

GKV-Kernprüfer

Handbuch zum
Test-Treiber

Version 0.93

Inhaltsverzeichnis

Einführung	3
1 Installation	4
1.1 Systemvoraussetzungen.....	4
1.2 Installation.....	4
1.2.1 Auslieferungsformat.....	4
1.2.2 Installationsprozess	4
1.2.3 Prüfen der Installation.....	4
1.3 Konfiguration	5
1.3.1 Konfigurationsdatei.....	5
1.3.2 Verwendbare Kernprüfer	6
1.3.3 Hinzufügen von Kernprüfern	6
2 Verwendung des Kernprüfer-Testers	10
2.1 Programmfenster	10
2.1.1 Menübalken und Toolbar.....	10
2.1.2 Testdaten- und Resultatsview.....	11
2.1.3 Statusbalken.....	11
2.2 Batch-Betrieb	12
2.3 Testdatenformat	12
3 Workflow	13
3.1 Laden einer Testdatei.....	13
3.2 Aufruf des Kernprüfers	14
3.3 Anzeige der Prüfergebnisse	15
3.3.1 Anzeige der Fehlercodes.....	15
3.3.2 Anzeige der Fehlertexte	16
3.3.3 Anzeige der Fehlerstrings	18
3.3.4 Detaillierte Ergebnisansicht	18
3.3.5 Weitere Funktionalitäten	19
3.4 Bearbeiten der Testdaten	19
3.4.1 Editieren	19
3.4.2 Erneute Prüfung.....	19
3.4.3 Speichern der Testdaten	19
3.5 Speichern der Prüfergebnisse	20
3.5.1 Format 1: Testfile mit Ergebnissen	20
3.5.2 Format 2: Ergebnisfile	21
4 Anhang I	22
4.1 Toolbar-Buttons	22
4.2 Ergebnisformatierung	23

Einführung

Beginnend mit dem Verfahren ZAHLS (Zahlstellen-Meldungen) wurde ab 2009 im System der GKV begonnen, einheitliche Kernprüfprogramme für einzelne Meldeverfahren der Sozialversicherung zu programmieren.

Als Programmiersprache wurde Java gewählt, die zu erstellenden Kernprüfprogramme sollten als nicht allein ablauffähige Bibliotheken zur Verfügung gestellt werden. Jedes Kernprüfprogramm muss eine definierte Schnittstelle implementieren.

Im Verlauf der Entwicklungsarbeit kristallisierte sich heraus, das

- Unterschiede in den Implementierungen zu leichten Inkompatibilitäten der verschiedenen Kernprüfprogramme führten,
- ein einheitliches Test-Vorgehen nicht erreicht wurde
- eine wiederverwendbare technische Unterstützung beim Test nicht existierte
- Aus diesem Grunde wurde beschlossen, den bereits existierenden Test-Treiber der AOK-Systems
- so umzuschreiben, dass die Möglichkeit der Verwendung mit allen anderen GKV-Kernprüfern gegeben ist
- seine Funktionalität so zu erweitern, dass bekannte technische Inkompatibilitäten automatisch erkannt und gemeldet werden.
- Sowohl Batch- als auch manuellen Betrieb zu unterstützen

Dieses Ziel wurde mit dieser Version des vereinheitlichen Testtreibers erreicht.

Das vorliegende Dokument beschreibt Installation, Konfiguration und Benutzung des Testtreibers, der von der AOK-Systems als Hilfsmittel bei der Entwicklung des GKV-Kernprüfers für das Verfahren EEL erstellt wurde.

1 Installation

1.1 Systemvoraussetzungen

Zur Ausführung des Kernprüfer-Testtreibers sind folgende Systemvoraussetzungen einzuhalten:

- Betriebssystem Windows 7 oder höher, Linux
- Java-Laufzeitumgebung (JDK), mindestens Version 11

Die Ablauffähigkeit unter anderen Betriebssystemen kann nicht garantiert werden. Die Software kann unter 32-bit als auch unter 64-bit Versionen der Java-VM eingesetzt werden. Ob eine JRE vorhanden ist, kann z.B. auf der Kommandozeile mit dem Befehl

```
java -version
```

geprüft werden, hier muss dann eine Ausgabe analog

```
openjdk version "11.0.7" 2020-04-14 LTS
OpenJDK Runtime Environment 18.9 (build 11.0.7+10-LTS)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10-LTS, mixed mode)
```

erscheinen.

1.2 Installation

1.2.1 Auslieferungsformat

Der Kernprüfer-Tester wird als Archivdatei im ZIP-Format ausgeliefert. Der Name der ZIP-Datei lautet KernprueferTester_xx.yy.zip, wobei xx.yy die Versionsnummer bezeichnet.

Das Layout des ausgelieferten Archivs sieht wie folgt aus:

```
/---+--- run.sh
+--- run.bat
+--- Handbuch.pdf
+--- KernprueferTester.jar
+--- lib ---+--- Kernpruefung.jar
+--- EELKernpruef.jar
```

Je nach Version befinden sich im Unterverzeichnis lib noch weitere Dateien, die für den Ablauf der Software benötigt werden.

1.2.2 Installationsprozess

Die ausgelieferte ZIP-Datei ist in ein beliebiges Verzeichnis auf dem Zielrechner zu entpacken.

Die oben aufgeführten Dateien sind dann im Zielverzeichnis der Extraktion vorhanden.

1.2.3 Prüfen der Installation

Nach dem Auspacken des Installationsarchives kann geprüft werden, ob der Tester prinzipiell funktioniert.

Dazu kann auf Windows-Systemen die Startdatei run.cmd verwendet werden; für Linux-Systeme ist run.sh zu verwenden.

Alternativ kann das Programm auf der Kommandozeile aus dem Installationsverzeichnis heraus mit dem Befehl

```
java -jar KernprueferTester.jar
```

gestartet werden.

Sollte hierüber oder über einen Doppelklick auf die KernprueferTester.jar ein Start fehlschlagen, sollte in der Windows Registry folgender Schlüssel überprüft werden:

Computer\HKEY_CLASSES_ROOT\jarfile\shell\open\command

Hier sollte für den Schlüssel (***Standard***) der Pfad für die Java 11 Installation gesetzt sein.

1.3 Konfiguration

1.3.1 Konfigurationsdatei

Das vorliegende Programm liest und speichert einige Einstellungen in einer Konfigurationsdatei. Der Name der Konfigurationsdatei lautet

```
config.properties
```

Sie muss sich in dem Verzeichnis befinden, aus welchem das Programm gestartet wird. Kann das Treiberprogramm keine Konfigurationsdatei öffnen, wird eine Datei mit Standardwerten angelegt. Das Format der Konfigurationsdatei entspricht dem in [1] beschriebenen Zeilen-Format für Konfigurationsdateien der Klasse `java.util.Properties`. Diese Dateien bestehen aus beliebig vielen Zeilen (getrennt durch CR, LF oder CRLF) mit je einem Schlüssel/Wert-Paar. Leer- und Kommentarzeilen (Zeilen beginnend mit „#“ oder „!“) werden ignoriert. Folgende Schlüsselwerte werden erkannt:

```
ACTVERF  
LASTDIR  
RESMODE  
IMPL.n  
LIB.n  
VERF.n  
PARM1.n  
PARM2.n
```

ACTVERF speichert den zuletzt verwendeten Kernprüfer, LASTDIR das zuletzt verwendete Arbeitsverzeichnis und RESMODE den zuletzt verwendeten Anzeigemodus. Die letzten drei Schlüsselwerte identifizieren die zu verwendeten Kernprüfer; für jedes zu verwendende Kernprüfprogramm sind jeweils alle 3 Schlüsselwerte anzugeben. Der Zusammenhalt der Schlüsselwerte zu einem Kernprüfer wird durch den Zähler `n` hergestellt. Es gehören also die Einträge IMPL.1, LIB.1 und VERF.1 zum ersten zu verwendenden Kernprüfer, IMPL.2, LIB.2 und VERF.2 zum zweiten usw. usf.

Die Nummerierung muss nicht lückenlos erfolgen und kann führende Nullen enthalten.
Weitere Einzelheiten zur Bedeutung der einzelnen Werte werden unten beschrieben.

1.3.2 Verwendbare Kernprüfer

Der vorliegende Kernprüfer-Tester ist in der Lage, beliebige GKV-Kernprüfprogramme zu treiben.
Voraussetzung dafür ist, dass ein zu verwendendes Kernprüfprogramm das vereinbarte Interface

kernpruefung.Kernpruefung

implementiert. Ein zu verwendendes Kernprüfprogramm muss weiterhin als Java-Bibliothek (jar-File) vorliegen. Der Name der implementierenden Klasse muss bekannt sein. Die Klasse muss zudem einen Default-Konstruktor besitzen. Diese Informationen können normalerweise aus der Dokumentation der einzelnen Kernprüfprogramme entnommen werden.

1.3.3 Hinzufügen von Kernprüfern

Zum Hinzufügen eines neuen Kernprüfers ist die Konfigurationsdatei anzupassen.
Es sind dazu pro Kernprüfer 3 neue Schlüssel/Werte-Paare einzufügen und zwar mit den Schlüsseln

IMPL.n
LIB.n
VERF.n

Dabei ist n durch eine Nummer zu ersetzen, die in der Konfigurationsdatei noch nicht verwendet wird.

Alle 3 neuen Schlüssel müssen die gleiche Nummer tragen.

Verfahrensname

Hinter dem Schlüssel VERF.n ist der Name anzugeben, unter dem der neue Kernprüfer im Kernprüfer-Tester angezeigt werden soll.

Beispiel:

VERF.7=ZAHLS 1.11.1

Der Wert kann frei gewählt werden, muss aber eindeutig sein, d.h. der angegebene Wert (hier „ZAHLS 1.11.1“) darf in der Konfigurationsdatei nicht mehrfach vorkommen.

Pfad zur Bibliothek

Hinter dem Schlüssel LIB.n ist der Pfad anzugeben, unter dem der neue Kernprüfer zu finden ist.
Es sind sowohl absolute als auch relative Pfade zulässig. Relative Pfade beziehen sich auf das Verzeichnis, aus dem das Programm gestartet wurde.

Es ist zu beachten, dass der Pfadtrenner unter Windows-Systemen mit dem Escape-Zeichen „\“ zu versehen ist und Pfade unter Linux-Systemen Groß/Kleinschreibung unterscheiden.
Beispiel für einen relativen Linux-Pfad:

LIB.7=lib/KernprueferEEL.jar

Beispiel für einen absoluten Windows-Pfad:

LIB.7=d:\\Kernpruefertester\\lib\\KernprueferEEL.jar

Man beachte die Escape-Zeichen „\“ vor den Pfadtrennern (ebenfalls „\“).

Klassenname

Hinter dem Schlüssel IMPL.n ist schließlich der Name derjenigen Klasse anzugeben, die den Kernprüfer implementiert. Diese Information ist der Dokumentation der jeweiligen Kernprüfprogramme zu entnehmen.

Es ist zu beachten, dass der vollständige Name der Klasse einschließlich des Packages anzugeben ist.

Beispiel:

IMPL.7=de.aoksystems.da.kernpruefer.eel.KernprueferEEL

Sollten alle drei Parameter korrekt angegeben worden sein, sollte der neue Kernprüfer im Kernprüfer-Tester unter dem bei VERF.n angegebenen Namen auswähl- und verwendbar sein.

Ist dies nicht der Fall, kann über die Logger-Ausgabe auf der Kommandozeile bzw. die Logger-Konsole die Fehlerursache ermittelt werden.

Typische Fehler umfassen:

- Kann Bibliothek nicht finden: ... : Falsche Pfadangabe unter LIB.n
- Kann Klasse nicht finden:... : Falscher Klassenname unter IMPL.n
- Reflection-Fehler bei ...: Die unter IMPL.n angegebene Klasse ist kein Kernprüfer
- Konnte keine Instanz erzeugen von Klasse: ...: Die unter IMPL.n angegebene Klasse hat keinen Default-Konstruktor

Weitere Informationen zur Fehlersuche können i.A. dem ebenfalls mitgeloggten Stacktrace entnommen werden.

Kernprüfer, die nicht fehlerfrei erzeugt werden können, werden deaktiviert. Dies wird ebenfalls geloggt.

Wenn beim Start kein einziger funktionierender Kernprüfer eingebunden werden kann, wird der interne Testprüfer namens „TestPruefer“ verwendet. Dieser ist nur für interne Tests des Kernprüfer-Testers verwendbar und hat keinen weiteren praktischen Nutzen.

Alternative Konstruktoren

Im Normalfall versucht der KernprüferTester, von der ermittelten Kernprüfer-Klasse eine Instanz

zu erzeugen, indem er den Default-Konstruktor aufruft.

Es gibt allerdings seit V 0.93 die Möglichkeit, andere – parametrisierte – Konstruktoren zur Instantiierung zu verwenden.

Es sind dazu Konstruktoren mit einem oder zwei Parametern möglich. Verwendet werden einparametrische Konstruktoren, wenn zur entsprechenden Kernprüfung n ein Parameter PARM1.n angegeben ist. Zweiparametrische Konstruktoren erfordern zusätzlich die Angabe des zweiten Parameters PARM2.n. Es ist nicht möglich, allein PARM2 anzugeben. Sind derartige Parameter angegeben, wird der Default-Konstruktor nicht verwendet, sondern ein passender Konstruktor gesucht.

Die Syntax der Parameter ist wie folgt:

PARM1.n=TYP:VALUE

Für TYP können – analog den JVM-Typ-Signaturen in JNI – folgende Java-Typen angegeben werden:

B: Byte	C: Character	D: Double	F: Float	I: Integer
J: Long	S: Short	Z: Boolean	Lxxx: Referenz-Typen	

Array-Typen (z.B. [C] sind nicht erlaubt.

Werden Referenz-Typen verwendet, folgt auf das „L“ der Klassentyp in Typ-Signatur-Schreibweise.

Beispiel für eine String-Variable:

Ljava/lang/String

Der geamte Parameter könnte dann z.B. so aussehen

PARM1.9=Ljava/lang/String:01.00

Hier würde versucht, folgenden Konstruktor der Klasse „Klasse1“ aufzurufen:

Klasse1("01.00");

Bzw.:

PARM1.9=Ljava/lang/String:02.00
PARM2.9=Ztrue

Klasse1("02.00", true);

Bzw.:

PARM1.9=D3.1415926
PARM2.9=F0.1

Klasse1(3.1415926,0.1);

Es ist zu beachten, dass die Konstruktion scheitern kann, wenn die dem Typ folgenden Initialisierungswerte keinen zulässigen Wert des Datentyps beschreiben (z.B. D:xy – es gibt keinen Double mit der literalen Darstellung xy) oder nicht aus einem String konstruierbar sind.

2 Verwendung des Kernprüfer-Testers

2.1 Programmfenster

Nach dem Start des Kernprüfer-Testers erscheint das Programmfenster des Kernprüfer-Testers:

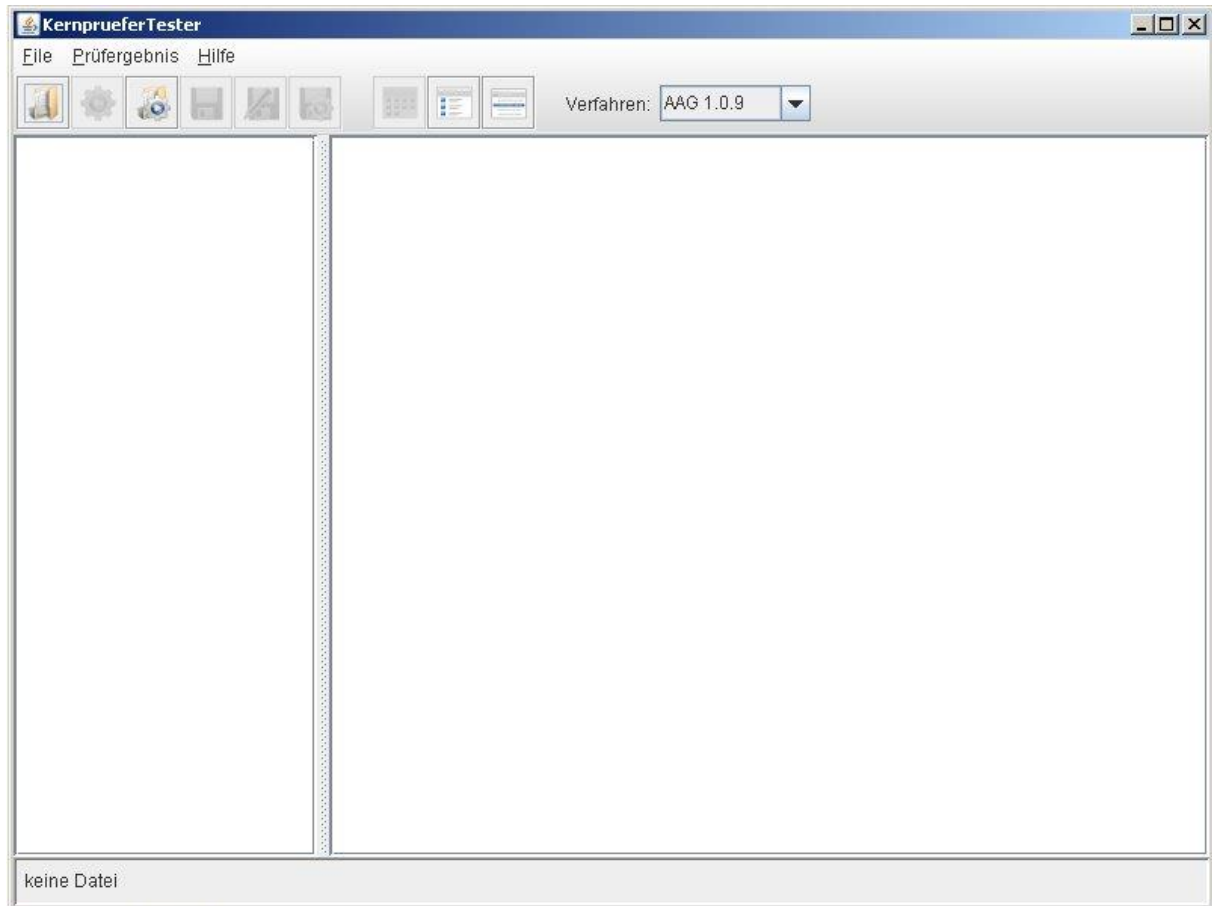


Abb.1: Startfenster

Das Programmfenster zerfällt in mehrere Teile:

- den Menübalken und die Toolbar
- die Resultats-View
- die Testdatenview
- den Statusbalken

2.1.1 Menübalken und Toolbar



Abb.2: Menübalken und Toolbox

2.2 Batch-Betrieb

Der Kernprüftester kann zusätzlich zum Online-Betrieb auch im Batchbetrieb laufen. Der Batchbetrieb wird immer dann automatisch benutzt, wenn Kommandozeilenargumente verwendet werden. Die zulässigen Kommandozeilenargumente sind im Folgenden aufgelistet:

```
-i infilenameName der Eingabedatei - Muss-Argument
-o outfilename      Name der Ausgabedatei - Kann-Argument
-v verfahren Name des Kernprüfers - Muss-Argument
-f1                erzeuge Format 1      - Kann-Argument
-f2                erzeuge Format 2      - Kann-Argument
```

Wird kein Ausgabedateiname angegeben, wird an den Namen der Eingabedatei „.result“ angehängt, um den Ausgabenamen zu erhalten.

Besteht die Ausgabedatei bereits, wird diese immer überschrieben.

Fehlen –f1 und –f2, wird –f1 als Ausgabeformat angenommen. –f1 beschreibt das Ausgabeformat, welches eine ausführliche Erklärung des Testlaufes enthält (im GUI-Modus unter „Testfile mit Ergebnis speichern“ zugänglich), f2 erzeugt ein „maschinenlesbares“ Ausgabeformat analog „Ergebnisfile speichern“

2.3 Testdatenformat

Das Format der zu prüfenden Testdaten ist wie folgt festgelegt:

- Textdatei in der Zeichenkodierung iso-8859-1/-15 (alternativ Windows-Codepage-1252)
- Pro Zeile ein Satz, als Zeilentrenner gelten CR, LF oder CRLF
- Zeilen mit „#“ in Stelle 1 werden als Kommentar angesehen und nicht geprüft
- Zeile mit „VOSZ“ am Satzanfang werden als Vorlaufsatz zwischengespeichert
- Zeilen mit „NCSZ“ am Satzanfang werden nicht geprüft
- Zeilen mit „DS“ oder „BW“ an Zeilenanfang werden als zu prüfender Satz angesehen und mit dem letzten gelesenen Vorlaufsatz an den gewählten Kernprüfer übergeben
- Alle andern Zeilen werden als „unbekannter Satz“ angesehen und nicht geprüft

3 Workflow

Der normalerweise anzuwendende Workflow bei der Verwendung des Kernprüfertesters sieht wie folgt aus:

- Laden einer Testdatei
- Aufruf des Kernprüfers
- Inspektion der Ergebnisse
- Speichern der Ergebnisse

Sind Testdaten geladen, können diese jederzeit manuell geändert und erneut geprüft werden. Geänderte Testdaten können ebenso als Datei gespeichert werden wie die Testergebnisse. Die einzelnen Arbeitsschritte werden im Folgenden näher erläutert:

3.1 Laden einer Testdatei

Um Testdaten überhaupt prüfen zu können, muss zuerst eine Testdatei geladen werden. Dazu ist der Menüpunkt File/Testfile öffnen... aufzurufen oder der Toolbar-Button für „Testfile öffnen“ zu drücken. (siehe Anhang I)

Danach erscheint ein Dateidialog, der es erlaubt, eine beliebige Datei auszuwählen:

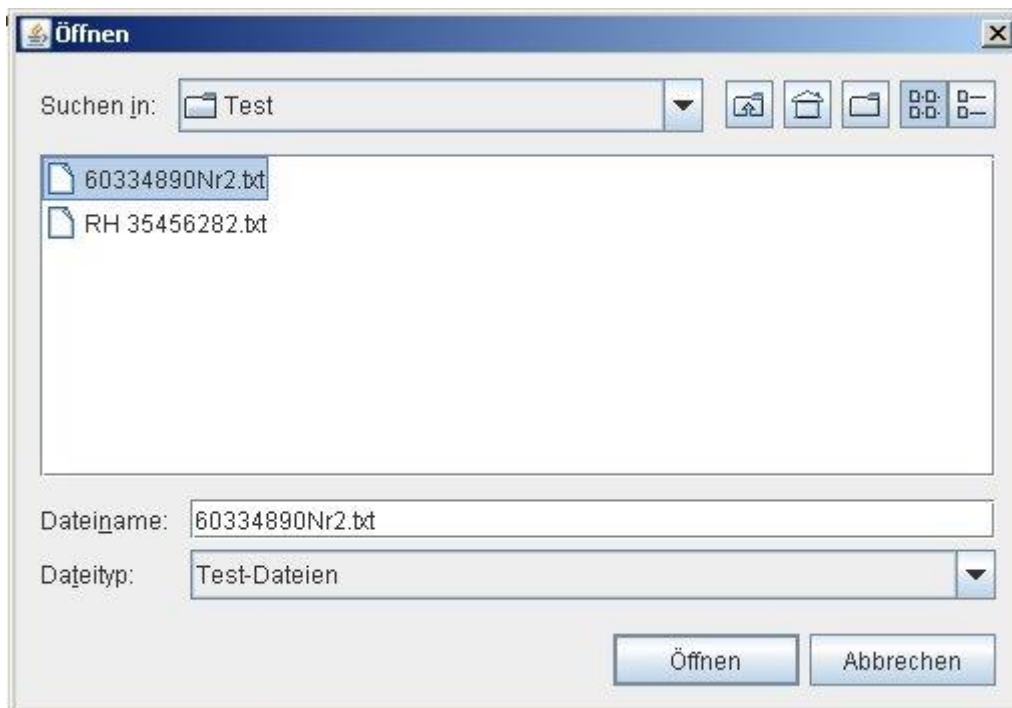


Abb.5: Testdatei öffnen

Als Dateityp ist „Testdateien“ voreingestellt, so dass nur Dateien mit den Endungen „txt“ und „dat“ angezeigt werden. Sollen auch Dateien mit einer anderen Endung angezeigt werden, ist hier „Alle Dateien“ auszuwählen.

Mit dem angebotenen Dateidialog kann auf dem Filesystem beliebig navigiert werden, um die zu öffnende Datei auszuwählen. Danach ist mit „Öffnen“ die Datei zu öffnen. Der Inhalt wird in den Editor der Testdatenview geladen, in der Statuszeile wird der vollständige Pfad der Testdatei angezeigt.

Zugleich werden mit dem erfolgreichen Laden einer Testdatei einige bisher ausgegraute Menüeinträge und Toolbar-Buttons aktiviert.

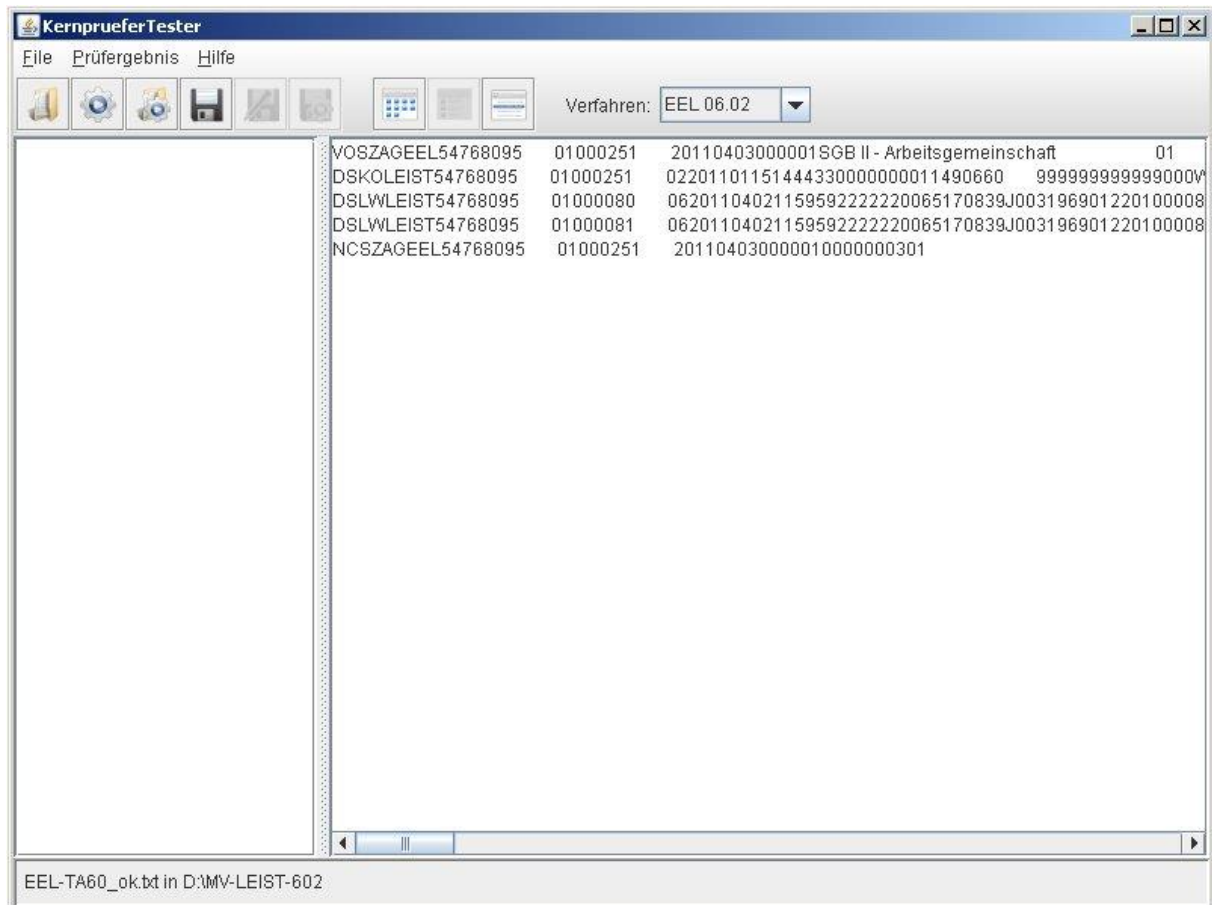


Abb.6: Kernprüftester mit geöffneter Testdatei

Sollte das Öffnen einer angewählten Datei nicht möglich sein, wird dies dem Benutzer durch einen Dialog angezeigt. In diesem Fall ist die Ursache der Lesesperre (z.B. ein Berechtigungsproblem) vor einem neuen Versuch zu beseitigen.

3.2 Aufruf des Kernprüfers

Sobald eine Testdatei geladen ist, können diese mit einem der installierten Kernprüfer getestet werden. Dazu ist der Menüpunkt File/Testfile prüfen... aufzurufen oder der Toolbar-Button für „Testfile prüfen“ zu drücken. (siehe Anhang I).

Im Ergebnis der Prüfung erscheint in der Resultatsview pro Eingabezeile eine Resultats-Zeile. Der Kernprüfer kann auf geladenen Testdaten beliebig oft aufgerufen werden. Werden Testdaten manuell im Editor geändert, werden beim nächsten Aufruf des Kernprüfers auch diese geänderten

Testdaten geprüft.

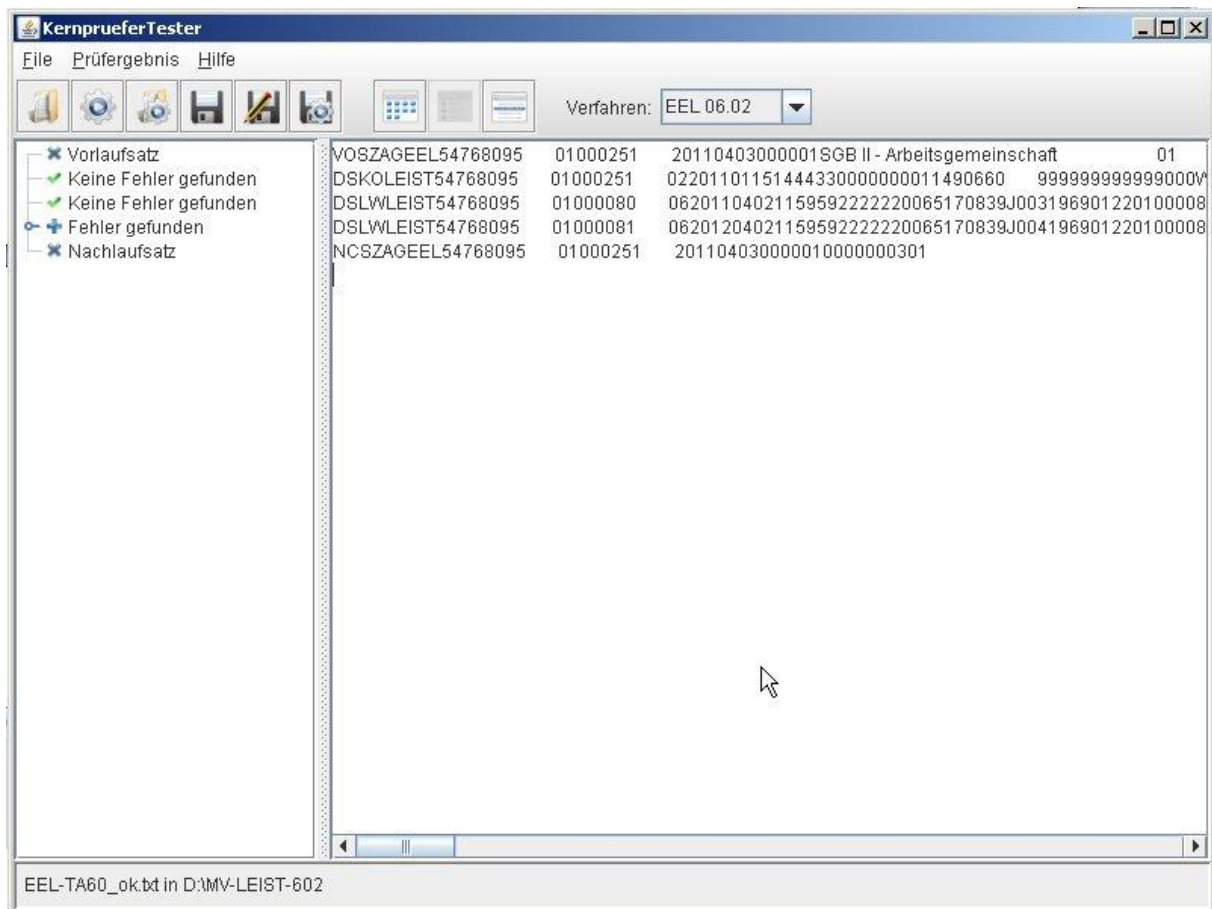


Abb.7: Kernprüftester mit geöffneter und geprüfter Testdatei

3.3 Anzeige der Prüfergebnisse

Die Anzeige der Prüfergebnisse kann in mehreren Formaten erfolgen.

- Fehlercodes
- Fehlertexte
- Gesamten Fehlerstring

Zudem kann eine detailliertere Kurzübersicht für jeden Fehler abgerufen werden.

3.3.1 Anzeige der Fehlercodes

Werden die Fehler im Modus „Fehlercodes“ angezeigt, dann werden in der Resultatsview pro Zeile nach der Ikone für den Gesamtstatus des Satzes alle gefundenen Fehlercodes aufgelistet. Als Fehlercodes gelten die 7-stelligen Fehlernummern der einzelnen Fehler lt. Satzbeschreibung.

Um diesen Anzeigemodus zu aktivieren, ist das Menü „Prüfergebnis/Nur Fehlercodes zeigen“ oder der entsprechende Toolbar-Button „Nur Fehlercodes zeigen“ anzuwählen.

Der gerade aktive Anzeigemodus ist daran zu erkennen, dass das entsprechende Menü und der Toolbar-Button ausgegraut sind.



Abb.8: Fehlercode-Anzeige

- Kommentare, Vor- und Nachlaufsätze werden mit der Ikone „nicht geprüft“ (siehe Anhang) versehen
- Fehlerfreie Sätze werden mit der Ikone „Testdaten ok“ versehen, dazu kommt der Text „Keine Fehler gefunden“
- Traten Hinweise und/oder Fehler auf, wird dies mit der Ikone „Testdaten hatten Fehler“ vermerkt. Nach der Ikone folgt eine Liste der Fehlernummern. In der obigen Abbildung sind das für Satz Nr. 4 die Hinweise DSTPH11, DSTPH21 und DSTPH31, für den 5. Satz die Fehler DSTP001, DSTP002 und DSTP003 sowie für Satz Nr. 6 die Fehler DSTP001 und DSTP002 sowie der Hinweis DSTPH31
- Traten technische Fehler auf, ist dies an der Ikone erkennbar. Auch hier wird eine Auflistung aller Fehler angefügt. Es reicht bereits ein technischer Fehler, damit die Zeile entsprechend markiert wird. Als technische Fehler werden in der Rückgabe eines Kernprüfers erkannt:
 - fehlendes „DBFE“ vor dem Fehlerbaustein und
 - Falsche Länge des Fehlerbausteins (Muss = 76 mit dem DBFE und trailing spaces)
- Schlussendlich wird die Ikone verwendet, um einen Abbruch des Kernprüfers zu signalisieren. Dies deutet entweder auf komplett korrupte Eingabedaten, einen falsch ausgewählten Kernprüfer oder einen schweren Programmfehler im Kernprüfer hin

3.3.2 Anzeige der Fehlertexte

Werden die Fehler im Modus „Fehlertexte“ angezeigt, dann werden in der Resultatsview pro Zeile nach der Ikone für den Gesamtstatus des Satzes (siehe Fehlercode-Anzeige) die Texte


- „Kommentar“, „Vorlaufsatz“ bzw. „Nachlaufsatz“ für die Fälle, die nicht geprüft wurden
- „Keine Fehler gefunden“, wenn der Satz fehlerfrei war
- „Hinweise gefunden“, wenn (nur) Hinweise gefunden wurden
- „Fehler gefunden“, wenn nur Fehler gefunden wurden

- „Fehler und Hinweise gefunden“, wenn beide Arten Fehlerbausteine gefunden wurden und
- Abbruch, wenn ein Abbruch des Kernprüfers auftrat

Die Fehler und Hinweis-Texte können bei beiden Ikonen  und  auftreten, je nachdem, ob die vom Kernprüfer gelieferten Fehlerbausteine technisch korrekt oder fehlerhaft waren.



Abb.9: Fehlertexte-Anzeige

Wurden vom Kernprüfer Fehler oder Hinweise gefunden, so werden diese im Anzeigemodus „Fehlertexte“ mit einer Baumansicht dargestellt. Diese kann durch einen Mausklick auf das Zeichen  aufgeklappt werden. Danach werden die einzelnen Fehlerbausteine detailliert angezeigt.

Achtung: Im aufgeklappten Zustand besteht keine 1:1-Beziehung mehr zwischen der Testdaten- und der Resultatsview.

Zu sehen ist dieses Verhalten in der folgenden Abbildung für den Satz 6:

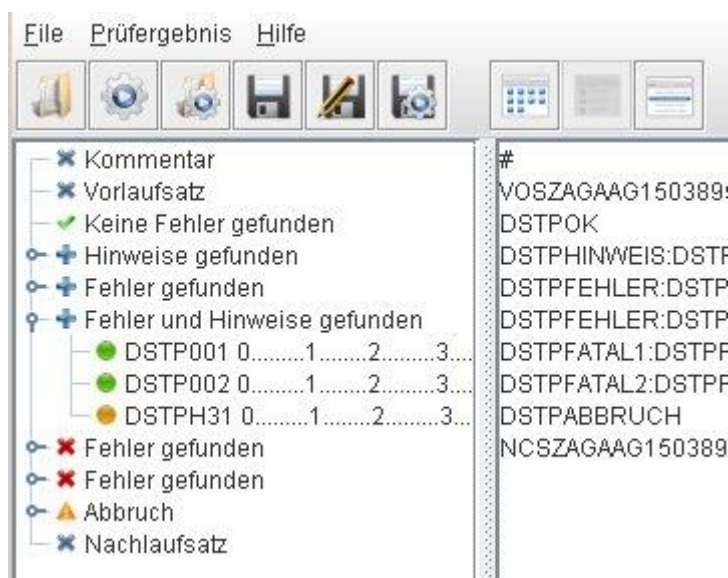





Abb.10: ausgeklappte Fehlertexte-Anzeige

Zu den einzelnen Fehlerbausteinen werden dann in der aufgeklappten Baumansicht angezeigt:

- Die Ikone , wenn es ein technisch korrekter Fehlerbaustein ist
- Die Ikone , wenn es ein technisch korrekter Hinweisbaustein oder eine Abbruchmeldung ist
- Die Ikone , wenn es ein technisch falscher Fehler- oder Hinweisbaustein ist

3.3.3 Anzeige der Fehlerstrings

Werden die Fehler im Modus „Fehlerstrings“ angezeigt, so entspricht die Anzeige im wesentlichen der Anzeige der Fehlercodes mit der Ausnahme, dass die kompletten DBFE-Bausteine angezeigt werden, so wie sie am Ende des Datensatzes anzuhängen wären, würden die Sätze an den Absender zurückgemeldet.



Abb.11: Fehlerstring-Anzeige

3.3.4 Detaillierte Ergebnisansicht

Fährt man mit der Maus über eine Zeile der Resultatsview und verharret dort, so werden in einer Tooltip-Anzeige detaillierte Informationen zum darunter liegenden Prüfergebnis angezeigt:

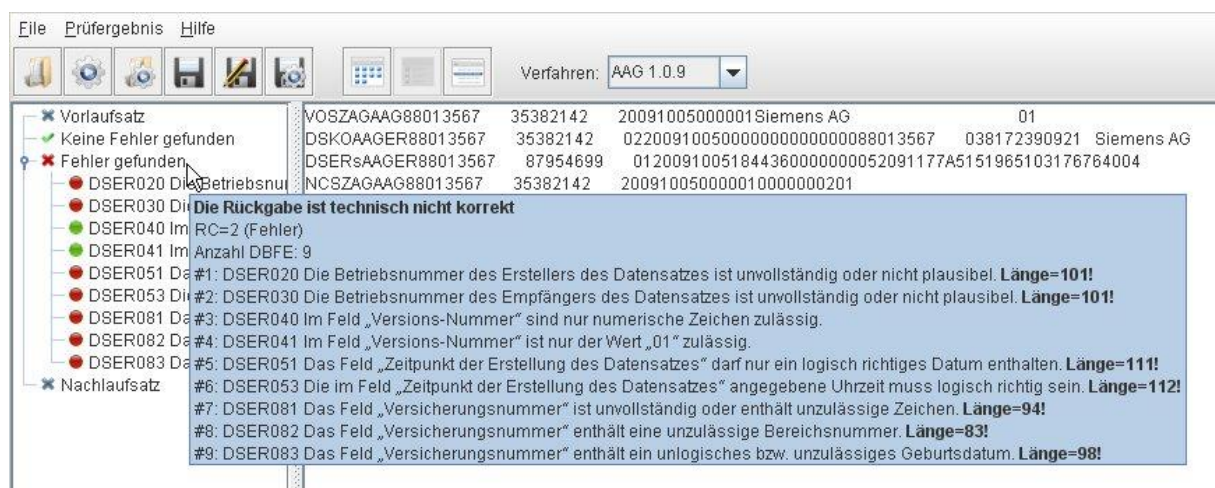


Abb.12: Tooltip-Anzeige

Angezeigt werden neben dem Returncode (RC) des Kernprüfers die Anzahl der Fehler, die einzelnen Fehlertexte sowie das Prüfergebn der technischen Prüfung (fett).

3.3.5 Weitere Funktionalitäten

Es ist in allen Anzeigemodi möglich, Teile der Ergebnisansicht zu markieren und in die Zwischenablage zu kopieren.

3.4 Bearbeiten der Testdaten

3.4.1 Editieren

Die geladenen Testdaten können jederzeit im Editor (Testdatenview) geändert werden.

3.4.2 Erneute Prüfung

Durch ein Drücken des Buttons „Testfile prüfen“ wird der gewählte Kernprüfer gegen die geänderten Testdaten ausgeführt. Zu beachten ist, das Testresultat hier das Testergebnis gegen die Daten im Editor darstellt und nicht mehr das Testresultat gegen die Daten auf dem Filesystem.

3.4.3 Speichern der Testdaten

Geänderte Testdaten können jederzeit in eine Datei gespeichert werden. Dazu ist der Menüpunkt File/Testfile speichern... aufzurufen oder der Toolbar-Button für „Testfile speichern“ zu drücken. Danach erscheint ein Dateidialog, der es erlaubt, eine beliebige Datei auszuwählen:

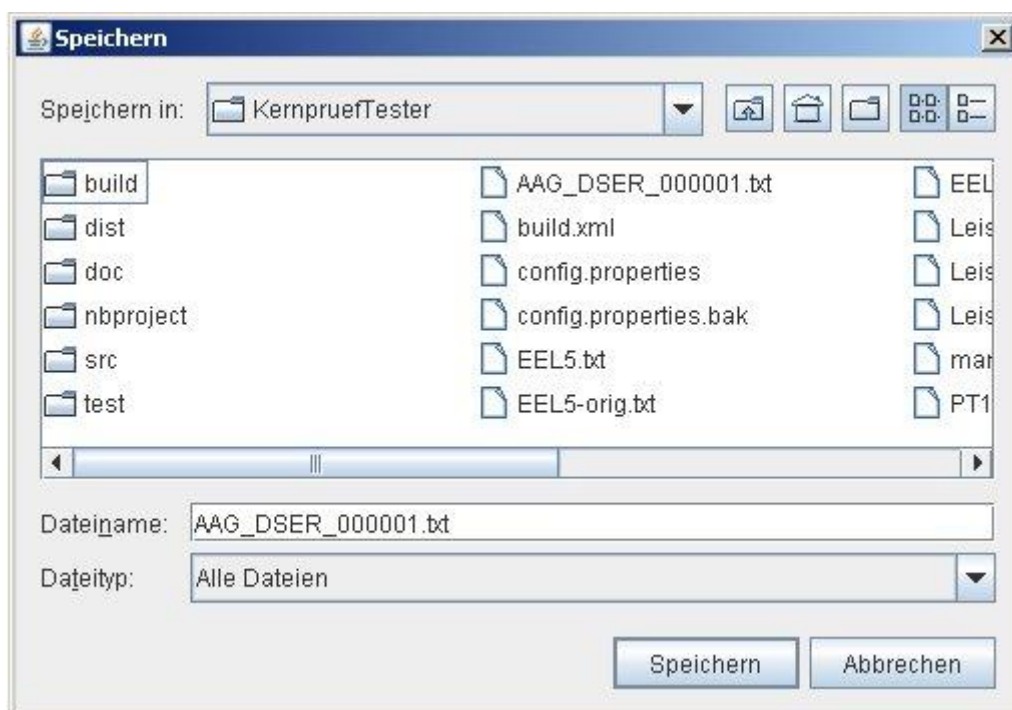


Abb.13: Testdatei speichern

Hier ist der Speicherort und der Name der zu speichernden Datei anzugeben. Voreingestellt ist Name und Pfad der bereits geladenen Datei.

Sollte die Datei bereits existieren, ist das Überschreiben im folgenden Dialog zu bestätigen:

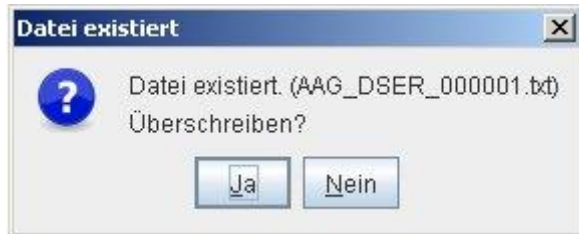


Abb.14: Sicherheitsabfrage beim Speichern

3.5 Speichern der Prüfergebnisse

Die erstellten Prüfergebnisse können in zwei Formaten auf eine Datei gespeichert werden:

- Testfile mit Ergebnissen
- Ergebnisfile

Während das erste Format hauptsächlich für manuelle Prüf- und Dokumentationszwecke geeignet ist, kann das zweite Format zur Erzeugung maschinell nutzbarer Daten genutzt werden.

Das Speichern der Prüfergebnisse entspricht im Ablauf dem Speichern der Testdatei, mit dem Unterschied, dass in der Voreinstellung dem Namen die Zeichenketten „TestMitResultat-“, bzw. „Resultat-“, vorangestellt werden. Die Voreinstellung kann durch den Nutzer immer manuell abgeändert werden.

3.5.1 Format 1: Testfile mit Ergebnissen

Das Format „Testfile mit Ergebnissen“ sieht wie folgt aus:

```
#
=====
Testfile erstellt am Mi Sep 21 21:15:59 MESZ 2011
aus File AAG_DSER_000001.txt
Kernpruefer: AAG 1.0.9
=====
Eingabe:
Zeile 0: VOSZAGAAG88013567      35382142      20091005000001Siemens AG
Zeile 1: DSKOAGER88013567      35382142      02200910050000000000000088013567 . . .
Zeile 2: DSERsAAGER88013567    87954699      012009100518443600000000052091177 . . .
Zeile 3: NCSZAGAAG88013567      35382142      2009100500000100000000201
=====
Ergebnisse:
Zeile 0: Vorlaufsatz
Zeile 1: RC=0: (Keine Fehler)
Zeile 2: RC=2: DSER020 DSER030 DSER040 DSER041 DSER051 DSER053 DSER081 DSER082 DSER083
Zeile 3: Nachlaufsatz
=====
```











Wie zu erkennen ist, folgt auf den Dateikopf, der allgemeine statistische Daten zum Testlauf enthält, die geprüfte Eingabe (hier abgeschnitten) und zum Schluss das Prüfergebnis.

3.5.2 Format 2: Ergebnisfile









Das Format „Ergebnisfile“ wird hier nicht angeführt. Es besteht aus den Eingabesätzen, an die an das Ende alle ermittelten DBFE-Bausteine angehängt werden. Das Feld FEKZ (Stelle 62) wird mit „1“ belegt, FEAN (Stelle 63) mit der Anzahl der ermittelten Fehler. Korrekte Sätze bleiben unangetastet.

4 Anhang I

4.1 Toolbar-Buttons

Toolbar-Element	Bedeutung	Erläuterung
	Testdatei öffnen	Öffnet eine Textdatei und lädt sie in den Editor
	Testdaten prüfen	Prüft die Daten, die im Editor vorhanden sind
	Testdatei öffnen und Testdaten prüfen	Öffnet eine Textdatei und lädt sie in den Editor, danach werden die geladenen Daten geprüft
	Testdatei speichern	Speichert die Daten des Editors auf eine Datei
	Testdatei einschließlich Testergebnisse speichern	Speichert die Testdaten des Editors samt dem Testergebnis in aufbereiteter Form
	Ergebnisdatei speichern	Speichert die Testdaten des Editors samt den gefundenen Fehlern in maschinenlesbarer Form
	Nur Fehlercodes zeigen	Zeigt nur die Fehlercodes (7-stellige Fehlernummer) als Kurzliste
	Fehlertexte zeigen	Zeigt nur die kompletten Fehlertexte (7-stellige Fehlernummer und Fehlertexte) als lange Liste
	Gesamten Fehlerstring zeigen	Zeigt den String aus allen Fehlerbausteinen, wie er an den jeweiligen Satz angehängt würde
	Verfahrens-Auswahl	Wählt den zu verwendenden Kernprüfer aus

4.2 Ergebnisformatierung

Format-Ikone	Bedeutung	Erläuterung
	Nicht geprüft	Der Satz wurde nicht geprüft (z.B. VOSZ/NCSZ)
	Testdaten ok	Die Testdaten hatten keinen Fehler
	Abbruch - Warnung	Die Testdaten wurden nicht geprüft, der Kernprüfer wurde mit RC 4 abgebrochen
	techn. Fehler	Beim Prüfen trat ein technischer Fehler auf (z.B. Fehlerbaustein hat falsche Länge)
	Testdaten hatten Fehler	Die Testdaten hatten Fehler, die Fehlermeldungen waren technisch ok
	Fehlerbaustein ist ok	Der so geflaggte Fehlerbaustein ist technisch ok (Länge 76 Zeichen, beginnt mit „DBFE“)
	DBFE ist ein Hinweis	Der so geflaggte Fehlerbaustein ist ein Hinweis und technisch ok (s.o.)
	DBFE ist techn. falsch	Der so geflaggte Fehlerbaustein ist techn. falsch (Länge <> 76, beginnt nicht mit DBFE)