Bundesverwaltungsamt

IsyFact

Nutzungsvorgaben Logging

Version 1.12
26. Oktober 2016

IsyFact

Ansprechpartner:

Referat Z II 2
Bundesverwaltungsamt
E-Mail: isyfact@bva.bund.de
Internet: www.isyfact.de

Dokumentinformationen

Dokumenten-ID:	Nutzungsvorgaben_Logging.docx

Java Bibliothek / IT-System

Name	Art	Version
isy-logging	Bibliothek	siehe isyfact-bom v1.3.6

Inhaltsverzeichnis

1. Einleitung

Im Konzept *Logging* [LoggingKonzept] werden die Anforderungen und Vorgaben für das Logging der Register Factory definiert, um eine einheitliche Umsetzung in der gesamten Anwendungslandschaft zu gewährleisten.

Das vorliegende Dokument beschreibt alle Aspekte, die bei der Entwicklung einer Anwendung zu berücksichtigen sind, um die definierten Anforderungen zu erfüllen. Die Register Factory stellt eine eigene Querschnittsbibliothek `isy-logging` bereit, um die Anforderungen einfach und einheitlich umsetzen zu können. Die Nutzung der Bibliothek ist ebenfalls in diesem Dokument beschrieben. Das Dokument richtet sich daher vorrangig an Entwickler, Konstrukteure und Technische Chefdesigner von Anwendungssystemen, die gemäß den Vorgaben der Register Factory umgesetzt werden.

Abgrenzung

Die Umsetzung bzw. Konfiguration des Loggings in Systemen, die nicht nach der Register Factory entwickelt wurden (bspw. Anwendungsservern), ist nicht Teil des Dokuments. Dies wird in den jeweiligen systemspezifischen Nutzungskonzepten adressiert.

2. Aufbau und Zweck des Dokuments

Zweck des Dokuments ist die Vereinheitlichung des Loggings in der Anwendungslandschaft, um eine effiziente Auswertbarkeit der Logeinträge durch die relevanten Parteien zu ermöglichen. Dies umfasst:

- **technische Vereinheitlichung** des Loggings, also die einheitliche Konfiguration und Nutzung des Logging-Frameworks im Rahmen der Anwendungsentwicklung, sowie die

- **fachliche Vereinheitlichung** des Loggings, also Vorgaben, *wann welche* Logeinträge mit *welchen Inhalten* zu erstellen sind.

Das Dokument ist entsprechend dieser Zielsetzungen in die folgenden Kapitel untergliedert:

Im Kapitel **Grundlagen** werden grundlegende Aspekte des Loggings beschrieben, die zum Verständnis der nachfolgenden Kapitel relevant sind und die Nutzungsvorgaben geprägt haben.

Im Kapitel **Einsatz des Logging-Frameworks** wird die Konfiguration und Nutzung von `isy-logging`, dem Logging-Framework der Register Factory, beschrieben.

Im Kapitel **Vorgaben zur Logerstellung** werden alle Aspekte beschrieben, die die fachlich / inhaltliche Umsetzung des Loggings betreffen. Dies beinhaltet die zu unterstützenden Auswertungsmöglichkeiten der Logeinträge sowie die konkreten Szenarien, in denen Logeinträge zu erstellen sind.

Im Kapitel **Ereignisschlüssel isy-logging** ist eine Übersicht der verwendeten Ereignisschlüssel dargestellt.

Das Dokument ist beim erstmaligen Lesen im Ganzen zu lesen und zu berücksichtigen, später dient es als Nachschlagewerk. Während der Anwendungsentwicklung ist insbesondere das Kapitel **Vorgaben zur Logerstellung** relevant und darin im Speziellen der Abschnitt 5.2.2.8, da die dort definierten Szenarien durch die Entwickler erkannt und gemäß den Vorgaben umgesetzt werden müssen.

3. Grundlagen

In diesem Kapitel sind die Grundlagen des Loggings beschrieben.

3.1. Grundprinzipien des Loggings

Im Folgenden werden zunächst die Grundprinzipien des Loggings beschrieben, die den nachfolgenden Kapiteln zu Grunde liegen und das Logging in der Register Factory geprägt haben. Diese sollen ein einheitliches Verständnis und „Gefühl“ für das Logging vermitteln.

Logs als Ereignisdatenbank: Logdateien werden häufig als einfache Textdateien verstanden, in die Einträge – vergleichbar mit der Ausgabe von Text in der Konsole – hinzugefügt werden. Zur Umsetzung eines effizienten Loggings müssen die Logs jedoch vielmehr als Ereignisdatenbank verstanden werden, in die wichtige Ereignisse über die komplette Anwendungslandschaft hinweg aufgenommen werden, um Analysen bzgl. des Verhaltens der Systeme und von wichtigen Systemereignissen zu ermöglichen. Das Schreiben eines Logeintrags darf daher explizit nicht wie das Schreiben eines Fließtextes in eine Logdatei verstanden werden, sondern wie das Speichern eines Ereignisses in einer Datenbank. Daher gelten die Prinzipien:

1. **Logeinträge als Ereignis der Anwendung begreifen**

2. **Logging als Backend – wie eine Datenbank – begreifen**

Zielgerichtetes Logging: Logging muss zielgerichtet erfolgen, d.h. das Schreiben eines Logeintrags erfolgt zu einem bestimmten Zweck. Logeinträge, die von niemandem ausgewertet werden, sind überflüssig. Logeinträge, die nicht die relevanten Informationen enthalten, um sinnvoll ausgewertet werden zu können, sind unbrauchbar. Daher gelten die Prinzipien:

3. **Definiere den Zweck**: Für jeden Logeintrag muss klar sein, zu welchem Zweck dieser erstellt wird.

4. **Formuliere Ausgaben für einen konkreten Anwendungsfall**: Das Erstellen eines Logeintrags erfolgt für einen konkreten Anwendungsfall, in dem der Logeintrag später genutzt wird. Die Inhalte des Logeintrags müssen alle Informationen enthalten, so dass dieser Anwendungsfall effizient ausgeführt werden kann.

5. **Definiere den Konsumenten**: Für jeden Logeintrag muss klar sein, wer der Konsument des Logeintrags ist – also durch welchen Akteur der Eintrag später ausgewertet wird. Dies wird erreicht, in dem jedem Anwendungsfall ein Akteure zugeordnet ist, durch den er ausgeführt wird.

6. **Verwende nur Ausgaben, die auch ausgewertet werden**: Logeinträge sind nur dann sinnvoll, wenn sie später auch

ausgewertet werden. Logeinträge, die nicht ausgewertet werden (Selbstzweck), sind zu vermeiden.

Logging als Funktionalität der Anwendungen: Logausgaben sind Teil der Funktionalität der Anwendung und daher gilt:

7. **Logging muss qualitätsgesichert werden**: Logeinträge der Log-Level INFO, WARN, ERROR und FATAL müssen (analog zur fachlichen Funktionalität der Anwendung) einem Review unterzogen werden.

3.2. Kennzeichnung von Logeinträgen

Jeder Logeintrag wird mit einem Log-Level gekennzeichnet. Darüber hinaus werden Logeinträge bestimmter Level zusätzlich mit einer Log-Kategorie und einem Ereignisschlüssel gekennzeichnet, um die maschinelle Auswertung der Logeinträge zu erleichtern. Die zugrundeliegenden Konzepte werden in den folgenden Abschnitten erläutert. Die in Abschnitt 5.2 beschriebenen Szenarien treffen konkrete Vorgaben, wann welche Ausprägungen zu verwenden sind.

3.2.1 Log-Level

Jeder Logeintrag wird bei seiner Erstellung mit einem Log-Level gekennzeichnet. Dieses dient der Klassifizierung des Logeintrags im Hinblick auf dessen Wichtigkeit. Die Wichtigkeit ergibt sich dabei insbesondere aus dem Einfluss des geloggten Ereignisses auf den Systembetrieb. Darüber hinaus wird der Log-Level zur Steuerung der Granularität der erzeugten Logausgaben verwendet. So kann das Log-Level bspw. auf INFO gestellt werden, wodurch nur noch Logeinträge der gleichen Kategorie oder darunter (WARN, ERROR; FATAL) erzeugt werden. Die folgenden Log-Level haben sich dabei im Java-Umfeld etabliert und werden auch in der Register Factory eingesetzt:

Level		Inhalt	Beispiel(e)
1	FATAL	Schwerwiegende Fehler, von denen sich die Anwendung nicht erholen kann und beendet werden muss ("Unrecoverable Error"). Es ist ein Eingreifen des Betriebs notwendig, um die Anwendung wieder zu starten.	Die Eingabe-Datei eines Batches kann nicht gelesen werden und der Batch muss beendet werden. OutOfMemoryError.
2	ERROR	Fehler, die zum Abbruch einer Operation geführt haben und behandelt (gefangen und nicht weitergereicht) wurden. Die Anwendung läuft weiter. Der Betrieb muss die Fehlerursache analysieren und ggf.	Nachbarsystem ist endgültig nicht erreichbar. Datenbank ist nicht erreichbar. Fehlerhafter Satz wurde gelesen.

		Gegenmaßnahmen einleiten, um ein erneutes Auftreten des Fehlers zu verhindern.	
3	WARN	Beim Durchführen einer Operation ist ein Problem aufgetreten. Die Operation wurde jedoch grundsätzlich durchgeführt. Der Betrieb muss im Hinblick auf die Fehleranalyse hierbei (zunächst) nicht aktiv werden.	Aufruf eines Nachbarsystems muss wiederholt werden.
4	INFO	(Status-) Hinweise	Meldungen über den laufenden Betrieb, zum Beispiel zu verarbeiteten Anfragen, ihrem Ergebnis und ihrer Dauer, Anwendung gestartet/beendet Durchführung eines Retries (bspw. auf Grund eines OptimisticLocking-Fehlers)
5	DEBUG	Erweiterte Informationen, um Details im Programmablauf zur Fehleranalyse nachvollziehen zu können.	Meldungen zu markanten Verarbeitungsschritten (bspw. „ Führe Suche nach Personen durch“)
6	TRACE	Feingranulare und umfangreiche Informationen als Ergänzung des DEBUG-Levels für besonders schwierige Fehleranalysen.	Vollständige HTTP-Kommunikation an einer Schnittstelle (Header, Body, technische Informationen).

Tabelle 1: Log-Level

Hinweis zum Log-Level FATAL: SLF4J und damit auch logback (welches isy-logging zugrunde liegt) besitzen kein Log-Level FATAL. Die Unterscheidung, ob ein Error-Logeintrag „fatal“ ist oder nicht, kann mit Hilfe von Markern umgesetzt werden[1]. isy-logging stellt Methoden zum Schreiben von Logeinträgen im Level ERROR als auch in FATAL bereit. Bei beiden wird ein Logeintrag im Level ERROR erzeugt, dieser aber je nach verwendeter Methode mit der Kategorie „FATAL“ oder „ERROR“ versehen. Das Log-Level FATAL kann aus Sicht eines Aufrufers von isy-logging daher wie alle anderen Level verwendet werden. Bei der Auswertung ist auf diese Besonderheit jedoch zu achten.

3.2.1.1 Ausgabe des Log-Levels FATAL

Das Log-Level FATAL wird ist durch SLF4J nicht vorgesehen und wird daher auch durch logback nicht unterstützt. Aus diesem Grund werden

[1] http://slf4j.org/faq.html#fatal

Logeinträge die im Level FATAL geschrieben werden, im Level ERROR in die Logdatei geschrieben und mit einer entsprechenden Log-Kategorie (FATAL bzw. ERROR) versehen (siehe auch nachfolgender Abschnitt)

3.2.2 Log-Kategorien

Log-Kategorien dienen der Klassifizierung der Logeinträge im Hinblick auf deren Zweck. Dies erleichtert die maschinelle Auswertung der Logs und ermöglicht es, gezielt Logeinträge einer Kategorie gesondert zu behandeln (bspw. deren Ausgabe umzuleiten oder zu verwerfen). Die Log-Kategorien stellen dabei Verfeinerungen der Log-Level dar. Log-Kategorien werden ausschließlich im Level INFO verwendet und werden in folgeender Tabelle beschrieben:

Level	Kategorie	Beschreibung	Beispiel(e)
ERROR	FATAL	Logeinträge des Log-Levels FATAL (vgl. Abschnitt 3.2.1.1)	*siehe Log-Level FATAL*
	ERROR	Logeinträge des Log-Levels ERROR (vgl. Abschnitt 3.2.1.1)	*siehe Log-Level ERROR*
INFO	JOURNAL	Informationen zu Systemzustand, Systemereignissen und durchgeführten Operationen.	Herunterfahren des Systems, Änderung der Konfiguration
	PROFILING	Informationen zum Laufzeitverhalten des Systems.	Dauer der Verarbeitung eines Nachbarsystem-aufrufs
	METRIK	Kennzahlen zum Systembetrieb und zur Systemnutzung.	Erfolgreiche/Fehlerhafte Nutzung einer Service-Methode
	SICHERHEIT	(Potentieller) Angriffsversuch.	Benutzer-Account wird gesperrt wegen zu vieler ungültiger Anmeldeversuche

Tabelle 2: Unterscheidung der Log-Kategorien

3.2.3 Ereignisschlüssel

Ereignisschlüssel dienen der *eindeutigen* Identifikation des Zwecks, auf Grund dessen der Logeintrag im Log-Level INFO erstellt wurde (bspw. Erstellung eines Logeintrags beim Verlassen einer Systemgrenze zur Performancemessung). Dies ist notwendig, da das Log-Level INFO eine Vielzahl unterschiedlicher Auswertungsmöglichkeiten bietet. Ohne die Verwendung des Schlüssels könnte der Zweck des jeweiligen Eintrags meist nur mit Kenntnis des Quellcodes oder Interpretation der Lognachricht ermittelt werden, was eine maschinelle Auswertung der Einträge erschwert oder gar unmöglich macht.

Wenn an mehreren Stellen Logeinträge für den gleichen Zweck erstellt werden, wird hierfür der gleiche Ereignisschlüssel verwendet. Dies ist bspw. im Logszenario „Loggen fachlicher Operationen“ (siehe Abschnitt 5.2.3.6) der Fall, in dem die Durchführung fachlicher Operationen jeweils mit dem gleichen Ereignisschlüssel geloggt werden, so dass alle diese Einträge mit einer einzelnen Abfrage auf den definierten Schlüssel ausgewertet werden können.

In den Log-Leveln FATAL, ERROR und WARN wird der jeweilige Fehlerschlüssel als Ereignisschlüssel verwendet. In den Log-Leveln DEBUG und TRACE werden keine Ereignisschlüssel Verwendet, da dort der Zweck bereits eindeutig durch den Log-Level (Zweck „Fehleranalyse“) bestimmt ist. Dadurch kann der Aufwand für die Verwendung der Ereignisschlüssel gering gehalten werden.

In Kapitel 6 wird eine Reihe von Standardschlüsseln definiert, die durch das Logging-Framework verwendet werden. Darüber hinaus können in jeder Anwendung eigene Ereignisschlüssel, für systemspezifische Zwecke, definiert werden.

Der Aufbau der Ereignisschlüssel entspricht dem folgenden Schema:

```
E[A-Z]{5}[0-9]{2}[0-9]{3}
```

Dieses setzt sich aus den folgenden Elementen zusammen:

- Jedem Schlüssel wird die feste Zeichenkette „**E**“ vorangestellt, um den Eintrag als Ereignisschlüssel zu Kennzeichnen und zu verhindern, dass dieser mit Fehler- oder Hinweisschlüsseln verwechselt wird.

- **5 Buchstaben**, zur Identifikation des Systems. Diese ist analog zur Identifikation, die auch bei den Ausnahme-IDs (siehe [FehlerbehandlungKonzept]) verwendet wird.

  Anmerkung: Die Systemidentifikation ist Teil des Ereignisschlüssels, um sicherzustellen, dass systemspezifische / bibliotheksspezifische Schlüssel nicht in mehreren Komponenten redundant vergeben werden.

- **2 Ziffern**, zur eindeutigen Identifikation der Komponente, in der der Logeintrag erstellt wird (Komponenten-ID). Bei der Erstellung einer neuen Anwendung ist in der Spezifikations- bzw. Konstruktionsphase festzulegen, welche Komponente welche ID zugeordnet wird. Dies ist ebenfalls analog zur Definition der Ausnahme-IDs – es wird für Ausnahme-IDs und Ereignisschlüssel die gleiche Komponenten-ID verwendet.

- **3 Ziffern**, als laufende Nummer der Ereignisschlüssel der jeweiligen Komponente.

Ein exemplarischer Ereignisschlüssel der Fachanwendung XYZ ist demnach:

```
EXYZFA12123
```

Der Aufbau des Ereignisschlüssels besitzt darüber hinaus explizit keine weitere Semantik, um Redundanzen mit den weiteren Attributen des Logeintrags zu vermeiden.

3.3. Vorgaben für Logdateien

Jede Applikation schreibt *eine* einzelne Logdatei. Weitere Logdateien sind nicht erlaubt. Ausnahmen bilden Logdateien, die durch den Container geschrieben werden, z.B. ein Wrapper- oder Access-Log eines Tomcat-Servers.

3.3.1 Namenskonventionen

Die Logdateien haben fest vorgegebene Namen, die dem folgenden Namensschema entsprechen:

```
<HOST>_<SYSTEM-ID>_<ZEITSTEMPEL>.log
```

Die einzelnen Platzhalter im Namensschema sind in folgender Tabelle beschrieben:

Bestandteil	Werte	Beschreibung
HOST	z.B. xyzweb01	Name des Servers, auf dem die Logs entstehen
SYSTEM-ID	*Durch den Technischen Chefdesigner für die jeweilige Anwendung in Abstimmung mit dem Auftraggeber festzulegen siehe* [Namenskonventionen].	Name der Anwendung[2] bzw. des Batches
ZEITSTEMPEL	YYYY-MM-DD_HH00	Datum der Logdateien inkl. stundengenauer Uhrzeit. Zu beachten ist, dass der Zeitstempel erst beim Rotieren der Logs an die Datei angehängt wird (siehe Abschnitt 3.3.3).
LAUFENDE-NUMMER	1, 2, 3 …	Bei Batches wird ein größenbasiertes Rollieren der Logdateien durchgeführt (siehe Abschnitt 3.3.3). Diese

[2] „Anwendung“ bezieht sich hierbei auf die Anwendungen, die im Tomcat laufen, in Abgrenzung zu Batches.

		werden dabei mit einer laufenden Nummer versehen.

Tabelle 3: Bestandteile eines Logdateinamens

Ein Beispiel des Namens einer Logdatei ist demnach:

`xyzweb01_xyzregister_2007-09-16_0900.log`
(Log vom 16. September 2007 ~9 Uhr,
von der Anwendung XYZ-Register auf dem Server xyzweb01)

`xyzweb01_batchxyz_1.log`
(Rollierte Logdatei des Batches XYZ-Batch auf dem Server xyzweb01)

3.3.2 Speicherort

Um die Logdateien durch die Infrastruktur möglichst einfach weiterverarbeiten zu können, werden Logdateien in einem definierten Logverzeichnis je Host abgelegt, welches Unterverzeichnisse für jede Anwendung besitzt. Diese Verzeichnishierarchie ist für alle Anwendungen und Umgebungen gleich, um den Pflegeaufwand für diese Aufgabe so gering wie möglich zu halten.

Logdateien müssen entsprechend dem folgenden Schema abgelegt werden:

`/var/log/<SYSTEM-ID>/<LOGDATEI>`

Bei der Einführung einer neuen Anwendung ist die System-ID (Anwendungsname/Batch-ID) entsprechend abzustimmen und der Betrieb darüber in Kenntnis zu setzen.

3.3.3 Log-Rotation und Komprimierung

Um zu verhindern, dass Logdateien zu groß werden und es gleichzeitig zu ermöglichen, die Logdateien nur für bestimmte Fristen vorzuhalten, werden die anfallenden Logeinträge stündlich in neue Dateien geschrieben (rollierendes Logging).

Zu beachten ist, dass für den Zeitstempel der rotierten Logdateien die Zeitzone UTC verwendet wird – analog zum Zeitstempel der einzelnen Logeinträge. Dieser kann von der Systemzeit des Systems abweichen.

Sonderfall: Batches

Für Batches wird eine größenbasierte Rotation (100MB) durch das Logging-Framework umgesetzt. Ein Batch erzeugt demnach pro Lauf potentiell mehrere Logdateien unabhängig von dessen Dauer.

Anmerkung: Es findet keine Komprimierung der Logdateien durch das Logging-Framework statt. Hintergrund ist, dass die Komprimierung bei Batches nicht eingesetzt werden kann und ein einheitlicher Mechanismus für alle Logdateien umgesetzt werden soll. Bei den Batches ist eine Komprimierung nicht möglich, da die Komprimierung durch das Logging-

Framework erst beim Rollieren der Logdatei erfolgt. Beim Ausführen des Batches wird die zuletzt angefangene Logdatei jedoch nicht mehr rolliert und damit auch nicht komprimiert.

4. Einsatz des Logging-Frameworks

In diesem Abschnitt wird der Einsatz des Logging-Frameworks `isy-logging` beschrieben.

4.1. Aufruf des Frameworks

Im Folgenden wird zunächst der Aufruf von `isy-logging` in einer Anwendung beschrieben.

4.1.1 Anlegen einer Logger-Instanz in der Anwendung

Jede Klasse, in der Logs geschrieben werden, muss eine eigene Logger-Instanz verwenden. Es ist nicht vorgesehen Logger zu vererben. Die Erzeugung der Logger-Instanz erfolgt mit der Logger-Factory, die durch `isy-logging` bereitgestellt wird.

```
public class MyClass {
	private static final IsyLogger LOG =
IsyLoggerFactory.getLogger(MyClass.class);
...
```

Der Name des Loggers muss dem Namen der Klasse entsprechen, in welcher der Logger instanziiert wird – dazu wird die Klasse beim Aufruf der Factory übergeben. Dies ist notwendig, um Logeinträge ihrer Quelle zuordnen zu können.

4.1.2 Schnittstelle des Loggers

Die Schnittstelle `IsyLogger` wird zum Erstellen der Logeinträge verwendet.

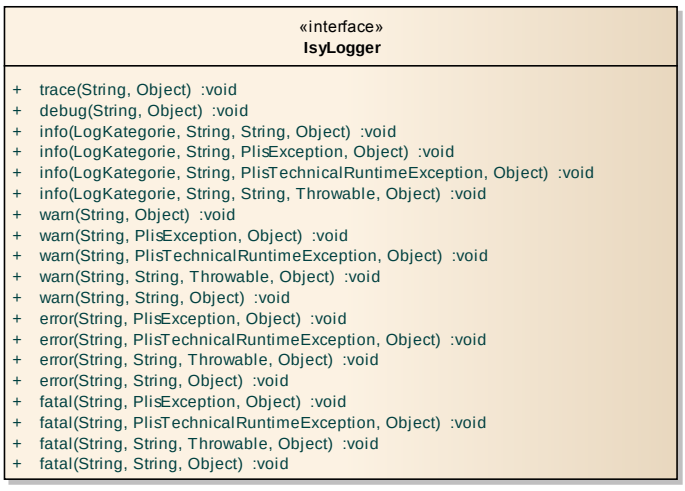

Abbildung 1: IsyLogger

Der Logger stellt eine Vielzahl von Methoden bereit, der Aufruf unterscheidet sich aber kaum von den üblichen Methoden anderer Frameworks – die Vielzahl der Methoden ergibt sich primär durch die Bereitstellung unterschiedlich typisierter Methoden zum Loggen von Exceptions

(`PlisExcetion`, `PlisTechnicalRuntimeException` und `Throwable`).

Anmerkungen:

- Zu jeder oben dargestellten Methode existiert zusätzlich eine Methode gleicher Signatur mit dem Suffix „Fachdaten“ (bspw. `debugFachdaten`) deren Verwendung in Nachfolgender Tabelle verdeutlicht wird.

- Die Schnittstelle bietet nicht alle bzw. andere Methoden an, als bspw. SLF4J oder log4j. Dies ist beabsichtigt, um die Log-Inhalte in Systemen, die gemäß der Register Factory entwickelt werden, besser standardisieren zu können. Drittsoftware (bspw. Frameworks wie Hibernate, Spring etc.) oder Systeme, die schrittweise auf das Logging-Framework migriert werden, nutzen automatisch die Logger-Schnittstelle, die durch logback bereitgestellt wird (zum Umgang mit Drittsoftware siehe Abschnitt 4.2.3, zur Migration siehe Abschnitt 9.1).

- Die Logeinträge werden beim Schreiben einheitlich mit einem Zeitstempel der Zeitzone „UTC“ versehen. Hierauf kann beim Aufruf des Loggers keinen Einfluss genommen werden.

Die Methoden der Schnittstelle werden in folgender Tabelle erläutert:

Interface: IsyLogger

Die folgenden Methoden dienen der *einfachen* Ausgabe von Lognachrichten:

- `warn(String nachricht, Object… werte)`
- `debug(String nachricht, Object… werte)`
- `trace(String nachricht, Object… werte)`

Der Aufruf wird an die entsprechende Methode des SLF4J-Loggers (mit gleicher Signatur) delegiert.

Dabei werden alle `werte` – also die Inhalte für Platzhalter in der Nachricht – zusätzlich als Marker übergeben, so dass sie im Logeintrag als separate Attribute ausgegeben werden können und damit einfacher auswertbar sind:

- `parameter1`: werte[1].
- `parameter2`: werte[2].
- etc.

Beispiel: Der Aufruf

```
debug("Die Methode {} wurde mit dem Parameter {}
          aufgerufen.", "addiere", "5")
```

ergänzt die folgenden Attribute im Logeintrag:

- `parameter1`: addiere
- `parameter2`: 5

In den Log-Leveln FATAL, ERROR, WARN und INFO existieren je drei Methoden zum Loggen von Exceptions:

- `<fatal/error/warn/info>(String nachricht, PlisException exception, Object… werte)`
- `<fatal/error/warn/info>(String nachricht, PlisTechnicalRuntimeException exception, Object… werte)`
- `<fatal/error/warn/info>(String schluessel, String nachricht, Throwable exception, Object… werte)`

Beim Loggen einer “PLIS-Exception” wird der Fehlerschlüssel automatisch als Ereignisschlüssel übernommen, bei allen anderen Exceptions muss zusätzlich ein Ereignisschlüssel übergeben werden.

Der Aufruf wird an die entsprechende Methode des SLF4J-Loggers delegiert. Als Marker werden dabei übergeben:

- `fehlerschluessel`: Mit dem Fehlerschlüssel der `PlisException`.
- `parameter[1..n]`: *wie oben*

Interface: IsyLogger
Zum Erstellen von INFO-Logeinträgen wird die folgende Methode bereitgestellt: • `info(LogKategorie kategorie, String schluessel, String nachricht, Object… werte)` Der Aufruf wird an die Methode `info` des SLF4J-Loggers delegiert. Als Marker werden dabei übergeben: • `kategorie`: Mit dem Wert des übergebenen Parameters • `schluessel`: Mit dem Wert des übergebenen Parameters • `parameter[1..n]`: *wie oben*
Zu allen Methoden, die bisher beschrieben wurden, wird eine äquivalente Methode mit gleicher Signatur angeboten, die zum Loggen von datenschutzrelevanten fachlichen Daten im jeweiligen Log-Level verwendet wird (vgl. Abschnitt 4.1.2.2). Die Methoden tragen dabei jeweils das Suffix „Fachdaten“ im Namen. Bspw.: • `debugFachdaten(…)`, `infoFachdaten(…)` etc.

4.1.2.1 Verwenden von Platzhaltern

In den Lognachrichten können Platzhalter verwendet werden, die beim Erstellen des Logeintrags mit den konkreten Werten des aktuellen Aufrufs ersetzt werden (bspw. gemessene Laufzeit).

Platzhalter sind in den Nachrichten durch geschweifte Klammern „{}“ zu kennzeichnen, bspw.:

```
RICHTIG:
  LOG.debug("Die Methode {} wurde mit dem Parameter {}
aufgerufen.", method.getName(), wert)
```

Die Verwendung des Parameters `werte` zum Ersetzen der Platzhalter ermöglicht es zudem, die übergebenen Parameter als separate Attribute in den Logeintrag zu übernehmen (siehe vorhergehender Abschnitt), was die Auswertbarkeit der Einträge erleichtert. Zudem wird die Performance des Systems leicht erhöht, da die Konkatenation des Strings nur dann erfolgt, wenn der Logeintrag auch geschrieben wird (d.h. der Log-Level eingeschaltet ist) – dieser Performance-Gewinn ist jedoch vernachlässigbar und nicht die eigentliche Motivation dieser Vorgehensweise.

Das direkte Konkatenieren von Zeichenketten zum Aufbau einer Lognachricht ist nicht erlaubt:

```
FALSCH:
    LOG.debug("Die Methode " + method.getName() + " wurde mit dem Parameter " +
wert + " aufgerufen.")
```

Das früher weitverbreitete „`isDebugEnabled`“ ist im Normalfall nicht mehr notwendig (da die Konkatenation durch logback nur stattfindet, wenn der

Logeintrag auch geschrieben wird) und sollte daher auch nicht mehr verwendet werden, um den Code übersichtlich zu halten:

```
FALSCH:
    if (LOG.isDebugEnabled()) {
       LOG.debug("Die Methode {} wurde mit dem Parameter {} aufgerufen.",
                   method.getName(), wert).
    }
```

Ausnahme ist hierbei jedoch das Loggen komplexer Meldungen:

```
RICHTIG:
    if (LOG.isDebugEnabled()) {
       LOG.debug("Debug-Meldung: {} ", myObject.complexMethod());
    }
```

In diesem Code-Beispiel wird sehr viel Rechenzeit verbraucht, um die Log-Information von der Methode `myObject.complexMethod()` zu bekommen. Um den komplexen Aufruf nur durchzuführen, wenn der Logeintrag auch wirklich geschrieben wird, ist es in diesem Fall sinnvoll die Prüfung `isDebugEnabled` durchzuführen.

4.1.2.2 Loggen datenschutzrelevanter Daten

Lognachrichten dürfen im Allgemeinen keine datenschutzrelevanten (fachlichen) Daten enthalten. Unter Umständen kann es jedoch notwendig sein, entsprechende Daten in den Logeintrag mitaufzunehmen, um die in Abschnitt 5.1 definierten Auswertungen zu ermöglichen. Bspw. wenn eine Fehleranalyse nur mit Kenntnis der an der Schnittstelle übertragenen Daten möglich ist.

Diese Logeinträge müssen bei ihrer Erstellung markiert werden, so dass sie durch die Log-Infrastruktur speziell geschützt werden können. Fachliche Daten dürfen daher ausschließlich mit Hilfe der Methoden `<trace/debug/info/warn/error/fatal>Fachdaten(…)` geloggt werden.

4.1.3 Hilfsklassen

`isy-logging` stellt die folgenden Hilfsklassen zum Erstellen von Logeinträgen bereit.

4.1.3.1 LoggingMethodInterceptor und LoggingMethodInvoker

Die Klassen `LoggingMethodInterceptor` und `LoggingMethodInvoker` bieten die Möglichkeit, einheitliche Logeinträge vor und nach dem Aufruf einer Methode für verschiedene Zwecke (insbesondere dem Messen der Laufzeit für das Profiling) zu erstellen. Beide erzeugen die gleichen Logeinträge, dienen jedoch unterschiedlichen Einsatzzwecken.

Der Interceptor wird per Spring als Method-Interceptor konfiguriert und kann dadurch querschnittlich für eingehende Methodenaufrufe konfiguriert werden – dies wird in Abschnitt 4.2.2.1 beschrieben.

Der Invoker wird direkt im Anwendungscode für die Durchführung von Methodenaufrufen verwendet. Zur Verwendung des Invokers, muss eine Instanz des Interceptors als Klassenvariable erstellt werden:

```
public class MyClass {
      private static final LoggingMethodInterceptor LOG_INTERCEPTOR =
                                    new LoggingMethodInterceptor(true, true, false,
false);
...
```

Der Konstruktor besitzt folgende Signatur:

- `LoggingMethodInvoker(Method methode, IsyLogger logger, boolean loggeAufruf, boolean loggeErgebnis, boolean loggeDauer, boolean loggeDaten, boolean loggeDatenBeiException, long loggeMaximaleParameterGroesse)`: Methode ist die aufzurufende Methode. Die Flags werden verwendet, um zu steuern, welche Logeinträge erstellt werden (siehe unten).

Das Loggen eines Methodenaufrufs erfolgt mit der Methode:

- `fuehreMethodeAus(Object zielobjekt, Object... parameter)`: Ruft per Reflection die Methode, welche per Konstruktor gesetzte wurde, auf dem Zielobjekt mit den übergebenen Parametern auf und schreibt die Logeinträge mit folgenden Ereignisschlüsseln (Details zu den Inhalten der jeweiligen Logeinträge finden sich in Kapitel 6):

Ereignisschlüssel LoggingMethodInvoker
Falls loggeAufruf = true
EISYLO01001
Falls loggeErgebnis = true und keine Exception geliefert wurde
EISYLO01002
Falls loggeErgebnis = true und eine Exception geliefert wurde
EISYLO01003
Falls loggeDauer = true und keine Exception geliefert wurde
EISYLO01004
Falls loggeDauer = true und eine Exception geliefert wurde
EISYLO01005

Darüber hinaus werden folgende Debug-Logeinträge erstellt:

Schlüssel	Level	Kategorie	Text
Falls loggeDatenBeiException = true			
	DEBUG		Die <Klasse. Methode> wurde mit folgenden Parametern aufgerufen <Parameter>. ANMERKUNG: Der Logeintrag wird als „Fachdaten“ gekennzeichnet.
Falls loggeDaten = true und eine Exception geliefert wurde			
	DEBUG		Die <Klasse. Methode> wurde mit folgenden Parametern aufgerufen <Parameter>. ANMERKUNG: Der Logeintrag wird als „Fachdaten“ gekennzeichnet.
Falls Debug-Einträge erstellt werden und ein Parameter zu groß ist			
	DEBUG		Die <Klasse. Methode> wurde mit einem zu großen Parameter aufgerufen. Position: <Position des Parameters>, Klasse: <Klasse des Parameters> ANMERKUNG: Der Logeintrag wird als „Fachdaten“ gekennzeichnet. Außerdem werden zu große Parameter in den oben genannten Logeinträgen durch „<Maximale Größe überschritten>“ ersetzt.

Den Aufrufen von Nachbarsystemen kommt eine besondere Wichtigkeit bei der Analyse des Laufzeitverhaltens von Systemen zu. Daher stellt der Invoker für Methodenaufrufe von Nachbarsystemen einen eigenen Konstruktor bereit:

- `LoggingMethodInvoker(Method methode, IsyLogger logger, boolean loggeAufruf, boolean loggeErgebnis, boolean loggeDauer, boolean loggeDaten, boolean loggeDatenBeiException, long loggeMaximaleParameterGroesse, String nachbarsystemName, String nachbarsystemUrl)`: Analog zu oben, nur das der Name und die URL des aufgerufenen Nachbarsystems übergeben wird.

Dieser Konstruktor ist beim Aufruf einer Serviceschnittstelle eines Nachbarsystems zu verwenden (vgl. auch Szenario „Performance

überwachen“ in Abschnitt 5.1.5). Zu beachten ist, dass die Klasse `IsyHttpInvokerClientInterceptor`, welche durch den Baustein Service bereitgestellt wird (siehe [ServiceDetailkonzept]), bereits einen entsprechenden Aufruf des Invokers durchführt. Bei Verwendung dieses Konstruktors werden die Logeinträge mit folgenden Ereignisschlüsseln erstellt:

Ereignisschlüssel LoggingMethodInvoker (Nachbarsystemaufruf)
Falls loggeAufruf = true
EISYLO01011
Falls loggeErgebnis = true und keine Exception geliefert wurde
EISYLO01012
Falls loggeErgebnis = true und eine Exception geliefert wurde
EISYLO01013
Falls loggeDauer = true und keine Exception geliefert wurde
EISYLO01014
Falls loggeDauer = true und eine Exception geliefert wurde
EISYLO01015

4.1.3.2 LogApplicationListener

Die Hilfsklasse `LogApplicationListener` dient dem Loggen von Änderungen des Systemzustands. Sie muss gemäß Abschnitt 4.2.2.3 als Spring-Bean konfiguriert, aber danach nicht mehr explizit aufgerufen werden. Die Klasse erstellt die Logeinträge mit folgenden Ereignisschlüsseln (Details zu den Inhalten der jeweiligen Logeinträge finden sich in Kapitel 6):

Ereignisschlüssel LogApplicationListener
Beim Starten einer Anwendung / eines Batches
EISYLO02001, EISYLO02003 und je ein Eintrag mit Schlüssel EISYLO02004 für die folgenden Parameter: Java-Version, Zeitzone, Heap-Size, File-Encoding
Beim Stoppen einer Anwendung / eines Batches
EISYLO02002

4.1.3.3 MdcHelper

Die Klasse MdcHelper erleichtert das Setzen von Informationen im MDC (Mapped Diagnostic Context).

Es werden Methoden zum Setzen und Lesen der Korrelations-ID bereitgestellt:

- pushKorrelationsId(…): Zum „pushen“ einer neuen Korrelations-ID in den MDC. Dies bedeutet: Wenn die Korrelations-ID „X“ gesetzt wird, wird diese im Attribut „korrelationsid“ im MDC gesetzt. Sollte

dieses Attribut bereits gesetzt sein (bspw. mit der Korrelations-ID „Y“), so wird das Attribut durch „Y;X“ ersetzt.

- liesKorrelaiionsId(): Liest die Korrelations-ID aus dem MDC.
- entferneKorrelationsId(): Entfernt die zuletzt „gepushte“ Korrelations-ID (bspw. „Y;X“ wird zu „Y“).
- entferneKorrelationsIds(): Entfernt alle Korrelations-IDs.

Darüber hinaus werden Methoden zum Kennzeichen der Inhalte im MDC als fachlich bereitgestellt (vgl. Abschnitt 4.1.2.2):

- setzeMarkerFachdaten(…): Markiert den MDC als fachlich / nicht fachlich.
- liesMarkerFachdaten(): Gibt an, ob der MDC fachliche Daten enthält.
- entferneMarkerFachdaten(): Entfernt den Marker für Fachdaten.

4.1.4 Diagnosekontext / Korrelations-ID

Die Korrelations-ID (siehe [ServicekommunikationKonzept]) ist in jedem Eintrag mitzuloggen, damit die Logeinträge einzelnen Aufrufen zugeordnet und über die Komponenten der Anwendungslandschaft verfolgt werden können. Das Ermitteln der Korrelations-ID erfolgt automatisch durch `isy-logging`. Hierzu wird der Mapped Diagnostic Context (MDC) verwendet, der durch SLF4J bzw. logback zur Verfügung gestellt wird. Der MDC wird über eine statische Methode gesetzt, und zwar pro Thread:

```
MDC.put("Korrelations-ID", "<Korrelations-ID>");
```

Die Korrelations-ID kann sich aus mehreren Unique-IDs zusammensetzen, durch die der Aufruf durch die Anwendungslandschaft nachverfolgt werden kann. Die IDs müssen hintereinander gehängt, getrennt durch ein Semikolon, im Kontext gesetzt werden, bspw.:

```
MDC.put("Korrelations-ID", "c15638a2-4c38-4d18-b887-5ebd2a1c427d;f60143b3-3408-
4501-9947-240ec1c48667;c893d44f-3b8e-446e-a360-06a520440e64");
```

Am Ende der Verarbeitung ist der MDC wieder zu entfernen:

```
MDC.remove("Korrelations-ID");
```

Anmerkung zu Multi-Threading

Es wird davon ausgegangen, dass es innerhalb eines Request kein Multi-Threading gibt, sondern nur in den Clients. Da der Client einem bestimmten Benutzer zugeordnet werden kann, wird hier kein MDC benötigt.

Sollte jedoch Multi-Threading innerhalb eines Requests vorhanden sein, so ist der MDC dem Thread mitzugeben. Somit müssen alle Klassen, die das Interface Runnable implementieren, eine Methode vorsehen, um den MDC von der Klasse zu bekommen, die den Thread startet. Ansonsten besitzt der gestartete Thread nicht den Kontext des aufrufenden Threads. Zusätzlich muss im Thread eine weitere Unique-ID an die Korrelations-ID im MDC angehängt werden, so dass auch die Logeinträge des Threads eindeutig identifiziert werden können.

4.2. Konfiguration

In diesem Abschnitt werden die notwendigen Konfigurationen zum Einrichten des Loggings beschrieben. Die Konfiguration erfolgt dabei ausschließlich über die Konfigurationsdatei von logback und Spring – `isy-logging` besitzt selbst keine zusätzliche Konfigurationsdatei.

4.2.1 Logback-Konfiguration

Folgende Aspekte sind bei der Logback-Konfiguration zu beachten:

Konfigurationsdateien

Alle Anwendungen dürfen ihre Logging-Konfiguration ausschließlich über die Konfigurationsdatei `logback.xml` vornehmen. Die Auslieferung der Logging-Konfiguration geschieht mit den applikationsspezifischen Konfigurationsdateien für die jeweilige Umgebung. Die Ablage der Konfigurationsdatei in Sourcen und Kompilaten ist durch das Konzept [ÜberwachungKonfigKonzept] definiert. Die Konfigurationsdateien dürfen nicht in einem Archiv (JAR-Bibliothek) abgelegt werden, sondern müssen als einzelne Dateien installiert werden.

Log-Level und Anpassung der Konfiguration zur Laufzeit

Logback wird so konfiguriert, dass die Konfigurationsdatei jede Minute automatisch neu geladen wird und durchgeführte Änderungen – ohne Neustart der Anwendung – übernommen werden. In Testumgebungen kann hierdurch zwischen verschiedenen Konfigurationsalternativen gewechselt werden. In Produktion ist die Konfiguration in aller Regel fix, da sie auf die betriebliche Infrastruktur abgestimmt sein muss. In der Produktionsumgebung darf daher nur der Log-Level angepasst werden.

Standardmäßig werden die Systeme in Produktion im Log-Level INFO betrieben. Bei Bedarf kann jedoch auf DEBUG und in Ausnahmefällen auf TRACE gewechselt werden, um detaillierte Informationen zur Fehleranalyse bereitzustellen. Andere Log-Level sind zu vermeiden.

4.2.1.1 Anwendungen (zeitbasiertes Rollieren)

Die Bilbiothek isy-logging stellt bereits einen vorkonfigurierten Appender bereit, durch den Logdateien gemäß den in Abschnitt 3.3 definierten Vorgaben erstellt werden. In der Anwendung selbst ist daher nur noch eine minimale Logging-Konfiguration notwendig:

```
<configuration scan="true" scanPeriod="1 minutes">
```

```
    <!-- Eindeutige Identifikation der Instanz der Anwendung. -->
    <contextName>testserver_testsystem</contextName>

    <!-- Pfad der Logdatei, ohne Endung -->
    <property name="LOGFILE_PATH"
value="logausgaben/testserver_testsystem" />

    <!-- MDC in die Ausgabe mitaufnehmen. -->
    <property name="INCLUDE_MDC" value="false" />

    <!-- Include der vorkonfigurierten Appender. -->
    <include resource="resources/isylogging/logback/appender.xml" />

    <!-- Root-Logger als Grundlage für alle Logger-Instanzen -->
    <root level="trace">
        <appender-ref ref="DATEI_ANWENDUNG" />
    </root>

</configuration>
```

Folgende Parameter sind zu setzen:

- LOGFILE_PATH: Der Pfad der Logdatei (LOGFILE_PATH) muss gemäß den Vorgaben aus Abschnitt 3.3 angepasst werden.
- INCLUDE_MDC: Gibt an, ob der komplette Inhalt des MDC in das Log aufgenommen werden soll (true) oder nicht (false).
- CONTEXT_NAME: Als `contextName` wird „<HOST>_<SYSTEM-ID>“ zur eindeutigen Identifikation der Instanz der Anwendung angegeben.

4.2.1.2 Batches (größenbasiertes Rollieren)

Die Konfiguration des Loggings für Batches erfolgt analog zum vorhergehenden Abschnitt. Es wird jedoch ein anderer Appender referenziert der ein größenbasiertes Rollieren umsetzt:

```
<configuration scan="true" scanPeriod="1 minutes">

    <!-- Pfad der Logdatei, ohne Endung -->
    <property name="LOGFILE_PATH"
value="logausgaben/testserver_testsystem" />

    <!-- MDC in die Ausgabe mitaufnehmen. -->
    <property name="INCLUDE_MDC" value="false" />

    <!-- Maximale Fenstergröße der zu erstellenden Logdateien. -->
    <property name="MAX_INDEX" value="20" />


    <!-- Include der vorkonfigurierten Appender. -->
    <include resource="resources/isylogging/logback/appender.xml" />

    <!-- Root-Logger als Grundlage für alle Logger-Instanzen -->
    <root level="trace">
        <appender-ref ref="DATEI_BATCH" />
    </root>

</configuration>
```

Der Parameter MAX_INDEX muss Standardmäßig auf „20“ gesetzt werden. Dieser wird im Folgenden erläutert:

Logback stellt die beiden Policies `TimeBasedRollingPolicy` und `FixedWindowRollingPolicy` zur Verfügung. Erstere wird im referenzierten Appender DATEI_ANWENDUNG verwendet, letztere im Appender DATEI_BATCH, um ein rein größenbasiertes Rollieren durchzuführen. „`FixedWindow`“ bedeutet hierbei, dass immer nur eine maximale Anzahl an Logdateien erhalten bleiben (gemäß der „Window-Größe“). Die Logdateien werden dabei mit einer laufenden Nummer versehen („Logdatei1“, „Logdatei2“ etc.). Beim Rollieren, wird die Nummer der vorhandenen Logdateien erhöht („Logdatei1“ wird „Logdatei2“ etc.) und die aktuelle Logdatei zu „Logdatei1“ umbenannt. Bei einer Fenstergröße von 20 wird dabei die vorher ggf. vorhandene „Logdatei20“ gelöscht. Da die Logdateien zwischen jedem Batchlauf archiviert werden, ist eine Angabe der Fenstergröße aus Sicht dieses Konzepts nicht notwendig – dies ist jedoch ein Pflichtparameter der logback-Konfiguration. Standardmäßig muss der Wert auf 20 gesetzt werden. Dieser Wert wurde gewählt, da es der Standardwert ist, und ein Batch in der Regel nicht mehr als 2GB Logs erstellt (sollte dies doch der Fall sein, muss die Fenstergröße entsprechend erhöht werden).

4.2.1.3 Lokale Entwicklungsumgebung (Konsolenausgabe)

In der lokalen Entwicklungsumgebung ist es hilfreich, die Logausgaben direkt auf der Konsole in einem einfachlesbaren Format auszugeben. Hierfür wird folgende Konfiguration verwendet:

```
<configuration scan="false">

    <!-- Include der vorkonfigurierten Appender. -->
    <include resource="resources/isylogging/logback/appender-
entwicklung.xml" />

    <!-- Root-Logger als Grundlage für alle Logger-Instanzen -->
    <root level="trace">
        <appender-ref ref="KONSOLE" />
    </root>

</configuration>
```

4.2.1.4 Weitere Konfigurationsmöglichkeiten

In diesem Abschnitt werden weitere Möglichkeiten der Konfiguration von logback beschrieben, die bei Bedarf genutzt werden können:

4.2.1.4.1 Logging für einzelne Klassen deaktivieren

Es kann sinnvoll sein, das Log-Level einer einzelnen Klasse oder eines Packages abweichend zum Root-Logger zu konfigurieren – bspw. falls ein Framework in einer bestimmten Klasse irreführende Logeinträge erzeugt. Dies geschieht nach folgendem Schema:

```
  <logger name="<Package- oder Klassenname>" level="<Log-Level>"
      additivity="false">
    <appender-ref ref="FILE" />
  </logger>

  <!-- Root-Logger als Grundlage für alle Logger-Instanzen -->
  <root level="debug">
    ...
```

Das Attribut „`additivity=false`“ gibt dabei an, dass für die konfigurierte Klasse bzw. das konfigurierte Package ausschließlich dieser Logger und nicht zusätzlich der Root-Logger verwendet werden soll.

4.2.2 Spring-Konfiguration

Im Folgenden werden die Spring-Konfigurationen zur Integration von logback in Spring und zur Konfiguration der genutzten Hilfsmechanismen (vgl Abschnitt 4.1.2.2) beschrieben.

4.2.2.1 LogbackConfigListener

Zum Initialisieren und sauberen Herunterfahren von logback in Web-Anwendungen, wird der `LogbackConfigListener` verwendet, der durch die Bibliothek `org.logback-extensions:logback-ext-spring` bereitgestellt wird. Hierzu ist eine entsprechende Konfiguration in der web.xml vorzunehmen:

```
<web-app>
...
<!--
        Angabe des Speicherorts der logback Konfiguration
        Wenn nicht angegeben, greift die Standardinitialisierung:
        Konfiguration im Classpath.
        Aufgrund des Deplyoments liegt die Konfiguration aber unter
        /classes/config/logback.xml
-->
<context-param>
        <param-name>logbackConfigLocation</param-name>
        <param-value>classpath:/config/logback.xml</param-value>
</context-param>
<!--
        Angabe des zu verwendenden Listeners fuer logback

        Dies ist noetig, da Container, die die Servlet 2.4 API implementieren
        verlangen, dass Listener vor load-on-startup Servlets geladen werden.
        Servlet 2.3 Container erzwingen dieses Vorgehen.
        Ausserdem ist der LogbackConfigListener vor dem ContextLoaderListener zu
        registrieren, siehe analoge Vorgaben für log4j unter
        http://static.springframework.org/spring/docs/2.0.x/
        api/org/springframework/web/util/Log4jConfigListener.html
        Anmerkung: Sollte ein aelterer Servlet-Container (2.2) verwendet werden ist
        LogbackConfigServlet anstelle von LogbackConfigListener zu verwenden.
 -->
<listener>
        <listener-class>ch.qos.logback.ext.spring.web.LogbackConfigListener</listener-class>
</listener>
<!--
        Bootstrap Listener zum Starten des Springs Haupt-WebApplicationContexts
        von Spring. Delegiert an ContextLoader.
        Sofern der Log4jConfigListener verwendet wird ist dieser Listener danach
```

```
        in der web.xml zu registrieren.

        Anmerkung: Sollte ein aelterer Servlet-Container (2.2) verwendet werden ist
        ContextLoaderServlet anstelle von ContextLoaderListener zu verwenden.
 -->
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
...
</web-app>
```

4.2.2.2 LoggingMethodInterceptor

Der `LoggingMethodInterceptor` besitzt die folgenden Konfigurationsparameter:

`loggeDauer`, `loggeAufruf`, `loggeErgebnis`, `loggeDaten`, `loggeDatenBeiException` und `loggeMaximaleParameterGroesse` (vgl. Abschnitt 4.1.3.1).

Es werden zwei Instanzen des LogInterceptors mit unterschiedlichen Ausprägungen der oben genannten Parameter konfiguriert:

- `boundaryLogInterceptor`: Dieser wird verwendet, um Aufrufe an Systemgrenzen zu loggen. Es müssen entsprechende Pointcuts für alle Service-Schnittstellen, GUI-Controller und Batchausführungsbeans definiert werden (siehe Szenarien in Abschnitt 5.2.2.1 und 5.2.2.2).

- `komponentLogInterceptor`: Dieser wird verwendet, um Aufrufe an Komponentengrenzen zu loggen. Es müssen entsprechende Pointcuts für alle relevanten Komponenten-Schnittstellen definiert werden (siehe Szenario in Abschnitt 5.2.2.3).

Die Konfiguration der Beans ist im Folgenden dargestellt:

```
<beans ...>
  <!-- Autoproxies einschalten -->
  <aop:aspectj-autoproxy />

  <!-- Interceptor zum Loggen an Systemgrenzen. -->
  <bean id="boundaryLogInterceptor"
class="de.bund.bva.isyfact.logging.util.LoggingMethodIntercep
tor">
        <property name="loggeDauer">
            <value>${isylogging.boundary.loggeDauer}</value>
        </property>
        <property name="loggeAufruf">
            <value>${isylogging.boundary.loggeAufruf}</value>
        </property>
        <property name="loggeErgebnis">
            <value>${isylogging.boundary.loggeErgebnis}</value>
        </property>
        <property name="loggeDaten">
```

```
            <value>${isylogging.boundary.loggeDaten}</value>
        </property>
        <property name="loggeDatenBeiException">
            <value>${isylogging.boundary.loggeDatenBeiException}
            </value>
        </property>
        <property name="loggeMaximaleParameterGroesse">
            <value>${isylogging.boundary.
loggeMaximaleParameterGroesse}
            </value>
        </property>
  </bean>

  <!-- Interceptor zum Loggen an Komponentengrenzen. -->
  <bean id="componentLogInterceptor"
class="de.bund.bva.isyfact.logging.util.LoggingMethodIntercep
tor">
        <property name="loggeDauer">
            <value>${isylogging.component.loggeDauer}</value>
        </property>
        <property name="loggeAufruf">
            <value>${isylogging.component.loggeAufruf}</value>
        </property>
        <property name="loggeErgebnis">
            <value>${isylogging.component.loggeErgebnis}</value>
        </property>
        <property name="loggeDaten">
            <value>${isylogging.component.loggeDaten}</value>
        </property>
        <property name="loggeDatenBeiException">
            <value>${isylogging.component.loggeDatenBeiException}
            </value>
        </property>
        <property name="loggeMaximaleParameterGroesse">
            <value>${isylogging.component.
loggeMaximaleParameterGroesse}
            </value>
        </property>
  </bean>

<!-- AOP-Advice für das Logging konfigurieren -->
  <aop:config>

    <!-- Pointcuts an den Systemgrenzen -->
    <aop:advisor order="1000"
      advice-ref="boundaryLogInterceptor"
pointcut="
target(de.bund.bva.xyz.
fachanwendungxyz.service.meldung.httpinvoker.v1_0.MeldungRemo
teBean)
or
target(de.bund.bva.xyz.fachawendungxyz.gui.meldung.MeldungCon
troller)" />

    <!-- Pointcuts an den Komponentengrenzen -->
    <aop:advisor order="1000"
      advice-ref="componentLogInterceptor"
pointcut="target(de.bund.bva.xyz.registerxyz.core.meldung.Mel
dung)" />

</beans>
```

ACHTUNG – folgende Aspekte müssen zwingend beachtet werden:

- Jeder definierte Advisor belegt ca. 5 MB im Heap Space. Damit die Anzahl der Advisor gering gehalten wird, wird für jeden Interceptor nur ein Advisor definiert und die verschiedenen `targets` mit „`or`“ verbunden. Analog zum Pointcut an den Systemgrenzen im obigen Beispiel.

- Die Werte der Konfigurationsparameter (`isylogging.boundary.loggeDauer`, `isylogging.boundary.loggeAufruf` etc.) müssen als betriebliche Konfigurationsparameter in der Anwendungskonfiguration abgelegt werden und wie folgt gesetzt werden:

Ereignisschlüssel isy-logging		
Parameter	**Default**	**Bemerkung**
`isylogging.boundary.loggeDauer`	`true`	Muss auf „`true`“ sein, um das Logszenario „*Aufruf an Systemgrenze*“ umzusetzen.
`isylogging.boundary.loggeAufruf`	`true`	
`isylogging.boundary.loggeErgebnis`	`true`	
`isylogging.boundary.loggeDaten`	`false`	Kann in einer Testumgebung oder temporär in Produktion auf „`true`“ gesetzt werden, um die gesamte Schnittstellenkommunikation zur Unterstützung der Fehlersuche auszugeben.
`isylogging.boundary.loggeDatenBeiException`	`true`	Muss auf „`true`“ sein, um das Logszenario „*Rückliefern einer Exception an Systemgrenze*“ umzusetzen.
`isylogging.boundary.loggeMaximaleParameterGroesse`	`0`	Setzt die maximale Größe von Parametern, die ins Log geschrieben werden dürfen, in Bytes. Ist nur aktiv, wenn `loggeDaten` oder `loggeDatenBeiException` auf „`true`“ gesetzt ist. `0` bedeutet keine Beschränkung.
`isylogging.component.loggeDauer`	`false`	Die Ausgabe der Dauer und der durchgeführten Aufrufe an Komponentengrenzen führt zu einem hohen Logvolumen. Daher ist es sinnvoll, den Parameter im Produktivbetrieb nur bei Bedarf auf „`true`“ zu stellen (vgl. Logszenario „*Aufruf an Komponentengrenze*“).
`isylogging.component.loggeAufruf`	`false`	
`isylogging.component.loggeErgebnis`	`false`	Kann in einer Testumgebung oder temporär in Produktion zur Unterstützung der Fehlersuche „`true`“ gesetzt werden.
`isylogging.component.loggeDaten`	`false`	
`isylogging.component.loggeDatenBeiException`	`false`	

Ereignisschlüssel isy-logging		
Parameter	**Default**	**Bemerkung**
`isylogging.component.loggeMaximaleParameterGroesse`	`0`	Setzt die maximale Größe von Parametern, die ins Log geschrieben werden dürfen, in Bytes. Ist nur aktiv, wenn `loggeDaten` oder `loggeDatenBeiException` auf „`true`“ gesetzt ist. `0` bedeutet keine Beschränkung.

Anpassen der Konvertierung

Ist der Parameter `loggeDatenBeiException` auf `true` gesetzt, werden die übergebenen Schnittstellenparameter der Methode, bei der eine Exception aufgetreten ist, falls sie nicht zu groß sind oder die Größenbeschränkung deaktiviert ist, konvertiert (serialisiert) und in den Logeintrag übernommen. Handelt es sich bei einem der Parameter um eine Objektstruktur, wird diese Struktur teilweise rekursiv durchlaufen und sämtliche Attribute in den Logeintrag übernommen. Bei dieser Konvertierung gelten standardmäßig folgende Regeln:

- Sämtliche Objekte im Package `de.bund.bva` (inkl. Subpackages) werden rekursiv durchlaufen.
- Alle anderen Objekte, Primitives und Enums werden mit `toString` umgewandelt.

Dieses Verhalten **kann** bei Bedarf konfigurativ angepasst werden, in dem die beiden Constructor-Argumente `converterIncludes` und `converterExcludes` angegeben werden. Dabei gilt:

- Alle Objekte aus Packages (und Sub-Packages) in der Liste `converterIncludes` werden Rekursiv durchlaufen.
- Alle Objekte aus Packages (und Sub-Packages) in der Liste `converterExcludes` werden ignoriert.
- Alle anderen Objekte werden mit `toString` umgewandelt.

Gründe für die Anpassung der Konfiguration können bspw. sein:

- Exkludieren einzelner Packages, die nicht serialisiert werden können oder nicht relevant sind und dadurch zu unnötigen Loginhalten führen.

- Inkludieren einzelner Packages, falls die Anwendung nicht in der Domäne de.bund.bva entwickelt wird.

Eine exemplarische Konfiguration ist im Folgenden dargestellt:

```
    <!-- Interceptor zum Loggen an Systemgrenzen. -->
    <bean id="xyzInterceptor"
class="de.bund.bva.isyfact.logging.util.LoggingMethodInterceptor">
        <constructor-arg name="converterIncludes">
            <list>
                <!-- included Packages -->
                <value>x.y.z</value>
            </list>
        </constructor-arg>
        <constructor-arg name="converterExcludes">
            <list>
                <!-- excluded Packages -->
                <value>a.b.c</value>
            </list>
        </constructor-arg>
        …
    </bean>
```

4.2.2.3 LogApplicationListener

Im Folgenden wird die Konfiguration des `LogApplicationListener` dargestellt:

```
<beans ...>

  <bean id="statusLogger"
class="
de.bund.bva.isyfact.logging.util.LogApplicationListener">
    <property name="systemart"
      value="<SYSTEMART>" />
    <property name="systemname"
      value="<SYSTEMNAME>" />
    <property name="systemversion"
      value="${<VERSIONSNUMMER>}" />
  </bean>

</beans>
```

Die Platzhalter müssen dabei wie folgt ersetzt werden:

- `SYSTEMART`: Kürzel der Systemart gemäß den Namenskonventionen (siehe [Namenskonventionen]) – bspw. `REG` bei einem Register, `GA` bei einer Geschäftsanwendung, `QK` bei einer Querschnittskomponente, `BAT` bei einem Batch.

- `SYSTEMNAME`: Name der Anwendung analog zu Abschnitt 4.2.1.1.

- `VERSIONSNUMMER`: Versionsnummer der Anwendung. Diese ist als interner Konfigurationsparameter in der Anwendung abzulegen.

4.2.3 Umgang mit Drittsoftware

Es muss sichergestellt werden, dass alle Bibliotheken – auch solche die nicht nach den Vorgaben der Register Factory entwickelt wurden – logback,

mit der in Abschnitt 4.2.1 definierten Konfiguration, nutzen. Dadurch wird gewährleistet, dass die definierten Vorgaben zu Logdateien und Struktur der Logeinträge einheitlich umgesetzt werden.

Beim Einsatz von Bibliotheken, die nicht nach der Register Factory entwickelt wurden, muss daher unterschieden werden:

- **Die Bibliothek loggt mittels logback oder SLF4J**: Es sind keine Maßnahmen notwendig.

- **Die Bibliothek setzt ein anderes Logging-Framework ein**: Es muss eine entsprechende „Bridge“ integriert werden, welche die Aufrufe der Bibliothek an das jeweilige Logging-Framework auf logback umleitet.

SLF4J stellt bereits fertige Bridges für alle gängigen Logging-Frameworks zur Verfügung, deren Einsatz im Folgenden beschrieben wird. Grundsätzlich ist es unkritisch, wenn alle Bridges konfiguriert werden. Um die Komplexität der Konfiguration und deren Wartung nicht unnötig zu erhöhen, sollten jedoch nur die Bridges eingerichtet werden, die auch tatsächlich benötigt werden.

Bei sämtlichen Bridges muss sichergestellt werden, dass das `logback.jar` als einzige SLF4J-Implementierung in der Anwendung vorhanden ist.

4.2.3.1 Bridge für log4j

SLF4J stellt mit der Bibliothek `log4j-over-slf4j.jar` eine Bridge von log4j zu slf4j zur Verfügung. Diese kann wie folgt eingesetzt werden:

1. `log4j*.jar` aus der Anwendung entfernen (bzw. sicherstellen, dass diese durch Maven nicht in die Anwendung integriert werden)

2. `log4j-over-slf4j.jar` in die Anwendung ergänzen

4.2.3.2 Bridge für commons-logging

Analog zu log4j in Abschnitt 4.2.3.1, nur dass die Bibliothek `jcl-over-slf4j.jar` verwendet wird.

4.2.3.3 Bridge für java.util.logging

SLF4J stellt für die java.util.logging API ebenfalls eine Bridge zur Verfügung (`jul-to-slf4j.jar`). Um die Bridge zu aktivieren müssen zunächst alle vorhandenen Log-Handler entfernt und danach ein Handler zum Weiterleiten der Log-Aufrufe an SLF4J installiert werden[3]. Diese kann wie folgt umgesetzt werden:

1. `jul-to-slf4j.jar` in die Anwendung ergänzen

[3] http://www.slf4j.org/api/org/slf4j/bridge/SLF4JBridgeHandler.html

2. Den folgenden Abschnitt in die Spring-Konfiguration der Anwendung ergänzen:

```
<!-- Bridge von java.util.logging nach SLF4J einrichten-->
<!-- 1. Entferne vorhandene Log-Handler -->
<bean id="slf4JBridgeHandler"
class="org.slf4j.bridge.SLF4JBridgeHandler"
  init-method="removeHandlersForRootLogger"/>
<!-- 2. Installiere den Handler der Bridge -->
<bean class="org.slf4j.bridge.SLF4JBridgeHandler"
  init-method="install"
  depends-on="slf4JBridgeHandler"/>
```

5. Vorgaben zur Logerstellung

Die Zielsetzung des Loggings ist es, unterschiedliche Auswertungen zu ermöglichen, um damit verschiedene Problemstellungen und Informationsbedarfe, die während des Betriebs der Systeme entstehen, einfach und effizient beantworten zu können. Grundlage hierfür bildet zum einen die technische Vereinheitlichung des Loggings, die in den vorangegangen Abschnitten (Nutzung und Konfiguration) beschrieben wurde. Zum andern muss das Logging jedoch insbesondere auch inhaltlich – also *wann* wird *was* geloggt – einheitlich und zielgerichtet im Hinblick auf die verschiedenen Auswertungen erfolgen. Dadurch wird sichergestellt, dass die Logeinträge einfach ausgewertet werden können und alle notwendigen Informationen vorliegen.

Aus diesem Grund werden im folgenden Abschnitt zunächst die verschiedenen Auswertungen beschrieben, die für alle Anwendungen relevant sind. Bei Entwurf eines Systems können systemspezifische Anforderungen definiert werden, die analog zu den hier aufgeführten Themen adressiert werden müssen. Es ist Aufgabe des Technischen Chefdesigners diese Anforderungen im Rahmen des Systementwurfs abzustimmen und zu berücksichtigen.

Die konkreten Szenarien, in denen Logeinträge zu erstellen sind, werden in Abschnitt 5.2 definiert.

5.1. Auswertungen

In diesem Abschnitt werden Auswertungen beschrieben, die auf den Logs der Anwendungslandschaft durchgeführt werden können müssen. Die Auswertung erfolgt dabei meist durch den Betrieb und nicht durch die Entwickler. Es ist jedoch Aufgabe der Entwickler sämtliche Informationen in den Logs bereitzustellen, so dass die Szenarien effizient durchgeführt werden können.

Es wird zwischen folgenden Akteuren unterschieden:

- Betrieb: Mitarbeiter der IT-Abteilung, in der das System bzw. die Anwendungslandschaft betrieben wird.

- Entwickler: Mitarbeiter der Entwicklungsabteilung, durch die die Anwendung entwickelt, gewartet und/oder weiterentwickelt wird.

- Fachbereich: Mitarbeiter des Fachbereichs / der Fachabteilung, durch die die Anwendung fachlich betreut und geführt wird.

5.1.1 Schwerwiegenden Fehler erkennen und behandeln

Akteur	**Betrieb, Entwickler**
Log-Level	**FATAL**
Kategorie	**FATAL**
Beschreibung	Schwerwiegende Fehler, von denen sich die Anwendung nicht erholen kann und beendet werden muss ("Unrecoverable Error"), müssen umgehend erkannt werden. Zu diesem Zweck überwacht das betriebliche Monitoring das Log-Level FATAL und alarmiert den Betrieb bei jedem neuen Eintrag. Logeinträge im Level FATAL signalisieren, dass der Systembetrieb unterbrochen ist und der Betrieb schnellstmöglich aktiv werden muss, um die Fehlerursache mit Hilfe der bereitgestellten Informationen zu analysieren, zu beheben und die Anwendung wieder neu zu starten. Falls der Betrieb im Rahmen der Fehleranalyse feststellt, dass die Exception auf einen Fehler in der Anwendung zurückzuführen ist, wird der Logeintrag zur Fehleranalyse an die Entwickler übergeben. Beispiele: • `OutOfMemoryError` • `StackOverflowError`

5.1.2 Beeinträchtigung des Betriebs erkennen und behandeln

Akteur	**Betrieb**, **Entwickler**
Log-Level	**ERROR**
Kategorie	**ERROR**
Beschreibung	Beeinträchtigungen des Systembetriebs (bspw. Netzwerkverbindung kann nicht aufgebaut werden), müssen umgehend erkannt werden. Zu diesem Zweck überwacht das betriebliche Monitoring das Log-Level ERROR und alarmiert den Betrieb bei jedem neuen Eintrag. Logeinträge im Level ERROR signalisieren, dass der Fehler durch die Anwendung behandelt wurde und die Anwendung weiterläuft. Der Betrieb muss jedoch schnellstmöglich aktiv werden, um die Fehlerursache mit Hilfe der bereitgestellten Informationen zu analysieren, zu beheben und damit ein erneutes Auftreten des Fehlers zu verhindern. Falls der Betrieb im Rahmen der Fehleranalyse feststellt, dass die Exception auf einen Fehler in der Anwendung zurückzuführen ist, wird der Logeintrag zur Fehleranalyse an die Entwickler übergeben. Beispiele: • Fehler bei Netzwerkverbindung • Datenbankverbindung konnte nicht aufgebaut werden

5.1.3 Unerwartetes Systemverhalten erkennen und behandeln

Akteur	**Entwickler**
Log-Level	**WARN**
Kategorie	**WARN**
Beschreibung	Unerwartetes Systemverhalten muss umgehend erkannt werden. Zu diesem Zweck überwacht das betriebliche Monitoring das Log-Level WARN. Die entsprechenden Logeinträge werden an die Entwicklungsabteilung zur Analyse des Verhaltens und Identifikation notwendiger Maßnahmen übergeben. Logeinträge im Level WARN signalisieren, dass der Fehler den Systembetrieb (wahrscheinlich) nicht beeinträchtigt. Die bereitgestellten Informationen richten sich an die Entwickler. Der Betrieb muss im Hinblick auf die Fehleranalyse hierbei zunächst nicht aktiv werden. Beispiele: • Inkonsistenzen im Datenbestand • `IllegalArgumentException`

5.1.4 Betriebliche Überwachung

Akteur	**Betrieb**
Log-Level	**INFO**
Kategorie	**METRIK**
Beschreibung	Logeinträge können dazu verwendet werden, Statistiken zu ermitteln, um eine betriebliche Überwachung des Systems zu realisieren. Die folgenden Auswertungen werden dazu durchgeführt: • Ermittlung der Anzahl der Aufrufe eines Services innerhalb der letzten Minute. • Ermittlung der Anzahl der Aufrufe eines Services, die einen Fehler erzeugt haben, innerhalb der letzten Minute. • Ermittlung der Durchschnittsdauer der letzten Aufrufe eines Services. • Ermittlung des Zeitpunkts, wann die letzte Prüfung des Systems durchgeführt wurde und wann die letzte Prüfung erfolgreich war. Detaillierte Informationen zur Systemprüfung und der zu erstellenden Logeinträge ist in **[ÜberwachungKonfigKonzept]** beschrieben.

5.1.5 Performance überwachen

Akteur	**Betrieb**
Log-Level	**INFO**
Kategorie	**PROFIL**
Beschreibung	„Performance-Analyse“ meint die Analyse von Laufzeiten an bestimmten kritischen Stellen der Anwendungslandschaft (bspw. an Service-Methoden) und insbesondere deren Entwicklung über die Zeit. Dies wird durchgeführt, um • Engpässe zu erkennen, bspw. wenn Aufrufe einer Komponente zunehmend länger dauern. • Auswirkung einer Änderung auf die Performance zu bewerten, bspw. um Laufzeiten vor und nach einer Aktualisierung der Datenbank zu vergleichen.

5.1.6 Nutzungshäufigkeit auswerten

Akteur	**Betrieb**
Log-Level	**INFO**
Kategorie	**METRIK**
Beschreibung	Die Analyse der Nutzungshäufigkeit bestimmter kritischer Stellen der Anwendungslandschaft (bspw. von Service-Methoden oder Komponenten) und insbesondere deren Entwicklung über die Zeit wird zu folgenden Zwecken durchgeführt: • Anomalien in Nutzung erkennen: Durch die betriebliche Überwachung der Nutzungshäufigkeit von Systemen können Ausreißer im Nutzerverhalten erkannt werden, die ggf. ein Fehlverhalten des Aufrufers (bspw. große Anzahl an Aufrufen weil Testsystem auf Produktivumgebung gelenkt ist) oder gar auf einen Missbrauchsversuch (Vielzahl unautorisierter Zugriffe, um Benutzerdaten zu erraten) hindeuten. • Auswirkung von Änderungen prognostizieren: Es kann bspw. überprüft werden, wie oft eine alte Schnittstelle noch verwendet wird und ob (bzw. mit welchem Aufwand) dieses abgeschaltet werden kann. • Auswirkung von Änderungen analysieren: Es kann bspw. überprüft werden, ob eine Erhöhung der Cache-Größe zur gewünschten Reduktion der Nachbarsystemaufrufe geführt hat.

5.1.7 Systemzustand und -ereignisse überwachen

Akteur	**Betrieb**
Log-Level	**INFO**
Kategorie	**JOURNAL**
Beschreibung	Die Analyse des Systemzustands und der Systemereignisse umfasst bspw. die Analyse, welche Version sich mit welcher Konfiguration in Betrieb befand, welche Änderungen vorgenommen wurden, ob die Anwendung gestartet oder beendet wurde etc. Diese Analyse wird querschnittlich zur Unterstützung der anderen Analysen durchgeführt, um bspw. Fehler auf Änderungen des Systemzustands zurückzuführen, oder Performance-Schwankungen zu erklären.

5.1.8 Verarbeitung eines Aufrufs in Anwendungslandschaft nachvollziehen

Akteur	**Entwickler**
Log-Level	**INFO**
Kategorie	**JOURNAL**
Beschreibung	Das Nachvollziehen, durch welche Systeme ein Aufruf der Anwendungslandschaft verarbeitet und weitergeleitet wurde (die Korrelation der Logs zu einem Aufruf aus verschiedenen Systemen), dient den folgenden Zwecken: • Unterstützung der Fehleranalyse, falls die systeminternen Logeinträge nicht ausreichend sind, bspw. weil der Fehler durch ein aufrufendes System verursacht wurde. • Nachvollziehen der Auswirkung eines Fehlers, um bspw. erkennen zu können, ob durch den Aufruf in einem anderen System bereits Daten verändert wurden, die zurückgesetzt werden müssen.

5.1.9 Fachliche Verarbeitung eines Aufrufs nachvollziehen

Akteur	**Fachbereich**
Log-Level	**INFO**
Kategorie	**JOURNAL**
Beschreibung	Der Fachbereich kann die Anforderung an ein System stellen, dass die fachliche Verarbeitung eines Aufrufs über das Logging nachvollziehbar sein muss. Hierzu werden an definierten Stellen in der Anwendung spezifische Logeinträge erstellt – bspw. beim Start oder Beenden eines Anwendungsfalls, beim Aufruf einer Anwendungsfunktion etc. Die Anforderungen an das Logging sowie die Auswertung der Logeinträge sind spezifisch für das jeweilige System und müssen mit dem Fachbereich abgestimmt werden.

5.1.10 Fehleranalyse (Debugging)

Akteur	**Entwickler**
Log-Level	**DEBUG, TRACE**
Kategorie	**DEBUG**
Beschreibung	Die Fehleranalyse ist das „klassische“ Szenario der Log-Auswertung. Hierbei werden detaillierte Debug-Informationen analysiert, um die Ursache eines Fehlers im Programmcode zu finden und diesen zu beheben.

5.2. Logszenarien

In diesem Abschnitt werden die verschiedenen Logszenarien beschrieben, die definieren, *wann welche* Logeinträge zu erstellen sind, um die im vorhergehenden Abschnitt definierten Auswertungen zu ermöglichen.

Die Bibliothek `isy-logging` stellt bereits einige Mechanismen bereit, durch die die notwendigen Logeinträge für einzelne Auswertungen querschnittlich und rein konfigurativ umgesetzt werden können. Diese sind in Abschnitt 5.2.2 beschrieben.

Logeinträge die individuell in bei der Anwendungsentwicklung zu erstellen sind, sind in 5.2.3 beschrieben.

Wichtig ist, dass bei der Umsetzung einer Anwendung *keine* Logeinträge erstellt werden, zu denen es *kein* Szenario gibt – oder umgekehrt: sollte es sinnvoll sein einen Logeintrag zu erstellen, dann muss dafür auch ein Szenario definiert werden.

Die Szenarien sind nach folgendem Schema aufgebaut:

Beschreibung	*Beschreibung der Situation innerhalb einer Anwendung.*
Logging	*Das durchzuführende Logging.*
Auswertungs-szenario	*Die Auswertungen, für die die erstellten Logeinträge verwendet werden.*

5.2.1 Vorgaben für alle Logszenarien

Die folgenden Regeln sind für alle Logeinträge zu beachten:

1. **Keine Binärdaten loggen**: Binärdaten sind nur schwer auswertbar und führen potentiell zu sehr langen Einträgen. Logeinträge größer 64 KByte führen zu Fehlern bei der weiteren Verarbeitung. Binärdaten dürfen daher nicht gelogged werden.

2. **Größe der Parameter beschränken**: Beim Loggen der Schnittstellenkommunikation können durch große Objektstrukturen ebenfalls sehr große Logeinträge entstehen. Das Loggen von Parametern kann durch entsprechende Konfiguration auf eine Maximalgröße beschränkt werden.

5.2.2 Konfiguration

Die folgenden Szenarien können rein konfigurativ umgesetzt werden, mit Mitteln, die durch `isy-logging` bereitgestellt werden. Sollte einer dieser Mechanismen in einer Anwendung nicht umgesetzt werden können (bspw. weil die Anwendung nur Teile der IsyFact einsetzt und bspw. Spring nicht verwendet), müssen die entsprechenden Einträge explizit durch Aufruf des Logging-Frameworks erstellt werden.

5.2.2.1 Aufruf an Systemgrenze

Beschreibung	Es wird eine Außenschnittstelle des Systems – Service, GUI-Controller oder Batch – aufgerufen (eingehender Aufruf).
Logging	Der Aufruf der Methode wird mit Hilfe des `LogInterceptor` geloggt. Dieser muss gemäß Abschnitt 4.2.2.2 für alle Außenschnittstellen des Systems konfiguriert sein.
Auswertungs-szenario	• Performance analysieren • Nutzungshäufigkeit analysieren • Verarbeitung eines Aufrufs in Anwendungslandschaft nachvollziehen • Betriebliche Überwachung

5.2.2.2 Rückliefern einer Exception an Systemgrenze

Beschreibung	Beim Aufruf eines Systems ist ein Fehler aufgetreten. Es wird eine Exception an den Aufrufer zurückgegeben.
Logging	Es müssen die übermittelten Eingabeparameter mit Hilfe des `LogInterceptor` geloggt werden. Dieser muss gemäß Abschnitt 4.2.2.2 für alle Außenschnittstellen des Systems konfiguriert sein.
Auswertungs-szenario	• Fehleranalyse (Debugging)

5.2.2.3 Aufruf an Komponentengrenze

Beschreibung	Es wird eine Methode einer Komponentenschnittstelle im Anwendungskern aufgerufen (eingehender Aufruf).
Logging	Das Loggen von Aufrufen an Komponentengrenzen liefert insbesondere für die Performanceanalyse wichtige Informationen, führt jedoch in den meisten Anwendungen zu einem sehr hohen Logvolumen. Jede Anwendung muss den LogInterceptor gemäß Abschnitt 4.2.2.2 konfigurieren, so dass das Logging an den Komponentengrenzen bei Bedarf aktiviert werden kann.
Auswertungs-szenario	• Performance überwachen • Nutzungshäufigkeit analysieren • Verarbeitung eines Aufrufs in Anwendungslandschaft nachvollziehen

5.2.2.4 Aufruf eines DAOs

Beschreibung	Es wird eine Methode eines DAOs aufgerufen (eingehender Aufruf).
Logging	Der Aufruf der Methode wird mit Hilfe des `LogInterceptor` geloggt. Dieser muss gemäß Abschnitt 4.2.2.2 für alle Komponentenschnittstellen konfiguriert sein.
Auswertungs-szenario	• Performance überwachen • Nutzungshäufigkeit analysieren • Verarbeitung eines Aufrufs in Anwendungslandschaft nachvollziehen

5.2.2.5 Aufruf eines Nachbarsystems

Beschreibung	Es wird ein entfernter Service eines Nachbarsystems aufgerufen.
Logging	Der Aufruf der Methode wird mit Hilfe des `LogInterceptors` geloggt. Dazu muss in der aufrufenden Klasse gemäß Abschnitt 4.1.3.1 eine Instanz der Klasse erstellt und das Remote-Interface des Nachbarsystem mit Hilfe der Methode `rufeNachbarsystemAuf` aufgerufen werden. Dies wird bereits durch die Erweiterung der HTTP-Invoker-Implementierung (IsyHttpInvokerProxyFactoryBean) des Bausteins Service umgesetzt (siehe **[ServiceDetailkonzept]**), so dass hierfür keine Anpassung notwendig ist. Bei Nachbarsystemen, die selbst kein Register Factory-konformes Logging umsetzen (Drittsoftware wie bspw. ein Suchverfahren), kann es notwendig sein, zusätzliche Informationen in der aufrufenden Anwendung zu loggen. Entsprechende Vorgaben werden in den Nutzungskonzepten der jeweiligen Bausteine definiert.
Auswertungs-szenario	• Nutzungshäufigkeit analysieren • Verarbeitung eines Aufrufs in Anwendungslandschaft nachvollziehen • *Weitere systemspezifische Auswertungen*

5.2.2.6 Hochfahren / Herunterfahren

Beschreibung	Ein Anwendungssystem oder ein Batch wird gestartet oder beendet.
Logging	Der Vorgang wird durch den `LogApplicationListener` geloggt. Dieser muss gemäß Abschnitt 4.2.2.3 konfiguriert sein.
Auswertungs-szenario	• Systemzustand und -ereignisse überwachen

5.2.2.7 Neueinlesen eines geänderten Konfigurationsparameters

Beschreibung	Es wird festgestellt, dass sich ein Konfigurationsparameter der betrieblichen Konfiguration oder eine Laufzeitkonfiguration geändert hat. Der geänderte Wert wird im laufenden Betrieb übernommen.
Logging	Es muss ein Logeintrag erstellt werden, der die Änderung des Konfigurationsparameters dokumentiert: `log.info("Der Konfigurationsparameter <Parameter> wurde geändert von <Alter Wert> auf <Neuer Wert>", <Name des Parameters>, <Alter Wert>, <Neuer Wert>)` Dies wird durch die Klasse `ReloadablePropertyKonfiguration` der Bibliothek isy-konfiguration bereits umgesetzt, so dass bei deren Verwendung hierfür nichts mehr zu tun ist.
Auswertungs-szenario	• Systemzustand und -ereignisse überwachen

5.2.2.8 Loggen der Schnittstellenkommunikation

Beschreibung	In Ausnahmefällen kann es notwendig sein, Teile oder die gesamten Daten, die über eine Schnittstelle ausgetauscht werden, zu loggen. Dies ist insbesondere dann der Fall, wenn: • Es sich um eine technisch sehr komplexe oder proprietäre Schnittstelle handelt. • Ein „unerklärliches“ Verhalten im Systembetrieb festgestellt wurde, welches mit den Standard Debug-Ausgaben nicht nachvollzogen werden kann.
Logging	Das Erstellen der Logeinträge erfolgt mittels des LogInterceptors der bereits für die Szenarien in den Abschnitten 5.1.1 und 5.1.2 konfiguriert wurde. Zur Ausgabe der Schnittstellenkommunikation muss der Schalter `loggeDaten` auf `true` gesetzt werden (Abschnitt 4.2.2.2).
Auswertungs-szenario	• Fehleranalyse (Debugging)

5.2.3 Anwendungsentwicklung

In diesem Abschnitt sind alle Szenarien beschrieben, bei denen Logeinträge im Anwendungscode explizit durch den Entwickler vorzusehen sind.

Wenn durch ein Logszenario ein Eintrag im Level INFO gefordert ist, muss ein entsprechender Ereignisschlüssel definiert werden – dies ist in Abschnitt 3.2.3 beschrieben. Die definierten Schlüssel müssen im Systementwurf dokumentiert werden – analog zu Kapitel 6 dieses Dokuments.

5.2.3.1 Behandlung einer Exception

Beschreibung	Es wird eine Exception gefangen und behandelt. Wichtig: Exceptions werden nur geloggt, wenn Sie auch behandelt werden. Wird eine Exception nicht behandelt (also an den Aufrufer weitergereicht), wird sie auch nicht geloggt.
Logging	Je nach Schwere des Fehlers wird die Exception in einem der folgenden Log-Level geloggt (siehe Abschnitt 3.2.1): • **FATAL**: Falls es sich um einen schwerwiegenden Fehler handelt (vgl. auch Szenario „Schwerwiegenden Fehler erkennen und behandeln“ in Abschnitt 5.1.1). • **ERROR**: Falls der Fehler zur Beeinträchtigung des Systembetriebs führt, das System aber weiterlaufen kann (vgl. auch Szenario „Beeinträchtigung des Betriebs erkennen und behandeln“ in Abschnitt 5.1.2) • **WARN**: Wenn es sich um ein inkonsistentes / unerwartetes Systemverhalten handelt, welches der Entwicklungsabteilung mitgeteilt werden muss (vgl. auch Szenario „Unerwartetes Systemverhalten erkennen und behandeln“ in Abschnitt 5.1.3). • **INFO**: Wenn es sich um einen „erwarteten“ Fehler handelt, der durch das System behandelt werden. Dies umfasst insbesondere auch Exceptions, die mit einem Retry behandelt werden – bspw. wenn eine `OptimisticLockException` gefangen und die Anfrage wiederholt wird. Das Erstellen der Logeinträge erfolgt mittels der Methoden `log.fatal(…)`, `log.error(…)`, `log.warn(…)` und `log.info(…)`. Sollte es zwingend notwendig sein, datenschutzrelevante fachliche Daten in den Logeintrag zu schreiben, muss stattdessen die entsprechende Methode zum Loggen von Fachdaten verwendet werden (vgl. Abschnitt 4.1.2.2): `log.fatalFachdaten(…)`, `log.errorFachdaten(…)` und `log.warnFachdaten(…)`. Die Lognachricht muss das eingetretene Szenario kurz und möglichst Präzise beschreiben, bspw.: „Fehler beim Zugriff auf die Datenbank“. Anmerkung: Zur Fehleranalyse sind insbesondere der Fehlerschlüssel, Fehlertext und der Stacktrace relevant. Diese werden automatisch durch das Logging-Framework geloggt und müssen daher nicht manuell in die Lognachricht übernommen werden.

Auswertungs-szenario	• Schwerwiegenden Fehler erkennen und behandeln • Beeinträchtigung des Betriebs erkennen und behandeln • Unerwartetes Systemverhalten erkennen und behandeln

5.2.3.2 Wichtige Systemereignisse

Beschreibung	Es tritt ein wichtiges Ereignis auf, welches für die Durchführung der folgenden Auswertungen relevant ist: • Betriebliche Überwachung • Performance überwachen • Nutzungshäufigkeit auswerten • Systemzustand und -ereignisse überwachen Es ist Aufgabe des technischen Chefdesigners diese Stellen im Rahmen des Systementwurfs zu Identifizieren und mit dem Auftraggeber abzustimmen.

Logging	Es muss ein spezifischer Ereignisschlüssel definiert, im Systementwurf dokumentiert und ein Logeintrag im Level INFO erstellt werden. Tritt an mehreren Stellen in der Anwendung das gleiche zu loggende Ereignis auf, kann der gleiche Ereignisschlüssel verwendet werden.
	Das Loggen der Einträge erfolgt mit der Methode
	`log.info(<kategorie>, <schluessel>, <nachricht>, <werte>)`
	Bspw.:
	`log.info(LogKategorie.JOURNAL, "SYSXY01234", "Das Sucherfahren lieferte einen Datensatz mit ID {}. Dieser ist nicht im Bestand vorhanden und wird im Suchverfahren gelöscht.", "12345");`
	`log.info(LogKategorie.SICHERHEIT, "SYSXY01235", "Innerhalb der letzten Minute wurden {} ungültige Logins mit Anwendernamen {} durchgeführt.", "29", "max_muster");`

Auswertungs-szenario	• Betriebliche Überwachung • Performance überwachen • Nutzungshäufigkeit auswerten • Systemzustand und -ereignisse überwachen

5.2.3.3 Durchführen einer Bulk-Query

Beschreibung	Es wird eine native SQL-Bulk-Query (Manipulation mehrerer Datensätze) in der Datenbank durchgeführt.

Logging	Es muss ein Logeintrag erstellt werden, der die Query beschreibt und die Anzahl der betroffenen Datensätze als Platzhalter enthält:
	`log.debug("<Beschreibung der Query mit Platzhalter für Anzahl der betroffenen Datensätze und Name der Query>", <Name der Query>, <Anzahl Datensätze>);`
	Bspw.:
	`log.debug("Query {} zum Löschen veralteter Sachverhalte wurde ausgeführt. Es wurden {} Sachverhalte gelöscht.", <Name der Query>, <Anzahl Datensätze>);`

Auswertungs-szenario	• Fehleranalyse

5.2.3.4 Unterstützung der Fehleranalyse (Debug)

Beschreibung	An allen Stellen der Verarbeitungslogik, die für eine spätere Fehleranalyse *relevant* sind, müssen entsprechende DEBUG-Einträge erstellt werden. Welche Stellen relevant sind, ist abhängig vom konkreten System und kann nicht allgemein festgelegt werden. Es liegt im Ermessen des Technischen Chefdesigners und der Entwickler, diese Stellen zu identifizieren. Typische Szenarien sind: • Unterschiedliche Zweige in `if-then-else` Anweisungen. • Passieren kritischer Verarbeitungsschritte bspw. dem Abschluss der Prüfung übermittelter Daten.

Logging	Das Erstellen der Logeinträge erfolgt mittels der Methode:
	`log.debug("<Nachricht>", "<Werte>");`
	Sollte es zwingend notwendig sein, fachliche, datenschutzrelevante Daten in den Logeintrag zu schreiben, muss stattdessen die Methode
	`log.debugFachdaten("<Nachricht>", "<Werte>");`
	verwendet werden (vgl. Abschnitt 4.1.2.2).
	Da dieses Szenario eine Vielzahl unterschiedlicher Fälle zusammenfasst, können hierfür keine konkreten Nachrichten vorformuliert werden. Zu beachten ist, dass die Nachrichten primär durch Entwickler im Falle eines Fehlers ausgewertet werden, meist aber nicht dem Entwickler, der den Code geschrieben hat.
	Die jeweilige Nachricht **muss** daher **zwingend** so formuliert werden, dass das Ereignis auch ohne Kenntnis des Quellcodes verstanden werden kann – analog zu den in Kapitel 5.2.3.5 definierten Lognachrichten. Bspw.:
	• **Falsch**: „Hier“, „Fertig“, „1“, „OK“
	• **Richtig**: „Regelwerksprüfungen erfolgreich abgeschlossen“, „Führe Suche nach Personen durch“
	Anmerkung: Für eine detaillierte Fehleranalyse ist es natürlich meist unerlässlich auch den Programmcode einzusehen. Durch die verständliche Formulierung der Logeinträge kann der Hergang, der zu einem Fehler geführt hat, jedoch viel einfacher und schneller nachvollzogen werden und damit die Fehlerquelle schneller identifiziert werden.

Auswertungs-szenario	• Fehleranalyse (Debugging)

5.2.3.5 Fachliche Korrelation von Einträgen

Beschreibung	In komplexen Verfahren kann es notwendig sein, die erstellten Logeinträge fachlich in Verbindung zueinander zu setzen – bspw. Kennzeichnen aller Logeinträge, die sich auf einen bestimmten Datensatz beziehen, durch Aufnahme des eindeutigen Schlüssel des betroffenen Datensatzes. Wenn diese Anforderung gegeben ist, kann dies beim Aufruf an der Systemgrenze (vgl. Szenario 5.2.2.1 *Aufruf an Systemgrenze*) wie im Folgenden beschrieben berücksichtigt werden.
Logging	An der Systemgrenze wird der eindeutige Schlüssel des Datensatzes in den MDC aufgenommen. Dabei müssen folgende Aspekte berücksichtigt werden: • Das Setzen und Entfernen des Schlüssels im MDC erfolgt direkt über die Klasse „MDC“, die von SLF4J bereitgestellt wird. • Der MDC ist eine Map (Mapped Diagnostic Context - Name-Wert-Paar). Als Name muss ein sprechender, gut auswertbarer Name verwendet werden bspw. „kundennummer“. • Sollte es sich bei dem Schlüssel um datenschutzrelevante Daten handeln, muss der MDC mit einem Marker als „fachlich“ gekennzeichnet werden (vgl. Abschnitt 4.1.3.3). • Der Schlüssel und der ggf. gesetzte Marker müssen in jedem Fall beim Verlassen der Methode, in der sie gesetzt wurden, wieder aus dem MDC entfernt werden. Das Entfernen sollte daher in einem „finally“-Block stattfinden.
Auswertungs-szenario	• Fachliche Verarbeitung eines Aufrufs nachvollziehen • Fehleranalyse (Debugging)

5.2.3.6 Loggen fachlicher Operationen

Beschreibung	Der Fachbereich hat für das System Anforderungen definiert, dass die Verarbeitung einzelner Aufrufe fachlich nachvollziehbar sein muss. Dies ist im Auswertungsszenario „Fachliche Verarbeitung eines Aufrufs nachvollziehen“ (siehe Abschnitt 5.1.9) beschrieben.
Logging	Die zu erstellenden Logeinträge sind spezifisch für das jeweilige Verfahren. Zu beachten ist jedoch, dass die Aufrufe im Log-Level INFO zu erstellen sind. Für alle Logeinträge, die dem gleichen Zweck dienen (also der gleichen fachlichen Anforderung), muss der gleiche Ereignisschlüssel verwendet werden.
Auswertungs-szenario	• Fachliche Verarbeitung eines Aufrufs nachvollziehen • Fehleranalyse (Debugging)

6. Ereignisschlüssel isy-logging

Im Folgenden werden die spezifischen Ereignisschlüssel der Komponente `isy-logging` beschrieben:

Ereignisschlüssel isy-logging			
Schlüssel[4]	**Level**	**Kategorie**	**Text**
EISYLO01001 (EPLILO01001)	INFO	JOURNAL	Methode <Klasse.Methode> wird aufgerufen.
EISYLO01002 (EPLILO01002)	INFO	METRIK	Aufruf von <Klasse. Methode> erfolgreich beendet.
EISYLO01003 (EPLILO01003)	INFO	METRIK	Aufruf von <Klasse. Methode> mit Fehler beendet.
EISYLO01004 (EPLILO01004)	INFO	PROFILING	Aufruf von <Klasse. Methode> erfolgreich beendet. Der Aufruf dauerte <Dauer in Millisekunden> ms.
EISYLO01005 (EPLILO01005)	INFO	PROFILING	Aufruf von <Klasse. Methode> mit Fehler beendet. Der Aufruf dauerte <Dauer in Millisekunden> ms.
EISYLO01011 (EPLILO01011)	INFO	JOURNAL	Die Methode <Klasse.Methode> des Nachbarssystems <systemname> wird unter der URL <url> aufgerufen.
EISYLO01012 (EPLILO01012)	INFO	METRIK	Aufruf von <Klasse. Methode> des Nachbarssystems <systemname> unter der URL <url> erfolgreich beendet.
EISYLO01013 (EPLILO01013)	INFO	METRIK	Aufruf von <Klasse. Methode> des Nachbarssystems <systemname> unter der URL <url> mit Fehler beendet.
EISYLO01014 (EPLILO01014)	INFO	PROFILING	Aufruf von <Klasse. Methode> des Nachbarssystems <systemname> unter der URL <url> erfolgreich beendet. Der Aufruf dauerte <Dauer in Millisekunden> ms.

Ereignisschlüssel isy-logging			
Schlüssel[4]	**Level**	**Kategorie**	**Text**
EISYLO01015 (EPLILO01015)	INFO	PROFILING	Aufruf von <Klasse. Methode> des Nachbarssystems <systemname> unter der URL <url> mit Fehler beendet. Der Aufruf dauerte <Dauer in Millisekunden> ms.
EISYLO02001 (EPLILO02001)	INFO	JOURNAL	ApplicationContext des Systems <Systemname> (Systemart) wurde gestartet oder aktualisiert.
EISYLO02002 (EPLILO02002)	INFO	JOURNAL	Der ApplicationContext des Systems <Systemname> (Systemart) wurde gestopped.
EISYLO02003 (EPLILO02003)	INFO	JOURNAL	Die Systemversion ist <Versionsnummer>.
EISYLO02004 (EPLILO02004)	INFO	JOURNAL	Der Laufzeitparameter <Parametername> besitzt den Wert <Wert>.

[4] Bis zur Version 1.0.7 von isy-logging wurde EPLILO als Präfix der Schlüssel verwendet. Diese alten Ereignisschlüssel sind jeweils in Klammer angegeben.

7. Quellenverzeichnis

[FehlerbehandlungKonzept]
Konzept Fehlerbehandlung
20_Bausteine\Fehlerbehandlung\Konzept_Fehlerbehandlung.pdf .

[LoggingKonzept]
Konzept Logging
20_Bausteine\Logging\Konzept_Logging.pdf.

[Namenskonventionen]
IsyFact – Namenskonventionen
00_Allgemein\IsyFact-Namenskonventionen.pdf.

[NutzungskonzeptPlisUtil]
Nutzungskonzept PLIS-Util
20_Bausteine\Util\Nutzungskonzept_plis-util.pdf.

[ServiceDetailkonzept]
Detailkonzept Komponente Service
10_Blaupausen\technische_Architektur\Detailkonzept_Komponente_Service.pdf .

[ServicekommunikationKonzept]
Grundlagen der Servicekommunikation innerhalb der Plattform
10_Blaupausen\Integrationsplattform\Grundlagen_interne_Servicekommunikation.pdf .

[ÜberwachungKonfigKonzept]
Konzept Überwachung und Konfiguration
20_Bausteine\Ueberwachung_Konfiguration\Konzept_Ueberwachung-Konfiguration.pdf .

8. Abbildungsverzeichnis

9. Weiterführende Themen

9.1. Migration von log4j auf isy-logging

Das Logging-Konzept wurde im Vergleich zur Register Factory Version 1.4 komplett überarbeitet. Dabei wurde insbesondere das Logging-Framework log4j durch logback ersetzt. Die Migration eines Systems auf das neue Logging-Konzept sollte in zwei Stufen erfolgen:

- **Technische Migration**: In einem ersten Schritt muss der Austausch von log4j mit logback erfolgen. Dies ist sehr einfach möglich, da mit der Bibliothek `log4j-over-slf4j` eine *Bridge* bereitgestellt wird, die es ermöglicht log4j zu ersetzen, ohne den Anwendungscode anzupassen. Die Bibliothek stellt dazu die gleichen Schnittstellen (im gleichen Pfad) wie log4j bereit und leitet die Aufrufe an SLF4J weiter. Für einen Austausch von log4j sind folgende Schritte notwendig:
  - `log4j*.jar` aus der Anwendung entfernen. Falls in den referenzierten Bibliotheken Abhängigkeiten auf log4j enthalten sind, bietet es sich an, eine Maven-Dependency auf log4j im Scope „provided“ zu ergänzen.
  - `isy-logging` auf die aktuelle Version aktualisieren (isy-logging ersetzen).
  - `log4j-over-slf4j*.jar` als Abhängigkeit ergänzen. Zu beachten ist, dass die Version dieser Bridge mit der Version der SLF4J-Api, die von isy-logging genutzt wird, übereinstimmt.
  - logback gemäß Abschnitt 4.2.1 konfigurieren.

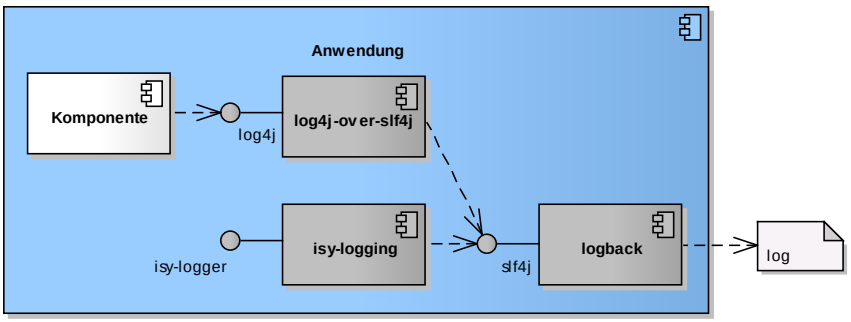

Abbildung 2: Zustand der Anwendung nach technischer Migration

- **Fachliche Migration**: Die Fachliche Migration, also das Erstellen von Logeinträgen gemäß diesen Nutzungsvorgaben, geht mit der Umstellung des Loggings auf den `IsyLogger` einher. Dies ist eine aufwändigere Änderung und sollte nur durchgeführt werden, wenn für die Anwendung konkreter Bedarf besteht, das Logging anzupassen.

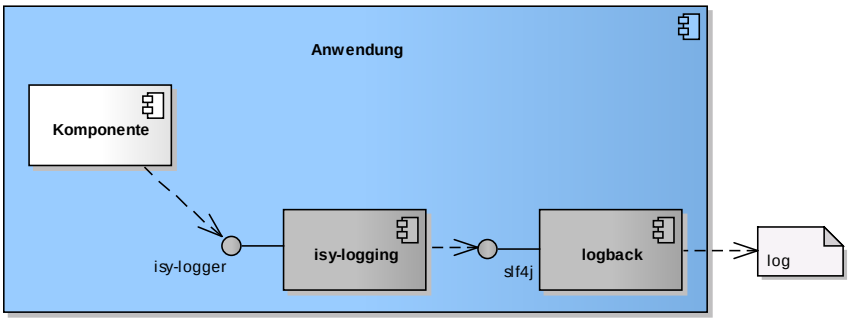

Abbildung 3: Zustand der Anwendung nach Abschluss der Migration

9.2. Log4j-Bridge

Durch den Wechsel in der IsyFact von logback auf log4j kann es vorkommen, dass in Altsystemen Bibliotheken eingebunden werden müssen, die bereits auf isy-logging umgestellt wurden, die Anwendung jedoch zunächst weiterhin log4j nutzt (insbesondere bei „Hotfixes“).

Zu diesem Zweck wird eine vollwertige SLF4J-Bridge bereitgestellt, die Anfragen an SLF4J auf log4j umleitet. Diese ermöglicht es insbesondere die Anfragen an Isy-Logging einheitlich auf log4j umzuleiten, so dass sich das Logging der Altanwendung unverändert verhält.

Zur Nutzung der Bridge sind folgende Anpassungen in der pom.xml der Anwendung notwendig:

- Sämtliche Abhängigkeiten zur SLF4J-API, Isy-Logging und anderen SLF4J-Implementierung der Anwendung durch entsprechende „excludes“ entfernen.

- Abhängigkeit zur isy-logging-log4j-bridge ergänzen.

  Bitte beachten: Die Bridge besitzt ihrerseits Abhängigkeiten auf isy-logging und die slf4j-api, die jedoch nicht „excluded“ werden dürfen. Durch das Vorgehen wird sichergestellt, dass nur die für die Bridge notwendigen Bibliotheken in der richtigen Version eingebunden werden.

Darüber hinaus sind keine weiteren Anpassungen notwendig.