-codesystem.xml`, the user should use: `ant -f install/install.xml upgrade:execute -Dtask=local:upgrade-ehf-codesystem.xml`

Execute Module Upgrade using `upgrade:module-upgrade`

The ANT property `database.task.execution.frommodule.upgrade` should be set to true before calling `upgrade:module-upgrade`.

The `upgrade:module-upgrade` is included as a part of `upgrade:all` where `database.task.execution.frommodule.upgrade` is set to true by default. This is intended for development environments.

The `upgrade:module-upgrade` target searches for a pattern defined by the ANT property `database.task.execution.frommodule.upgrade.pattern` (e.g. `**/upgrade-ehf-*.xml`), then appends "local:" to all scripts found and executes all of these as upgrade tasks.

Note: Currently all tasks found are sorted alphabetically. If you wish to ensure a script is run before another either ensure the script is numbered or use the next approach.

Adding the `upgrade-ehf-[modulename].xml` script reference to an assembly Migration XML file

A new Migration Step `com.icw.ehf.migration.steps.MigrationConfigStep` has been introduced that allows XML Migration scripts to be executed from other Migration XML scripts.

Note: Make sure the name of the Migration XML script is different that the `database.task.execution.frommodule.upgrade.pattern` pattern if you wish to use `upgrade:module-upgrade` as well.

To add the `upgrade-ehf-codesystem.xml` to a `MigrationConfigStep` simply configure as the following making sure to add the *path* and *task*:

```
<com.icw.ehf.migration.steps.MigrationConfigStep>
<path>@upgrade.path@</path>
<task>local:upgrade-ehf-codesystem.xml</task>
```

```
</com.icw.ehf.migration.steps.MigrationConfigStep>
```

Furthermore, in the assembly add this new Migration XML script e.g. ehf-1-assembly.xml (notice the name does not conflict with the database.task.execution.frommodule.upgrade.pattern pattern) to the upgrade.tasks. For eHF this is defined in configuration.product.instance.properties.

```
upgrade.tasks=\
    local:ehf-1-assembly.xml,\
    local:upgrade-record-medical-harmonization.xml,\
    local:upgrade-initialize-encryption.xml,\
    local:upgrade-medical-cabinet.xml,\
    local:upgrade-record-medical.xml
```

5 Fixed Change Requests

This section provides a tabular view of change requests for the current eHF release. The first sorting criterion is the *priority* and then the *key*.

5.1 Sprint 29 and 30

Changes listed in the following sections contain the changes since the last delivery from 09/30/2009 to 10/26/2009.

5.1.1 High Priority Change Requests

So far now high-priority known change requests occurred.

5.1.2 Medium Priority Change Requests

KEY	SUMMARY	PRIORITY	COMPONENT
BAS-1484	[Codesystem import process] Migrate the process to the new codesystem API	Medium	eHF Codesystem
BAS-1471	Upgrade IPF to version 2.x	Medium	eHF Commons Camel
BAS-1385	Enable the extending of existing codes by additional language translated terms and synonyms (at bootstrap time)	Medium	eHF Codesystem
BAS-1380	Migrate the eHF document module to the the new codesystem API and the eHF profile 1.3	Medium	eHF Document
BAS-1379	Migrate the eHF record-medical module to the UML profile eHF 1.3	Medium	eHF Record Medical
BAS-1345	Update hibernate patch version in all project.xml for eHF	Medium	none
BAS-1323	Make InstanceIdentifier.extension optional	Medium	eHF Core
BAS-1262	when there is a bug in a codesystem there the stacktrace gives no hint where the error occurred	Medium	eHF Codesystem

5.1.3 Low Priority Change Requests

So far now low-priority known change requests occurred.

5.2 Sprint 24 to 28

5.2.1 High Priority Change Requests

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1387	Enable the assigning of the same code set to several categories	High	eHF Codesystem
BAS-1381	Revise the code meta data validation	High	eHF Core
BAS-1318	#CSR: Add new health risk code for "chron. Nierenerkrankung/ nephrot. Syndrom"	High	eHF Codesystem Repository
BAS-1313	[Bulgarian values] Add the Bulgarian values for new codes in eHF 2.8	High	eHF Codesystem Repository
BAS-1281	#CSR: Add document code in code system DocumentCode and provide in code sets for all	High	eHF Codesystem Repository
BAS-1132	[CH Localization] Revise French and Italian values due to LifeSensor UI review	High	eHF Codesystem Repository
BAS-1123	[CH localization, Insurance] Insurance type values must be changed	High	eHF Codesystem Repository

5.2.2 Medium Priority Change Requests

KEY	SUMMARY	PRIORITY	COMPONENT
BAS-1421	Allergies: Rename qualifierCode to valueQualifierCode	Medium	none
BAS-1365	Observation: Rename qualifierCode to valueQualifierCode	Medium	none
BAS-1364	Observation: Rename negationIndicator -> valueNegationInd	Medium	eHF Record Medical
BAS-1363	RecordMedical: Act: Changes in record-Medical for diagnosis: 1. uncertaintyCode 2. date of diagnosis 3.actionNegationInd	Medium	eHF Record Medical
BAS-1359	Enhance Act with moodCode and performingTime and provide Medication.priority	Medium	eHF Record Medical
BAS-1324	Provide InstanceIdentifiers for Act in record-medical plus the corresponding finder method	Medium	eHF Record Medical
BAS-1320	Modify Encounter: relax Constrain, add ServiceDeliveryLocation, codecategory for Encounter.code	Medium	eHF Record Medical

KEY	SUMMARY	PRIORITY	COMPONENT
BAS-1312	[CSR: Document type] Update the German value for the doc type "1.14"	Medium	eHF Codesystem Repository
BAS-1254	[Codesystem AppointmentCategory] Change the English value for code "130.160"	Medium	eHF Codesystem Repository
BAS-1178	[Observation] Values for observation codes and tupel codes must be changed	Medium	eHF Codesystem Repository
BAS-1174	MedicalProcedure.value (in model) should have codeSystemValidation flag set instead of codeValidation	Medium	eHF Record Medical
BAS-1137	[Validation] Check the length of the code values at check-in time	Medium	eHF Codesystem Repository
BAS-1122	[CH localization] The codesystems must provide de-CH specific values for values with 'ß' characters	Medium	eHF Codesystem Repository
BAS-1102	Code Category, Set(s) and System(s) for Substance.value	Medium	eHF Codesystem Repository
BAS-1101	Code Category, Set(s) and System(s) for Substance.classifier	Medium	eHF Codesystem Repository
BAS-909	#Codesystem: Values longer than 256 characters must be stored	Medium	eHF Codesystem
BAS-836	Terminology server for procedure catalogue (OPS) needed	Medium	eHF Codesystem
BAS-227	enforce use of timezone when time information is entered via WebServices API [eHF-1281]	Medium	eHF Record

5.2.3 Low Priority Change Requests

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-812	[CSR Family History] New codes for personal relationship	Low	eHF-Codesystem Repository

6 Fixed Issues

This section provides a tabular view of fixed issues for the current eHF release. The first sorting criterion is the *priority* and then the *key*.

6.1 Sprint 29 and 30

Fixed issues listed in the following sections contain the changes since the last delivery from 09/30/2009 to 10/26/2009.

6.1.1 Fixed High Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1439	Delete issued by LifeCycleManager fails in case of class inheritance	High	eHF Commons
BAS-1303	UploadDocumentPart fails to load the author attributes of a children from entity	High	eHF Document

6.1.2 Fixed Medium Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1480	Assembly with nullcipher configuration cannot be built because MedicineCabinetIntegrationTest fails	Medium	eHF Assembly
BAS-1436	NPE in ChecksumDataCollectorVisitor method visitDevice	Medium	eHF Entity
BAS-1408	Proxies for exported beans do not carry any target class information	Medium	eHF Commons
BAS-1406	Index Generation - Creates uncompileable code when custom indexes and association indexes are applied in the same class	Medium	eHF Generator
BAS-1378	The setting of the values of scope and userId of participants throughout Document and RecordMedical is inconsistent after harmonization.	Medium	eHF Composition, eHF Document, eHF Record Medical
BAS-1372	IPF inside: assembly fails	Medium	eHF Assembly, eHF Commons

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1344	ReferenceResolver in ehf-composition should not throw an exception when empty set of references is provided	Medium	eHF Composition
BAS-1333	An attribute of an embedded object is never encrypted if it has a 1 to many composition relation to the parent.	Medium	eHF Record Medical
BAS-1321	WEBAPI / Reference / create(ExternalType): Attribute externalType.externalComplexType is not stored	Medium	eHF Reference
BAS-1302	Code validation error messages have changed.	Medium	eHF Commons, eHF Commons Codesystem, eHF Generator, eHF Testclient
BAS-1301	Record Medical: upgrade scripts adjustments	Medium	eHF Record Medical
BAS-1300	Medicine Cabinet: upgrade scripts adjustments	Medium	eHF Medicine Cabinet
BAS-1292	Validation doesn't work for code categories	Medium	eHF Codesystem, eHF eHF Commons Codesystem
BAS-1246	Incorrect cascade update on grandchild object when adding child object to parent	Medium	eHF Commons, eHF Generator, eHF Reference
BAS-1228	Editing any medical data entry by changing performer data leads to exception	Medium	eHF Commons
BAS-1106	CLONE -CLONE - memory leak - eHF 2.9 and onwards	Medium	eHF Record Medical

6.1.3 Fixed Low Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1079	DocumentModuleServiceSecure stopped working	Low	eHF Document

6.2 Sprint 24 to 28

6.2.1 Fixed High Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1327	Missing Transaction Context for eHF-Configuration	High	none
BAS-1288	ReadParticipantEventListener is not triggered	High	eHF Commons, eHF Entity, eHF Record Medical
BAS-1246	Incorrect cascade update on grandchild object when adding child object to parent	High	eHF Commons, eHF Generator, eHF Reference
BAS-1216	nested objects in objectListener do not have actual values	High	eHF User Management
BAS-1210	nested objects in objectListener do not have actual values	High	eHF Generator
BAS-1200	phr-communication does not work after update from ehf-2.8 to stable (2.9)	High	eHF Generator
BAS-1199	CodesystemModuleServiceDtoA requires existing transaction	High	eHF Codesystem
BAS-1198	The new hibernate eHF based Example class produces wrong example object	High	eHF Commons
BAS-1190	Concurrency issue on module's context initialization	High	eHF Commons
BAS-1146	Column length for encrypted attributes are too small	High	eHF Core
BAS-1131	Web content is not overwritten during customization	High	eHF Customization Maven Plugin
BAS-1129	loadMetaData throws Exception	High	eHF Document
BAS-1125	# WEBAPI / record-admin/ after correction of testclient UpdatePolicyOrProgramMethod updates test run into exceptions "Adder method for association xxx can not be invoked!"	High	eHF Record Admin

6.2.2 Fixed Medium Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1373	Setting of DocumentIdentifier of DocumentContentContextXto is ignored by the Document Module.	Medium	eHF Document
BAS-1332	Decryption of values with HyperGeometricNumericOPE fails	Medium	eHF Encryption

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1331	Sorting by date does not work with HyperGeometricNumericOPE	Medium	eHF Encryption
BAS-1319	ORA-01450: maximum key length (6398) exceeded: for DocumentContainer	Medium	eHF Document
BAS-1314	Generated setter method in an Embedded Object for an attribute of type List does not work	Medium	eHF Generator, eHF Reference
BAS-1310	Constraint violation checking that the begin date of a medication or diagnosis is not in the past of the end date is not forwarded through the web service interface	Medium	eHF Record Medical
BAS-1304	Generator - ehf-service custom methods incorrectly convert return domain object in adapter layer	Medium	eHF Generator, eHF Reference
BAS-1299	import of ErrorProcessor not possible since it does not implement any interface	Medium	eHF Assembly, eHF Commons
BAS-1298	[Codesystem Issues] Wrong value in mime type code system and wrong key in allergy code system	Medium	eHF Codesystem Repository
BAS-1289	Record-Medical: Participant Scope is missing in Web Service Response	Medium	eHF Record-Medical
BAS-1283	Generator - ehf-service crud tag, the crud services are only accessible via the adapter level and not the service level	Medium	eHF Generator, eHF Reference
BAS-1274	maximum length test for uploadDocumentPart and uploadDocumentTree failed	Medium	eHF Document
BAS-1269	Not possible to persist secured=false domain objects	Medium	eHF Commons, eHF Reference
BAS-1252	[Codesystem import process] Out of memory if code system with 30.000 codes is imported	Medium	eHF Assembly, eHF Codesystem ,eHF Codesystem Repository, eHF Document
BAS-1249	The isNull() check in the Date object in the ehf-core should check for empty isoDate value	Medium	eHF Core

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1245	WEBAPI / Document / downloadDocumentContentWith fails when downloading virus infected document	Medium	eHF Document
BAS-1235	checkConstraint should return false on non invariant constraint	Medium	eHF Commons
BAS-1233	checkConstraint should return false on non invariant constraint	Medium	eHF Commons
BAS-1227	loadMetaData for non-existing objectQualifier thrown unspecified error	Medium	eHF Assembly, eHF Commons
BAS-1226	Unhandled NullPointerException in RecordModuleService while updating Medication	Medium	eHF Record Medical
BAS-1217	CLONE -GUI> Medical Data> Name in "Last Changed by" is not updated	Medium	eHF Core
BAS-1215	Exception when query from code system returns more than 500 entries for a pageQualifier with pageSize divisible by 500.	Medium	eHF Codesystem Repository
BAS-1205	[CodeSystem] Wrong code example in eHF reference documentation	Medium	eHF Codesystem
BAS-1194	WEBAPI / Account / deleteAccount: all tests result in "ORA-00942 table or view does not exist" on Tomcat log but only when run against an upgraded DB	Medium	eHF Testclient
BAS-1184	removeChild methods on parent Crud service do not work - because Hibernate PersistentSet does not honor hashCode>equals contract	Medium	eHF Commons
BAS-1182	# WEBAPI / medical procedure /trunk, stable, ehf2.8: create medical procedure returns VAL-000313 when providing invalid value-code	Medium	eHF Record Medical
BAS-1177	Oracle driver shall not be part of eHF.war or eHF.zip (license constraint)	Medium	eHF Assembly
BAS-1173	professional user without delete rights can delete a document	Medium	eHF Document
BAS-1172	# WEBAPI / medication +vaccination / BWC	Medium	eHF Record Medical

KEY	SUMMARY	PRIORITY	COMPONENTS
	Charly1,ehf22,ehf2.8 - > ehf2.9: SAXException: Invalid element in com.icw.record.service.Manufact - changeDate		
BAS-1171	Current make grants concept does not take different data sources into account.	Medium	eHF Record Medical
BAS-1169	Potential wrong property replacement in grant-schema- permissions.sql and create- datasource-user.sql	Medium	eHF Assembly
BAS-1167	[ErrorCode] Inconsistent definition of keys for several locales	Medium	eHF Codesystem Repository
BAS-1165	# WEBAPI / STABLE_BRANCH/ record-admin / create EmergencyContact: creating contact without telecom.value (number) returns wrong exception	Medium	eHF Record Admin
BAS-1150	Input Validation also affects loadMetaData / loadByScope Method	Medium	eHF Commons
BAS-1149	BeanMapping missing for ProcedureXto in Record WSDL	Medium	eHF Record
BAS-1145	Tokens in grant-schema- permissions.sql are not replaced	Medium	eHF Assembly
BAS-1144	eHF Customization Plugin copies transformation files	Medium	eHF Customization Maven Plug-in
BAS-1139	Behaviour of EHF-BMD-0215 testcases is not deterministic concerning the validation error messages that are reported.	Medium	eHF Commons
BAS-1136	BED is not deletable after an EXP uploaded one.	Medium	eHF Document
BAS-1135	Problem with managing Date and TimeInterval objects	Medium	eHF Core
BAS-1133	Undesired behavior of the Date object constructor	Medium	eHF Core
BAS-1130	A PolicyHolder (Contact) with an Organization is attached to a PolicyOrProgram even though a VaildationException is thrown.	Medium	eHF Record Admin

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1128	# WEBAPI / record-admin/ create HealthcareProvider: organization.address.useCode does not have default "WP"	Medium	eHF Record Admin
BAS-1127	Date values that are set to null lead to unexpected behaviour during validation.	Medium	eHF Record Medical, eHF Generator
BAS-1115	# WEBAPI / record- medical/update nominal observation: update bloodgroup observatiuon by user with pharmacist profile succeeds	Medium	eHF Record Medical
BAS-1113	LoadObservationViewsByScope returns IllegalArgumentException if Code for ObservationViews are not setted.	Medium	eHF Record Medical
BAS-1110	# WEBAPI / record-admin/ update emergency contact: update with oversize telecom.value does not fail, but updates only first valid telecoms and removes rest	Medium	eHF Record Admin
BAS-1107	ClamAV repeatable scans are not possible with the same content scanner	Medium	eHF Content Scanner
BAS-1095	Build fails on first build after checkout	Medium	eHF Build Maven Plugin
BAS-1093	On loadMetadata, the Date object is updated	Medium	eHF Commons
BAS-1092	Reference not cleared from container if document was deleted	Medium	eHF Composition, eHF Document
BAS-920	Auto-Generation of Make- Grants.sql Script does not include Encryption	Medium	eHF Assembly, eHF Generator

6.2.3 Fixed Low Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1170	create-datasource-user.sql should be integrated into the grant-schema-permissions.sql	Low	eHF Assembly

7 Known Issues

This section provides a tabular view of known issues for the current eHF release. The first sorting criterion is the *priority* and then the *key*.

7.1 Known High Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1541	Synchronization between internal/external state doesn't work	High	eHF Commons
BAS-1537	Upgrade of CLOB fields for pseudonymization doesn't not work	High	eHF Encryption
BAS-1535	Participant is not visible when documents are loaded	High	eHF Assembly, eHF Document
BAS-1532	BWC Code Validation: Wrong code used in BWC transformation eHF 2.4 - 2.5	High	eHF Record, eHF Admin
BAS-1455	The pseudonymization migration shows random behaviour	High	eHF Commons
BAS-1266	Update method is not working for composition in ModuleServiceAdapter	High	eHF Commons

7.2 Known Medium Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1540	In update/updateObjects() setting description of VacTypes/AdversR. to null, the server still returns blanks.	Medium	eHF Record Medical
BAS-1531	Application Event for Audit - Update User event must not logged when user session is closed	Medium	eHF Usermanagement
BAS-1529	No detailed information for invalid ehf-transformation BeanBinder configuration	Medium	eHF Transform
BAS-1525	Configuring charset for response is not possible	Medium	eHF Commons Camel
BAS-1523	Generation of schema element names can exceed Oracle limitation on length of these identifiers (30 characters)	Medium	eHF Generator, eHF Reference

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1507	Update ehf-multiartifact-maven-plugin maven-cobertura-plugin dependency	Medium	eHF Multiartifact Maven Plugin
BAS-1504	ehf-encryption-*-context.xml is duplicated in the assembly module	Medium	eHF Assembly
BAS-1473	Assembly Configuration contains duplicate properties	Medium	eHF Assembly
BAS-1426	Classes of src/mock/java end up in module runtime JARs	Medium	eHF Multiartifact Maven Plugin
BAS-1424	RecordWS> No error codes returned when try to create new object using non-existing/deactivated user	Medium	eHF Record Medical
BAS-1343	ContainerBasedAccessDecision does not work for documents	Medium	eHF Composition
BAS-1295	Order of elements in SOAP Responses does not fit to WSDL definition	Medium	eHF Commons
BAS-1283	Generator - ehf-service crud tag, the crud services are only accessible via the adapter level and not the service level	Medium	eHF Generator, eHF Reference
BAS-1267	Test of updating medication object is expected to break due to a validation exception but it does not	Medium	eHF Record Medical
BAS-1255	Entities are not deleted	Medium	eHF Entity
BAS-1166	Creating an EntryObject by the Reference webservice without setting a scope is answered with a misleading validation exception.	Medium	eHF Authorization, eHF Reference
BAS-1138	Hibernate: many-to-one and one-to-one associations do not support orphan delete	Medium	eHF Generator
BAS-1120	PagedResult ordered by performer is not possible with medical objects	Medium	eHF Record Medical

7.3 Known Low Priority Issues

KEY	SUMMARY	PRIORITY	COMPONENTS
BAS-1539	After restart of local testenviroment: Batch update returned unexpected row count	Low	eHF Token Service

KEY	SUMMARY	PRIORITY	COMPONENTS
	from update [0]; actual row count: 0; expected: 1		

8 Third-Party Components

The following table summarizes the changes in third-party components used for eHF development, testing and deployment. The changed version is highlighted in bold.



Note: The third party components will be updated with the final version of the release notes for eHF 2.9.

9 References

ICW Developer Network

<http://idn.icw-global.com/> ↗

ICW eHealth Framework Reference Documentation

BAS Technology and eHF Committers, InterComponentWare AG (2008-2009)

<http://idn.icw-global.com/downloads.html> ↗

ICW eHealth Framework Technical Whitepaper

BAS Technology and eHF Contributors, InterComponentWare AG (2008-2009)

<http://idn.icw-global.com/downloads.html> ↗