

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Theoretische Informatik

FLEXITRUST CA

**Ein flexibles, skalierbares und dynamisch
erweiterbares Trustcenter**

Diplomarbeit von ALEXANDER WIESMAIER

15. Februar 2001

angefertigt bei PROF. DR. J. BUCHMANN
betreut von MARKUS RUPPERT

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt 15. Februar 2001

Alexander Wiesmaier
wiesmaie@cdc.informatik.tu-darmstadt.de

Die in diesem Dokument erwähnten Soft- und Hardwarebezeichnungen sind in den meisten Fällen auch eingetragene Warenzeichen und unterliegen als solche den gesetzlichen Bestimmungen.

Vorwort

Die in dieser Arbeit beschriebene Certification–Authority (CA) ist die Neuauflage der von Markus Tak im Rahmen seiner Diplomarbeit angefertigten Trustcenter–Software. Einige Teile des vorliegenden Dokumentes, vor allem, was die Motivation und die Beschreibung der Randbedingungen angeht, sind seiner Ausarbeitung¹ entlehnt. Ähnlich verhält es sich mit Teilen der Software. Wo dies möglich war, habe ich auf den von Ihm entworfenen Code zurückgegriffen.

Zusätzlich zu der in Markus Taks Version vorhandenen flexiblen Auswahl an kryptographischen Algorithmen und den zugehörigen Providern, ist die neue Version skalierbar und dynamisch erweiterbar. Da die Software noch immer weiterentwickelt wird, ist diese Arbeit als ein dokumentarischer Schnappschuß zu verstehen, der den grundsätzlichen Entwurf der Software aufzeigen soll. Für einen genauen Einblick in den derzeitigen Stand der Entwicklung steht die Dokumentation des Quellcodes im Javadoc–Format zur Verfügung.

Ein Prototyp der Software wurde in Zusammenarbeit mit der Firma KOBIL Systems GmbH bereits praktisch eingesetzt. Die dabei gewonnenen Erfahrungen und Rückmeldungen waren besonders interessant für die weitere Entwicklung des Projektes.

Besonderer Dank gilt Markus Tak und Markus Ruppert, die bei Problemen stets einen kühlen Kopf bewahren und gute Ratschläge zur Hand haben, sowie Professor Buchmann, der das Projekt tatkräftig unterstützt und immer ein offenes Ohr hat.

¹[Tak99]

Inhaltsverzeichnis

1	Einleitung	8
1.1	Leseanweisung	8
1.1.1	Konventionen	8
1.1.2	Einteilung	8
1.2	Public–Key–Infrastrukturen	9
1.2.1	Public–Key–Kryptographie	9
1.2.2	Certification–Authority	10
2	Analyse	12
2.1	Istzustand	12
2.2	Sollzustand	12
2.3	Vorüberlegungen	14
2.3.1	Flexibilität	14
2.3.2	Skalierbarkeit	14
2.3.3	Erweiterbarkeit	15
2.3.4	Fusion	15
2.3.5	Abstraktion	15
2.3.6	Vorgehensweise	15
3	Advanced Processing Infrastructure	16
3.1	Design	16
3.1.1	Processable	16
3.1.2	Processor	18
3.1.3	Queue	18
3.1.4	Importer	19
3.1.5	Exporter	19
3.1.6	Dormitory	19
3.1.7	Condition	20
3.1.8	Stock	20
3.1.9	Logger	20
3.1.10	Synchronizable	21
3.1.11	Monitor	21
3.1.12	Kompression	21

3.2	Implementierung	21
3.2.1	Allgemeines	21
3.2.2	de.tud.cdc.flexiTrust.ca.api	22
4	FlexiTRUST	27
4.1	Design	27
4.1.1	Impliziertes Design	27
4.1.2	Weiteres Design	28
4.1.3	Aktuelles Design	29
4.1.4	Schnittstellen	30
4.2	Implementierung	30
4.2.1	Allgemeines	30
4.2.2	de.tud.cdc.flexiTrust.ca.app	31
4.2.3	de.tud.cdc.flexiTrust.ca.init	36
4.2.4	de.tud.cdc.flexiTrust.ca.log	39
4.2.5	de.tud.cdc.flexiTrust.ca.pro	41
4.2.6	de.tud.cdc.flexiTrust.ca.util	44
5	Betrieb	47
5.1	Eigenschaften	47
5.2	Installation	48
5.2.1	Systemanforderungen	48
5.2.2	Installationsvorgang	48
5.3	Initialphase	49
5.3.1	Issuer	49
5.3.2	Administrator	49
5.4	Alltäglicher Betrieb	49
5.4.1	Start	49
5.4.2	Betrieb	50
5.4.3	Beendigung	51
5.5	Das Makefile	52
5.5.1	Variablen	52
5.5.2	Targets	52
5.6	Die Properties	55
5.6.1	Global	55
5.6.2	CaKeys	55
5.6.3	AdminKeys	56
5.6.4	Certification–Authority	56
5.6.5	Administrator	57
5.6.6	Logging	57
5.7	Erweiterung	57
5.8	Testsysteme	58

6	Ausblick	60
A	Javadocs	62
A.1	de.tud.cdc.flexiTrust.cs.api	63
A.2	de.tud.cdc.flexiTrust.ca.app	69
A.3	de.tud.cdc.flexiTrust.ca.init	82
A.4	de.tud.cdc.flexiTrust.ca.log	84
A.5	de.tud.cdc.flexiTrust.ca.pro	89
A.6	de.tud.cdc.flexiTrust.ca.util	93
	Literaturverzeichnis	97
	Index	102

Abbildungsverzeichnis

2.1	Aufgabenstellung	13
3.1	Design der Advanced Processing Infrastructure	17
3.2	Implementierung der Advanced Processing Infrastructure	23
4.1	Das Application–Package	32
4.2	Das Initiation–Package	37
4.3	Das Logging–Package	40
4.4	Das Processables–Package	42
4.5	Das Utilities–Package	45
A.1	Übersicht über alle Packages	62

Kapitel 1

Einleitung

1.1 Leseanweisung

1.1.1 Konventionen

Die in diesem Dokument abgebildeten Paket- und Klassen-Diagramme sind in UML-Notation¹. Für die verwendeten Schriftarten gilt folgende Konvention:

- **Fettgedrucktes** wird direkt nach seinem Vorkommen erklärt.
- *Kursives* deutet auf etwas im Zusammenhang besonders Wichtiges hin.
- Schreibmaschinenschrift wird für Paket-, Klassen-, Feld- und Methodennamen verwendet.

1.1.2 Einteilung

Es folgt ein kurzer Überblick über den groben Aufbau dieses Schriftstücks. Der Leser kann hieran entscheiden welche Kapitel für ihn von Interesse sind.

Kapitel 1. Einleitung: Das gerade aufgeschlagene Kapitel. Es enthält neben dieser Leseanweisung ein paar einführende Worte zur Public-Key-Kryptographie und zu den Aufgaben eines Trustcenters.

Kapitel 2. Analyse: Zuerst beleuchte ich kurz die Ausgangssituation, die zu Beginn der Diplomarbeit herrschte (Istzustand). Danach wird die aktuelle Aufgabenstellung beschrieben (Sollzustand). Schließlich beschäftige ich mich mit einigen Vorüberlegungen zum Design der Software.

Kapitel 3. Advanced Processing Infrastructure: Dieses Kapitel erklärt das Design und die Implementierung der Advanced Processing Infrastructure.

¹[FS98]

Kapitel 4. FlexiTRUST: Hier wird das Design und die Implementierung der Flexi-TRUST Certification-Authority erklärt.

Kapitel 5. Betrieb: Behandelt den Umgang mit der Applikation. Wichtige Punkte bezüglich des Betriebes der Software werden erläutert.

Kapitel 6. Ausblick: Alles was angedacht, aber noch nicht umgesetzt ist, wird in diesem Kapitel angesprochen.

Anhang A. Javadocs: Hier befinden sich die Javadocs zu allen Klassen. Sie bieten die Möglichkeit genaue Implementierungsdetails einzusehen.

1.2 Public-Key-Infrastrukturen

Die in dieser Ausarbeitung beschriebene Applikation ist der direkte Nachfolger von Markus Taks Trustcenter-Software. In seiner Ausarbeitung² gibt er eine ausführliche Einführung in Public-Key-Infrastrukturen (PKI). Ich empfehle Details dort nachzulesen. In diesem Abschnitt gebe ich nur einen groben Überblick darüber.

1.2.1 Public-Key-Kryptographie

Bei der Public-Key-Kryptographie besitzt jeder Benutzer ein Schlüsselpaar. Der eine Schlüssel heißt öffentlicher Schlüssel (public key) und ist jedem zugänglich. Der andere heißt privater Schlüssel (private key) und ist nur dem Besitzer zugänglich. Diese beiden Schlüssel besitzen folgende Eigenschaften:

- 1: Der private Schlüssel kann praktisch³ nicht aus dem öffentlichen Schlüssel berechnet werden.
- 2: Ein mit dem öffentlichen Schlüssel verschlüsselter Text ist praktisch nur mit dem privaten Schlüssel zu entschlüsseln.

Will nun Alice eine verschlüsselte Nachricht an Bob schicken, so verschlüsselt sie ihren Text mit Bobs öffentlichem Schlüssel. Die Nachricht kann nun nur von Bob mit seinem privaten Schlüssel entschlüsselt werden. Entsprechend verschlüsselt Bob seine Nachrichten an Alice mit deren öffentlichem Schlüssel.

Zur Erzeugung von Signaturen sind die Rollen von öffentlichem und privatem Schlüssel vertauscht. Der private Schlüssel wird zum Verschlüsseln benutzt, der öffentliche zum Entschlüsseln. Die Eigenschaft 1 bleibt unverändert, Eigenschaft 2 lautet hier:

²[Tak99]

³praktisch = mit vertretbarem Aufwand

1 EINLEITUNG

2': Ein mit dem öffentlichen Schlüssel entschlüsselbarer Cyphertext⁴ ist praktisch nur mit dem privaten Schlüssel erzeugbar.

Will Alice eine Nachricht signieren, so verschlüsselt sie diese⁵ mit ihrem privaten Schlüssel. Um diese Signatur zu verifizieren testet Bob, ob er den verschlüsselten Text⁶ mit dem öffentlichen Schlüssel von Alice entschlüsseln kann. Gelingt dies, so ist die Signatur gültig. Entsprechend signiert Bob seine Nachrichten an Alice mit seinem privaten Schlüssel.

Es ist leicht einzusehen, daß Public–Key–Kryptographie nur funktioniert, wenn der private Schlüssel wirklich nur dem rechtmäßigen Besitzer zugänglich ist, und der öffentliche Schlüssel dem rechtmäßigen Besitzer zuverlässig zugeordnet werden kann. Der Schutz des privaten Schlüssels kann durch Passworte oder Chipkarten gewährleistet werden. Die Zuordnung des öffentlichen Schlüssels zu seinem rechtmäßigen Besitzer kann durch geeignete Public–Key–Infrastrukturen, speziell durch Certification–Authorities, erreicht werden. Diese werden nachfolgend vorgestellt.

1.2.2 Certification–Authority

Certification–Authorities (auch Trustcenter genannt) gewährleisten die sichere Zuordnung von öffentlichen Schlüsseln zu ihren rechtmäßigen Besitzern. Wichtigste Voraussetzung ist, daß die Benutzer ihrer CA (Certification–Authority) vertrauen. Die CA beglaubigt die Zuordnung eines öffentlichen Schlüssels zu einem Teilnehmer in einem Zertifikat, das von ihr digital signiert, und damit vor Veränderung geschützt ist. Indem sich verschiedene CAs gegenseitig zertifizieren (Crosszertifizierung), können sich Benutzer unterschiedlicher Trustcenter gegenseitig auf die Zuverlässigkeit ihrer öffentlichen Schlüssel verlassen. Ein Trustcenter muß nachfolgend aufgezählte Teilaufgaben bewältigen. Die Reihenfolge dieser Aufzählung entspricht auch der zeitlichen Reihenfolge, in der diese Aufgaben bearbeitet werden.⁷

Registrierung: Benutzer, die ihren öffentlichen Schlüssel zertifizieren lassen wollen, müssen erfaßt und ihre Identität muß überprüft werden. Bringt der Teilnehmer seinen öffentlichen Schlüssel mit, muß sichergestellt werden, daß dies auch wirklich sein Schlüssel ist. Zur Vermeidung von Namensgleichheiten bekommt jeder Teilnehmer eine eindeutige Kennung.

Schlüsselerzeugung: Ist kein Schlüssel vorhanden, so muß ein neues Schlüsselpaar erzeugt werden. Dazu werden verschiedene Angaben des Benutzers (Algorithmus, Schlüssellänge, ...) benötigt.

⁴mit bekanntem Klartext

⁵bzw. deren Hashwert, um die Signatur kurz zu halten

⁶dessen Klartext er bereits kennt

⁷der Zeitstempeldienst steht in keiner zeitlichen Abhängigkeit zu den anderen Aufgaben

Zertifizierung: Das Zertifikat wird ausgestellt. Es enthält unter anderem den Gültigkeitszeitraum, den Namen, die Kennung und den öffentlichen Schlüssel des Teilnehmers.

Personalisierung: Wurde ein privater Schlüssel erzeugt, so muß dieser auf ein Geheimnisträger aufgebracht, und bei der CA gelöscht werden. Außerdem können weitere Informationen über den Teilnehmer gespeichert und Sicherheitssperren⁸ eingerichtet werden.

Verzeichnisdienst: Um die Zertifikate der Öffentlichkeit zugänglich zu machen, muß ein Verzeichnisdienst geführt werden. Dort können gültige Zertifikate und Revokationslisten heruntergeladen werden.

Revokation: Verliert ein Teilnehmer seinen Geheimnisträger, oder wurde sein privater Schlüssel auf andere Weise korrumpiert, so kann er die Widerrufung seines Zertifikats beantragen. Die Revokation kann aber auch auf Initiative der CA erfolgen.

Zeitstempeldienst: Um Daten einem bestimmten Zeitpunkt zuordnen zu können, werden die an diesen Dienst übergebene Daten mit einem signierten Zeitstempel versehen.

Zusätzlich sind eine Reihe weiterer Sicherheitsmaßnahmen notwendig. So muß der Bereich, in denen die Schlüssel der CA vorgehalten bzw. die Benutzerschlüssel erzeugt werden (die Kern-CA) baulich von der Außenwelt getrennt werden, und die dort aufgestellten Rechner nicht ans öffentliche Netzwerk angeschlossen sein. Weiterhin muß eine niedergeschriebene Sicherheitspolitik existieren, in der Zugangsrechte und Zertifizierungsabläufe genauestens festgeschrieben sind. Natürlich müssen alle Aktionen der CA geloggt und bestimmte Produkttypen (Zertifikate, Revokationen, ...) dauerhaft gespeichert werden.

⁸z.B: Passwörter

Kapitel 2

Analyse

2.1 Istzustand

Die bei Beginn dieser Arbeit vorhandene Version von FlexiTRUST ist das Ergebnis einer stetigen Weiterentwicklung von Markus Taks Diplomarbeit. Deren Zielsetzung war die Entwicklung eines modularen Trustcenters. Es wurde Wert auf eine skalierbare Registrierung und den dynamischen Austausch kryptographischer Algorithmen gelegt. Es sollte gezeigt werden, daß es möglich ist ein kryptographisch flexibles Trustcenter zu schaffen. Das Signaturgesetz¹ sollte bedacht, aber die Konformität dazu nicht erzwungen werden. Als Programmiersprache wurde Java gewählt.

2.2 Sollzustand

Aufgabe dieser Diplomarbeit ist die Schaffung einer völlig neuen Version von FlexiTRUST, die neben der flexiblen Auswahl der kryptographischen Algorithmen (und deren Providern) auch Skalierbarkeit und dynamische Erweiterbarkeit der kompletten Applikation ermöglichen soll. Die Anforderungen an eine CA unterliegen starken Schwankungen. Beispielsweise liegt die Belastung eines Trustcenters bei Neueinführung in einem Unternehmen oder bei Hinzunahme einer neuen Abteilung um ein Vielfaches über dem normalen Level. Um dies ausgleichen zu können muß die CA skalierbar sein. Da auch in Zukunft neue Sicherheitsstandards zu erwarten sind, ist es nötig, daß ein Trustcenter in der Lage ist diese zu adaptieren. Um die CA für diesen Vorgang nicht abschalten zu müssen, ist es notwendig sie dynamisch erweitern zu können.

Um ein vernünftiges Maß an Arbeit für eine Diplomarbeit nicht zu überschreiten, sollen nicht alle Teilaufgaben einer CA² implementiert werden. Vielmehr soll durch wohlüberlegtes Design eine solide Basis-CA geschaffen werden, die durch stetige

¹[fWuT97]

²siehe Abschnitt 1.2.2 auf Seite 10

dynamische Erweiterung zu einer vollständigen CA ausgebaut werden kann. Besonders die Registrierung wird gesondert als Praktikumsaufgabe³ bzw. Diplomarbeit behandelt.

Damit ergibt sich als aktuelle Aufgabenstellung eine CA zu erschaffen, die ein Systemverhalten hat, wie es in Abbildung 2.1 zu sehen ist. Die Anträge (Applications)⁴ können auf verschiedene Art (Disc, DB, ...) in den Zuständigkeitsbereich der Kern-CA⁵ (Certificator) gebracht werden. Innerhalb des Kernbereiches werden die Anträge auf Gültigkeit überprüft und bearbeitet. Während der Bearbeitung können vom Certificator Zufallszahlen (Random Numbers), Schlüssel (Keys) und weitere kryptographische Strukturen erzeugt werden. Es entstehen verschiedene Produkte (Certificate, CRL, ...), die wiederum auf unterschiedliche Weise aus der Kern-CA heraus gebracht werden können. Die zu empfehlende Transportmethode in bzw. aus dem Kernbereich der CA, ist die Übertragung der Daten mittels eines Wechseldatenträgers. Dies ermöglicht die beste Kontrolle über die ein- und ausgehenden Daten, und erhöht somit die Sicherheit der CA.

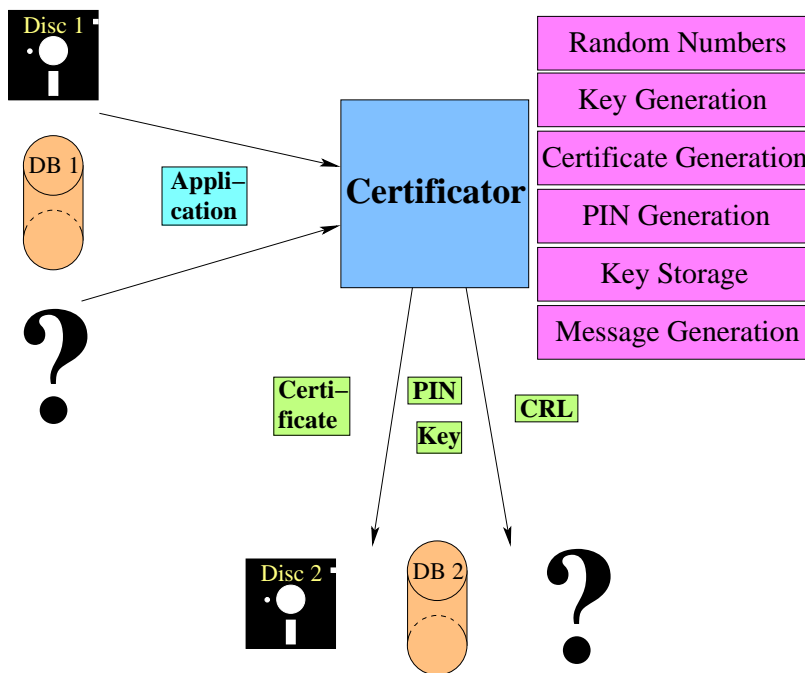


Abbildung 2.1: Aufgabenstellung

³[DS00]

⁴z.B. Zertifikatsantrag, Revokationsantrag, ...

⁵der sicherheitssensible, baulich getrennte Bereich der CA

Als Programmiersprache soll wieder Java⁶ verwendet werden, da diese einige entscheidende Vorteile bietet:

- Es entstehen plattformunabhängige Programme.
- Java bietet mit der Java Cryptographic Architecture (JCA) ein Framework zur flexiblen Verwendung verschiedener kryptographischer Algorithmen und Providern.
- Am hiesigen Fachbereich wurden eine Reihe von Java-Bibliotheken entwickelt (CODEC, SENECA, ...), die zur Implementierung eines Trustcenters überaus vorteilhaft sind.
- Die Java Database Connectivity (JDBC) bietet eine flexible, herstellerunabhängige Schnittstelle zum Zugriff auf Datenbanken.
- Der Vorgänger des neuen Trustcenters wurde in Java implementiert, und kann im Umgang mit vorgenannten APIs als Wegweiser dienen.

Die Konformität zum Signaturgesetz⁷ steht auch in dieser Arbeit nicht im Vordergrund. Es soll lediglich möglich sein, das zu entwickelnde Programm später so anzupassen, daß diese Konformität erreicht wird.

2.3 Vorüberlegungen

2.3.1 Flexibilität

Die Flexibilität in der Auswahl kryptographischer Algorithmen nebst Providern kann durch die Verwendung der Java Cryptographic Architecture (JCA) erreicht werden. Dies entspricht genau dem Verwendungszweck dieses APIs, außerdem wurde die Praxistauglichkeit dieser Lösung bereits in Markus Taks Arbeit⁸ gezeigt.

2.3.2 Skalierbarkeit

Die Skalierbarkeit eines Vorgangs erreicht man durch dessen Parallelisierung. Übertragen auf Software bedeutet dies die Verteilung der Rechenarbeit auf mehrere Computer innerhalb eines Netzwerkes. Diese Art der Verteilung ist aus der heutigen Praxis nicht mehr wegzudenken, und wird durch die Programmiersprache Java massiv unterstützt.

⁶[Mic00b]

⁷[fWuT97]

⁸[Tak99]

2.3.3 Erweiterbarkeit

Um die dynamische Erweiterbarkeit einer Applikation zu gewährleisten, muß diese modular aufgebaut sein. Außerdem muß sie über Schnittstellen verfügen, die allgemein genug sind, um auch die Implementierung von zur Designzeit noch unbekanntem Funktionen zu erlauben.

2.3.4 Fusion

Zur praktischen Umsetzung sowohl der Skalierbarkeit als auch der dynamischen Erweiterbarkeit muß die Applikation in kleinere Teilapplikationen zerlegt werden, die zusammen die gemeinsame Aufgabe lösen. Aufgrund dieses gemeinsamen Ansatzes (Aufteilung der Applikation) habe ich mich entschlossen, die Lösung dieser beiden Probleme (Skalierbarkeit und Erweiterbarkeit) nicht getrennt zu suchen, sondern zusammen anzugehen.

2.3.5 Abstraktion

Zur Realisierung der dynamischen Erweiterbarkeit werden sehr allgemeine Schnittstellen benötigt. Zusätzlich sind Skalierbarkeit und dynamische Erweiterbarkeit nicht nur für CA-Software sinnvoll, sondern können für beliebige Applikationen nützlich sein. Deshalb werde ich deren (gemeinsames) Design auf einer höheren Ebene, die von der konkreten Aufgabe einer Applikation abstrahiert, angehen.

2.3.6 Vorgehensweise

Für den weiteren Verlauf habe ich die Arbeit in zwei Schritte zerlegt. Zuerst werde ich ein allgemeines Framework entwickeln, welches mir die geforderten Eigenschaften an Skalierbarkeit und dynamischer Erweiterbarkeit liefert. Die Beschreibung des Designs und der Implementierung hiervon sind in Kapitel 3 ab Seite 16 zu finden. Danach werde ich mit Hilfe dieses Frameworks die Basis-CA implementieren. Design und Implementierung dazu sind in Kapitel 4 ab Seite 27 beschrieben.

Kapitel 3

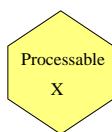
Advanced Processing Infrastructure

3.1 Design

Das von mir erdachte Konzept zur skalierbaren Verteilung einer Applikation mit der Möglichkeit der dynamischen Erweiterung habe ich *Advanced Processing Infrastructure* (API) genannt. Die Grundidee besteht darin, eine zu bewältigende Aufgabe nicht von außen zu zerlegen, sondern dieser Aufgabe zu gestatten sich selber zu verteilen. Jede Aufgabe wird als eigene Klasse implementiert, und wird nach dem Object–Flow–Prinzip abgearbeitet. Das heißt, die Umgebung stellt geeignete Arbeitsplätze zur Verfügung, und das Aufgabenobjekt sucht diese der Reihe nach auf, um seine Arbeit zu verrichten. Um Skalierbarkeit zu erreichen, können bei Bedarf neue Instanzen der Arbeitsplätze hinzugefügt werden, welche die bereits vorhandenen entlasten. Dies ermöglicht dann auch die dynamische Erweiterbarkeit, da ebenso völlig neuartige Arbeitsplätze hinzugefügt werden können.

Abbildung 3.1 auf Seite 17 zeigt eine schematische Darstellung der Advanced Processing Infrastructure. Die als Fläche dargestellten Objekte stellen die einzelnen Komponenten des Systems dar. Die Pfeile deuten den möglichen Weg eines Aufgabenobjektes durch das System an. Im folgenden werden die Funktionen der Komponenten des Systems näher erläutert. Die Beschreibung der Implementierung ist in Abschnitt 3.2 auf Seite 21 zu finden.

3.1.1 Processable



Die Aufgabenobjekte, die durch das System wandern um Ihre Arbeit zu verrichten, heißen Processables. Jede Processable–Instanz trägt die alleinige Verantwortung für die Korrektheit ihres Arbeitsablaufes bzw. eine korrekte Fehlerbehandlung. Um dem System eine völlig neue Aufgabe hinzuzufügen (d.h. es dynamisch zu erweitern), wird einfach ein neuer Processable–Typ implementiert, welcher die neue Arbeit verrichtet. Die bereits existierenden Processable–

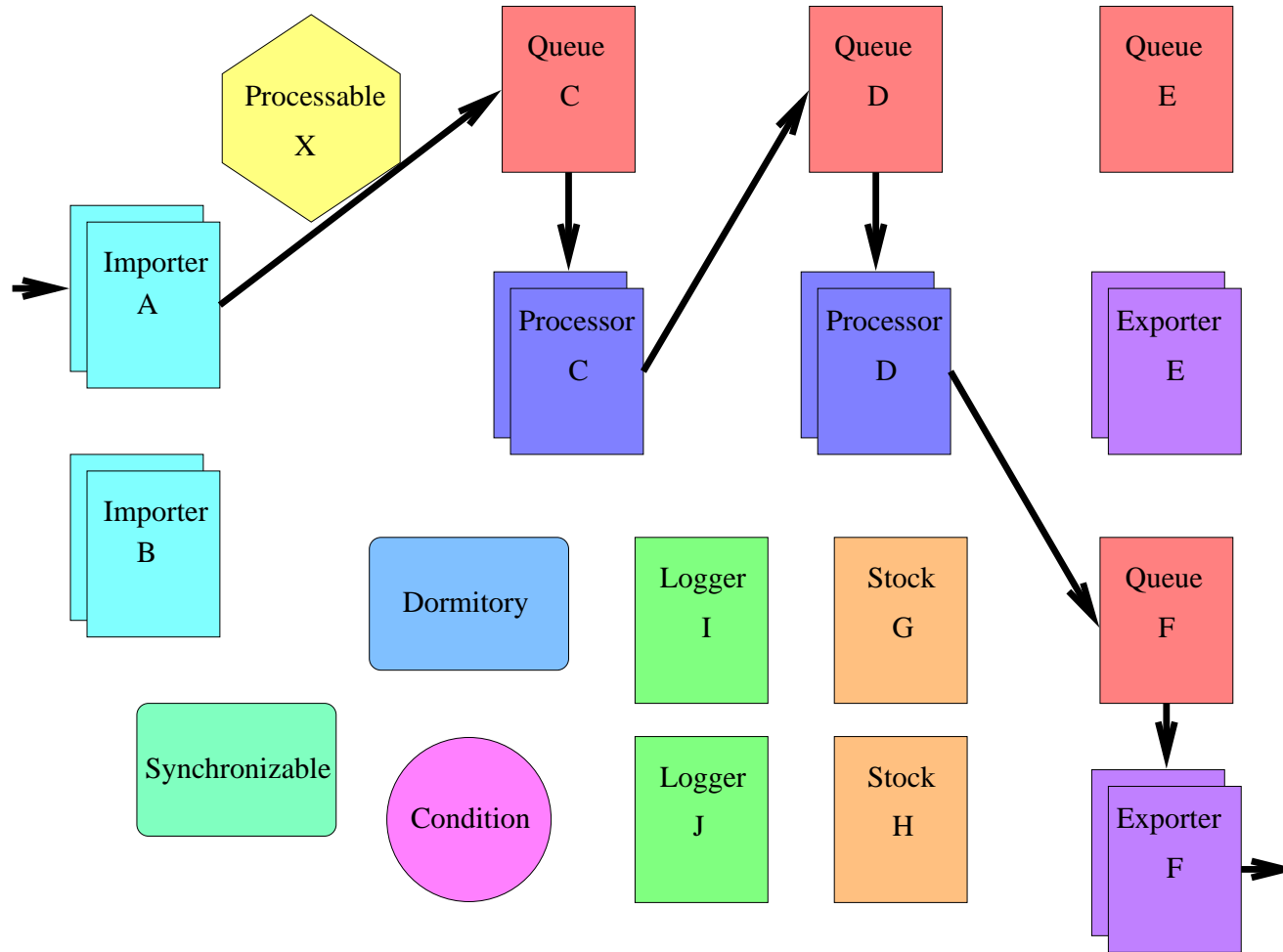
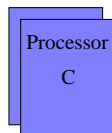


Abbildung 3.1: Design der Advanced Processing Infrastructure

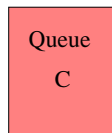
Typen können völlig unberührt davon bleiben. Es können beliebig viele gleichartige oder verschiedene Processable-Instanzen zur selben Zeit im System existieren.

3.1.2 Processor



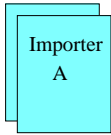
Processors dienen als die benötigten Arbeitsplätze. Hier verrichten die Processables ihre eigentliche Arbeit. Dazu stellen die Processors den Processables eine geeignete Umgebung und geeignetes Werkzeug zur Verfügung. Für alle Arbeiten die ein Processable zu verrichten hat müssen geeignete Arbeitsplätze durch Processors bereitgestellt werden. Wird das System um einen neuen Processable-Typ (eine neue Aufgabe) erweitert, so kann es nötig sein auch einen neuen Processor-Typ (einen neuen Arbeitsplatz) zu implementieren. Die bereits bestehenden Processor-Typen können davon völlig unberührt bleiben. Es können beliebig viele gleichartige oder verschiedene Processor-Instanzen zur selben Zeit im System existieren. Jedoch sollte immer mindestens eine Processor-Instanz von jedem Typ existieren, um den Processables alle Arbeitsplätze zu bieten.

3.1.3 Queue



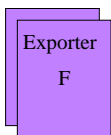
Um die Last gleichmäßig zu verteilen und ein dynamischen Kommen und Gehen von Processors zu ermöglichen existieren die Queues im System. Jedem Processor-Typ ist genau eine Queue zugeordnet. Statt direkt zu einem Processor zu wandern, reiht sich ein Processable in der diesem Processor-Typ zugeordneten Queue ein. Die Processors haben Zugriff auf ihre Queue und holen sich, wenn sie Kapazität frei, ein Processable aus der Queue. So wird die Rechenlast auf alle Processors gleichmäßig verteilt. Außerdem ist es möglich dem System zur Laufzeit neue Processor-Instanzen hinzuzufügen. Die neuen Instanzen bedienen sich einfach aus der ihnen zugeordneten Queue und entlasten so die bereits existierenden Processors. Umgekehrt ist es auch möglich Processor-Instanzen aus dem System zu entfernen. Ein zu beendender Processor holt sich kein neues Processable mehr aus der Queue, sondern quittiert sein Dienst, wenn ihn das letzte Processable verlassen hat. Das System ist also skalierbar. Reiht sich ein Processable in eine Queue ein, so bedeutet dies nicht notwendig, daß das Objekt wirklich dort hin wandert. Hält der aktuelle Processor entsprechende Vorkehrungen bereit, so bleibt das Processable bei diesem, und hinterlegt in der Queue eine Nachricht wo es zu finden ist. Ein anderer Processor, der ein Processable aus dieser Queue bearbeiten möchte, liest die Nachricht, und holt sich das Processable vom vorhergehenden Processor ab (Peer-To-Peer-Transport). Hierdurch werden Netzlast und Flaschenhalsbildung an der Queue deutlich gesenkt. Beendet ein Processor seinen Dienst, so steht er auch nicht mehr für dem Peer-To-Peer-Transport zur Verfügung. Er sorgt deshalb dafür, daß alle noch bei ihm vorhandenen Processables zu den entsprechenden Queues wandern. Zu jedem Processor-Typ muß immer genau eine Queue im System existieren.

3.1.4 Importer



Importer ermöglichen es Processables in das System zu bringen. Sie stehen konzeptuell auf der Grenze zwischen Außenwelt und Systeminneren, und dienen als Eingangsportal. Die initiale Station eines Processables ist immer ein Importer, von hier beginnt für das Processable seine Reise durch das System. Wird eine neue Art des Imports benötigt, so wird ein neuer Importer-Typ implementiert. Die bereits vorhandenen Importer-Typen können davon völlig unberührt bleiben. Es können beliebig viele gleichartige oder verschiedene Importer-Instanzen zur selben Zeit im System existieren. Jedoch sollte immer mindestens eine Importer-Instanz existieren, da sonst keine Möglichkeit besteht Processables in das System zu bringen.

3.1.5 Exporter



Damit die Processables nach Erfüllung ihrer Aufgabe das System auch wieder verlassen können, gibt es die Exporter. Genauso wie die Importer stehen sie konzeptuell auf der Grenze zwischen Außenwelt und Systeminneren. Sie dienen als Ausgangsportal. Die letzte Station eines Processables ist immer ein Exporter. Wie bei den Processors wandern die Processables nicht direkt zu den Exportern, sondern reihen sich in eine entsprechende Queue ein. Wird eine neue Art des Exports benötigt, so wird ein entsprechender Exporter-Typ implementiert. Die vorhandenen Exporter-Typen können davon unberührt bleiben. Es können beliebig viele gleichartige oder verschiedene Exporter-Instanzen zur selben Zeit im System existieren. Jedoch sollte immer mindestens eine Exporter-Instanz existieren, da sonst für die Processables keine Möglichkeit besteht das System zu verlassen.

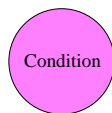
3.1.6 Dormitory



Es kann Situationen geben, in denen ein Processable durch äußere Umstände¹ aufgehalten wird. In diesem Fall begibt sich das Processable zum Dormitory und wartet dort, bis es mit seiner Aufgabe fortfahren kann. Auch hier führt der Weg zu einer konkreten Instanz über eine Queue. Die im Dormitory befindlichen Processables werden überwacht, und aufgefordert das System zu verlassen, wenn sie sich zu lange dort aufhalten. Es können beliebig viele gleichartige oder verschiedene Dormitory-Instanzen zur selben Zeit im System existieren. Jedoch sollte immer mindestens eine Dormitory-Instanz existieren, da sonst für die Processables keine Möglichkeit besteht in den Wartemodus zu gehen, und sie das System stattdessen mit unvollendeter Aufgabe verlassen müßten.

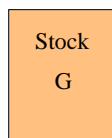
¹z.B: ein Drucker hat kein Papier mehr

3.1.7 Condition



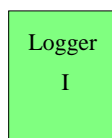
Mittels einer Condition teilt ein Processable dem Dormitory mit, auf welches Ereignis es wartet. Das Dormitory prüft in regelmäßigen Abständen die übergebene Condition. Trifft sie zu, wird das Processable geweckt, und kann seine Aufgabe fortführen. Bleibt die Condition zu lange unerfüllt, so initiiert das Dormitory den Export des wartenden Processables. Das Processable führt dann eventuell notwendige Aufräumarbeiten durch, und verläßt das System mit entsprechender Fehlermeldung.

3.1.8 Stock



Der Stock ist eine zentrale Stelle zur Verwaltung der Ressourcen,² die von der Applikation benötigt werden. Hier versorgen sich alle bisher genannten Komponenten mit den von ihnen benötigten Daten. Damit der Stock nicht zum Flaschenhals des Systems wird, kann man verschiedenen Ressourcen zu Gruppen zusammenfassen, und diese durch jeweils eigene Stock-Typen verwalten lassen. Außerdem sollten alle darauf zugreifenden Instanzen geeignete Cache-Mechanismen implementieren. Die Anzahl der im System vorhandenen Stock-Instanzen hängt also von der Gruppierung der zu verwaltenden Ressourcen ab. Kommt eine neue Aufgabe ins System, die neue Ressourcen benötigt, so kann ein neuer Stock-Typ eingeführt werden. Die vorhandenen Stock-Typen bleiben davon völlig unberührt. Es darf beliebig viele Stock-Typen im System geben. Aber von jedem Typ darf maximal eine Instanz vorhanden sein.

3.1.9 Logger



Mittels des Loggers wird ein zentrales Logging realisiert. Alle vorgenannten Komponenten können hier ihre Log-Nachrichten ablegen. Auch hier können verschiedene Gruppen von Nachrichten gebildet werden die durch jeweils eigene Logger-Typen³ erfaßt werden, um Flaschenhalsbildung zu vermeiden. Wiederum sind alle zugreifenden Instanzen gehalten geeignete Mechanismen zur Entlastung der Logger bereitzuhalten⁴. Es darf beliebig viele Logger-Typen im System geben. Aber von jedem Typ darf nur maximal eine Instanz vorhanden sein.

²beispielsweise die Schlüssel einer CA

³es könnte z.B. für jeden Log-Level einen eigenen Logger geben

⁴z.B. nur Nachrichten mit hinreichendem Log-Level verschicken

3.1.10 Synchronizable

Synchronizable

Um eine Möglichkeit zu haben dem kompletten System Änderungen anzuzeigen oder Steuerungsbefehle⁵ zu übermitteln existiert das Konzept der Synchronizables. Alle vorgenannten Komponenten sollten dieses Konzept implementieren. Synchronisationsnachrichten werden an alle Synchronizables übertragen. Die einzelne Instanz reagiert dann entsprechend.

3.1.11 Monitor

Um dem Benutzer die Möglichkeit zu geben mit dem System zu kommunizieren, werden Monitore verwendet. Diese können auf oben beschriebenen Komponenten aufgesetzt werden, und kontrollieren deren Arbeitsablauf. Im Normalfall wird ein Monitor eine herkömmliche GUI sein.

3.1.12 Kompression

Da an zentralen Stellen (Stock, Queue, ...) besonders viel Netzverkehr ist, können sie zum Flaschenhals des Systems werden. Um dem entgegenzuwirken empfiehlt sich die Kompression aller zu versendenden Objekte. Bei Queues ist diese Lösung uneingeschränkt tauglich, da die Processables dort auch in komprimiertem Zustand abgelegt werden können. Bei aktiven Komponenten (Stock, Processor, ...), in denen die Objekte allerdings unkomprimiert verarbeitet werden müssen, wird die Senkung der Netzlast durch die Erhöhung der Prozessorlast erkaufte. Hier muß also der richtige Kompromiß zwischen Verringerung des Verkehrs und Erhöhung der Rechenarbeit gefunden werden. Dieser hängt von der Bandbreite des vorliegenden Netzes und der Rechenleistung der Computer ab, und muß deshalb am konkreten System entschieden werden. Analoge Überlegungen gelten auch für die komprimierte Speicherung persistenter Daten.

3.2 Implementierung

3.2.1 Allgemeines

Da Java als Programmiersprache für die CA festgelegt wurde, wird natürlich auch die Advanced Processing Infrastructure darin implementiert. Aber auch ohne diese Vorgabe wäre Java die Sprache der Wahl, da sie geradezu ideale Konzepte zur Realisierung der Advanced Processing Infrastructure bietet:

Plattformunabhängigkeit: Da Rechnernetze im allgemeinen nicht homogen sind, ist es nötig die Darstellung von Objekten und deren Kommunikation plattformunabhängig zu halten. Java erfüllt diese Anforderung ohne weiteren programmier-technischen Aufwand.

⁵z.B. Herunterfahren der Applikation

3 ADVANCED PROCESSING INFRASTRUCTURE

Serialisierung: Um Objekte von einem Rechner zum anderen wandern zu lassen, ist es nötig sie linear⁶ darstellen zu können. Das Serializable-Konzept⁷ von Java dient genau diesem Zweck.

Remote Method Invocation (RMI): Objekte einer verteilten Applikation, die sich auf verschiedenen Rechnern befinden, müssen irgendwie miteinander kommunizieren. RMI⁸ bietet die Möglichkeit Methoden auf entfernten Objekten aufzurufen, und damit eine Kommunikation zu realisieren. Für die Parameterübergabe wird die eben erwähnte Serialisierung benutzt.

Java Intelligent Network Infrastructure (Jini): Damit entfernte Objekte miteinander arbeiten können, müssen sie Kenntnisse über die Aufenthaltsorte der jeweils anderen Objekte haben, und sich gegenseitig eindeutig identifizieren können. Diesem Zweck dient Jini.⁹ Es ermöglicht das dynamische Auffinden und Identifizieren von entfernten Objekten. Es ist möglich (und überaus sinnvoll) innerhalb eines Jini-Systems mittels RMI zu kommunizieren.

In Java gibt es keine Mehrfachvererbung. Deshalb habe ich mich entschlossen, alle im vorhergehenden Abschnitt aufgeführten Konzepte als Interfaces zu implementieren. Dies ermöglicht es dem späteren Benutzer des Frameworks das Vererbungskonzept ohne Einschränkung zu benutzen. Um den Umgang mit Jini allerdings einfacher zu gestalten, habe ich eine `JiniHelper`-Klasse implementiert, an die verschiedene, jini-spezifische Aufgaben delegiert werden können. Diese Implementierungsart hat den weiteren Vorteil, daß die Verteilung nicht an Jini gebunden ist, sondern später auch mittels anderer Konzepte¹⁰ realisiert werden kann.

3.2.2 `de.tud.cdc.flexiTrust.ca.api`

Die Klassen der Advanced Processing Infrastructure befinden sich alle im Package `de.tud.cdc.flexiTrust.ca.api`. Abbildung 3.2 auf Seite 23 zeigt die Struktur dieses Packages und aller darin enthaltene Klassen inklusive ihrer Felder und Methoden. Jedes der im Abschnitt Design vorgestellten Konzepte wird als eigenes Interface implementiert. Der Name eines Interfaces ist jeweils der selbe wie der des zugehörigen Konzeptes. Im folgenden werden die in diesem Package enthaltenen Klassen beschrieben. Genauere Implementierungsdetails dazu kann man den Javadocs im Anhang A ab Seite 63 entnehmen.

⁶z.B. als Byte-Array

⁷[Mic00b]

⁸[jGu00], [Mic00f]

⁹[Mic00d]

¹⁰man könnte beispielsweise CORBA nutzen

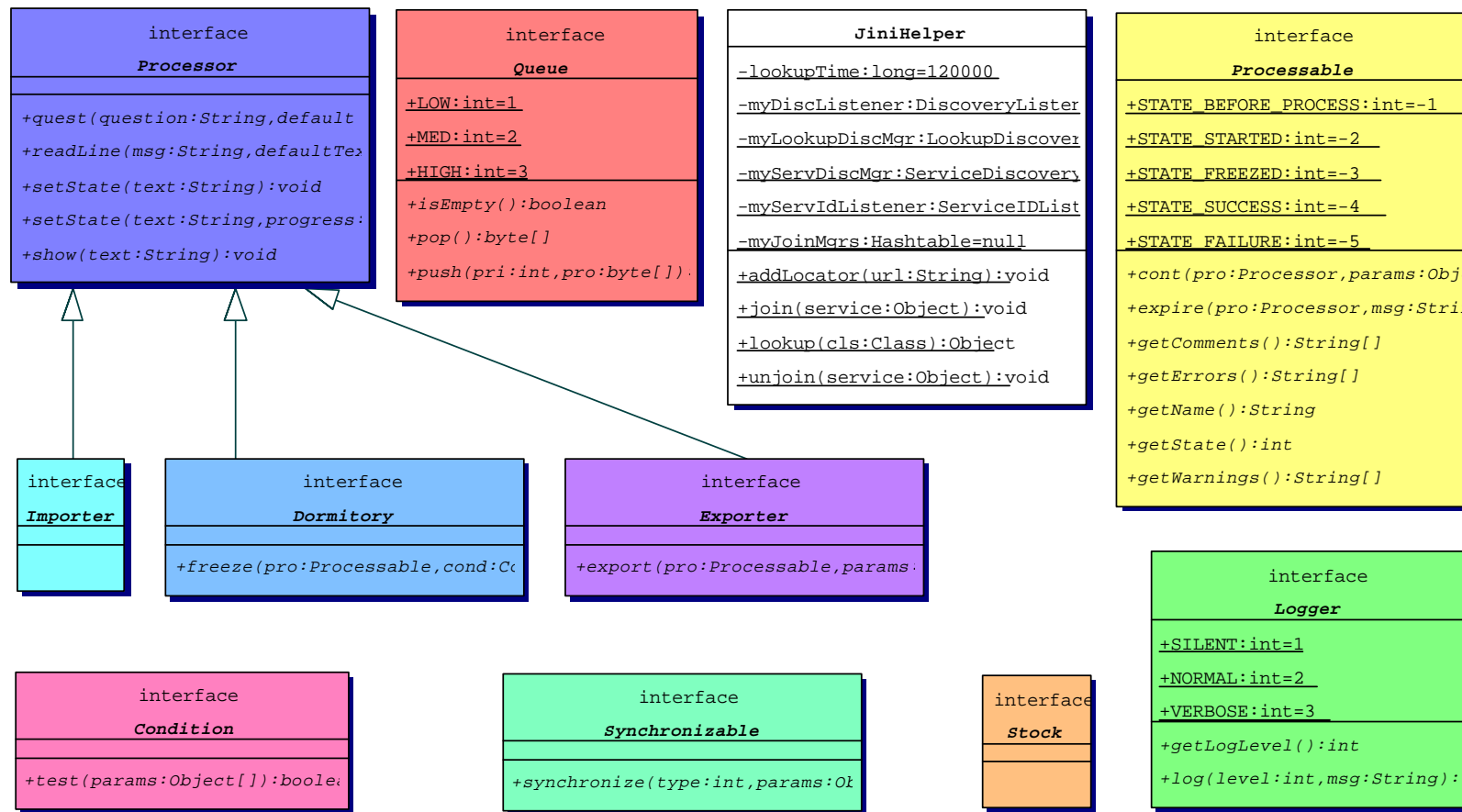


Abbildung 3.2: Implementierung der Advanced Processing Infrastructure

3 ADVANCED PROCESSING INFRASTRUCTURE

Processable.java



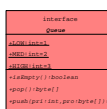
Alle Instanzen, die von einem Processor bearbeitet werden sollen, müssen dieses Interface implementieren. Ein Prozeß besteht normalerweise aus mehreren Arbeitsschritten, welche möglicherweise bei unterschiedliche Processors zu erledigen sind. Alle diese Arbeitsschritte werden dann innerhalb einer Klasse implementiert und ein Schrittzähler zeigt den nächsten Arbeitsschritt an. Das Processable kann sich in fünf verschiedenen Stati (before process, started, freezed, success, failure) befinden, welche durch konstante Integervariablen definiert werden. Es bietet Methoden, die es veranlassen mit seiner Arbeit weiterzumachen bzw. die Arbeit abzubrechen und das System zu verlassen. Außerdem bietet es die Möglichkeit den Status, Kommentare, Warnungen und Fehlermeldungen abzufragen.

Processor.java



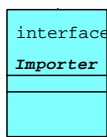
Um einem Processable Methoden anbieten zu können, muß die entsprechende Instanz das Processor-Interface implementieren. Ein Processor bietet den Processables die Möglichkeit zur Kommunikation mit dem Benutzer. Es gibt Methoden um dem Benutzer einen aktuellen Status (inklusive des prozentualen Fortschritts) und getrennt davon einen beliebig formatierten Text anzuzeigen. Weiterhin können beliebige Benutzereingaben gelesen oder auf eine bestimmte Eingabe des Benutzers gewartet werden.

Queue.java



Jedes Objekt, welches als Warteschlange für eine zugehörige Processor-Klasse dienen soll, muß das Interface Queue implementieren. Processables können mit drei verschiedenen Prioritäten (low, med, high), welche durch konstante Integervariablen definiert werden, in die Queue eingereiht werden. Höherpriorisierte Processables überholen niederpriorisierte in der Queue, verlassen diese also vorrangig. Die Queue bietet eine Methoden zum Testen, ob sie Elemente enthält und zwei weitere zum Einreihen bzw. Entnehmen von Instanzen.

Importer.java



Jede Klasse, die Processables importieren will, muß das Importer-Interface implementieren. Da Processables zum Importer kommen, dort in Aktivitäten verwickelt sind, und diesen wieder verlassen, bilden die Importer genau betrachtet eine Untergruppe der Processors. Deshalb beerbt das Importer-Interface das Processor-Interface. Damit bietet es auch die von diesem bekannte Kommunikation mit dem Benutzer an. Da

Importer im Umgang mit den Processables eine aktiv Rolle einnehmen, haben sie, außer den von Processor geerbten, keine zugreifbaren Methoden.

Exporter.java



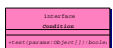
Dieses Interface wird von allen Klassen, die dem Export von Processables dienen sollen, implementiert. Die Exporter bilden aus den selben Gründen eine Processor-Untergruppe wie die Importer. Daher beerbt auch das Exporter-Interface das Processor-Interface. Es bietet neben den (geerbten) Processor-Methoden eine eigene zum Export von Processables.

Dormitory.java



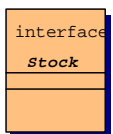
Implementierer dieses Interfaces haben Dormitory-Funktion. Da auch Dormitories eine Station auf dem Weg eines Processables sind, bilden sie eine weitere Untergruppe der Processors. Darum beerbt das Dormitory-Interface das Processor-Interface. Zusätzlich zu den von Processor geerbten Methoden bietet das Dormitory eine Methode um Processables schlafen zu legen und bei Bedarf aufzuwecken.

Condition.java



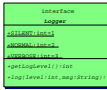
Instanzen, welche das Condition-Interface beerben, werden im Zusammenhang mit den Dormitories benötigt. Sie werden dem Dormitory zusammen mit dem schlafen zu legenden Processable übergeben. Um festzustellen, ob das schlafende Processable wieder geweckt werden kann, befragt das Dormitory die Condition. Dazu besitzt sie eine entsprechende Methode.

Stock.java



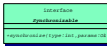
Jede Klasse, die eine zentrale Ressourcenverwaltung anbietet, implementiert das Stock-Interface. Da völlig offen ist, welche Art von Ressourcen eine implementierende Klasse verwalten wird, macht die Definition von Methoden an dieser Stelle wenig Sinn. Dennoch hat dieses Interface seine Daseinsberechtigung. Es erlaubt den Mitgliedern der Jini-Föderation nach allen vorhandenen Stocks zu suchen.

Logger.java



Um ein zentrales Logging zu implementieren beerbt man das `Logger`-Interface. Der `Logger` kennt drei Log-Level (`silent`, `normal`, `verbose`), welche durch konstante Integervariablen definiert werden. Er bietet eine Methode, um Log-Nachrichten zusammen mit deren Level anzunehmen. Außerdem gibt es eine weitere Methode, welche den aktuellen Log-Level des Loggers zurückgibt.

Synchronizable.java



Alle Klassen, die innerhalb der Advanced Processing Infrastructure untereinander synchronisierbar sein sollen, müssen das `Synchronizable`-Interface implementieren. Dieses definiert eine Methode, die als Parameter einen Synchronisations-Typ und dazugehörige Synchronisations-Daten bekommt. Der Typ legt fest, welche Art von Synchronisation stattfinden soll, und bestimmt den Typ der Daten. Ein Objekt entscheidet Anhand des Synchronisations-Typs, ob es an der aktuellen Synchronisation teilnehmen muß.

JiniHelper.java



Der `JiniHelper` ist die einzige instanziierbare Klasse des Advanced Processing Infrastructure Frameworks. Sie vereinfacht und optimiert die Benutzung von Jini.¹¹ Damit die Klasse ihre Aufgabe erfüllen kann, ist es nötig, daß alle Instanzen innerhalb einer Java Virtual Machine (JVM) die selbe `JiniHelper`-Instanz benutzen. Aus diesem Grund sind alle Methoden dieser Klasse `static`. Wenn diese Klasse geladen wird (bei ihrer ersten Referenzierung) wird nach dem `Lookup-Service (LUS)` gesucht und dessen Proxy geladen. Alle weiteren Aufrufe von `JiniHelper`-Methoden (z.B. durch vorbeikommende `Processables`) verursachen keinen weiteren Suchaktionen nach dem LUS. Außerdem wird die Suche nach häufig benutzten Diensten durch ein Cache-Mechanismus optimiert. Der `JiniHelper` bietet Methoden zum `Join`, `Lookup` und `Unjoin` von Diensten. Wenn bekannt, kann die URL des LUS dem `JiniHelper` mitgeteilt werden. Dann kann der `JiniHelper` den LUS mittels `Unicast-Paketen`¹² finden. Dies ist notwendig, wenn das Subnetz in dem die Applikation läuft kein `Multicast` unterstützt, oder die Anwendung über Subnetzgrenzen¹³ hinaus verteilt ist.

¹¹[Mic00d]

¹²anstatt mittels `Multicast-Paketen`

¹³`Multicast-Pakete` verlassen ihr Subnetz normalerweise nicht

Kapitel 4

FlexiTRUST

4.1 Design

4.1.1 Impliziertes Design

Einige grundlegende Designentscheidungen werden durch die Benutzung der Advanced Processing Infrastructure impliziert. Dieses implizierte Design werde ich im folgenden behandeln.

- Die verschiedenen Anträge (Zertifikatsantrag, Revokationsantrag, ...) werden als jeweils eigene Processables implementiert. Ein Antrag ist also in der Lage sich selber zu bearbeiten. Da er (als Processable) die volle Verantwortung für seine Aufgabe hat, obliegt es ihm zu prüfen, ob er gültig (d.h. von einer dazu berechtigten Autorität unterzeichnet) ist. Ebenso muß der Antrag dafür Sorge tragen nach vollendeter Aufgabe eine gültige Signatur von der CA zu erhalten.
- Um die von der RA erstellten Anträge in die Kern-CA¹ zu bekommen, werden Importer benutzt. Kommen mehrere Anträge in einem Paket an, so ist es die Aufgabe des jeweiligen Importers die Anträge auszupacken und loszuschicken. Außerdem gehört es zu den Pflichten des Importers zu prüfen, ob das Paket gültig (d.h. von einer dazu berechtigten Autorität unterzeichnet) ist.
- Mittels Exportern werden bearbeitete Anträge² aus der Kern-CA heraus transportiert. Je nach Typ können sie auch Anträge sammeln und diese als Paket exportieren. Dann muß der jeweilige Exporter dafür sorgen, daß sein Paket eine gültige Signatur erhält. Außerhalb werden die Anträge weiterverarbeitet. Z.B. werden die Zertifikate dem Teilnehmer übergeben und der Öffentlichkeit zugänglich gemacht.

¹siehe Abschnitt 8 auf Seite 11

²diese enthalten jetzt die erstellten Produkte

4 FLEXITRUST

- Alle Ressourcen (Schlüssel, Zertifikate, ...) werden zentral mittels eines Stock verwaltet. Dieser ist auch die für die Wartung der Ressourcen zuständig.
- Die Aktivitäten aller Teile der Kern-CA werden an zentraler Stelle mittels eines Loggers geloggt.

4.1.2 Weiteres Design

Neben den oben genannten implizierten, gibt es noch weitere, ebenso grundlegende Designentscheidungen, welche in diesem Abschnitt behandelt werden.

- Die CA soll grundsätzlich in der Lage sein mit mehreren Issuern simultan zu arbeiten. Ein Issuer definiert hierbei eindeutig:

Algorithmus: Der zur Signatur zu verwendende Algorithmus.

Provider: Der Provider des Signaturalgorithmus.

Schlüsselpaar: Das zur Signatur zu verwendende Schlüsselpaar.

Administratoren: Die dem Issuer zugeordneten Administratoren. Ein Administrator ist hier eine Instanz, die vom Issuer (durch Vergabe eines Zertifikats) dazu berechtigt ist für ihn tätig³ zu sein.

- Jeder Administrator besitzt ein eigenes Schlüsselpaar, um damit Signaturen ausstellen zu können. Administratoren können mit verschiedenen Rechten ausgestattet sein. Die verschiedenen Rechte werden durch unterschiedliche Extensions in den Zertifikaten zum Ausdruck gebracht. Diese Zertifikate werden im Stock der CA abgelegt. Wenn ein Administrator nicht mehr vertrauenswürdig ist, so wird sein Zertifikat gelöscht. Das bedeutet, die CA benötigt keine interne Revokation. Das Zertifikat eines Administrators ist genau dann gültig, wenn es sich im Stock befindet. Ändern sich die Rechte eines Administrators, so wird sein bisheriges Zertifikat gelöscht und er erhält ein neues mit entsprechenden Extensions.
- Überprüft ein Issuer die Signatur eines Administrators⁴, so holt er sich das zum Administrator gehörende Zertifikat aus dem zentralen Stock und nicht aus der Signatur des Antrages. So kann der Issuer sicher sein, die aktuellen Rechte des Administrators zu berücksichtigen.
- Um im Fehlerfall doppelt verwendete Seriennummern auf jeden Fall auszuschließen, wird die nächste freie (anstatt der letzten verwendeten) Seriennummer abgespeichert.

³z.B. Anträge auszufüllen

⁴beispielsweise auf einem Antrag

- Um die die Konfigurationsdaten (Properties) leicht und zentral warten zu können, werden diese in ein einziges Text-File eingetragen.
- Es existiert eine vollständige und eine schlanke Revokationsliste. Die vollständige enthält alle jemals ausgesprochenen Revokationen. Die schlanke nur Revokationen von Zertifikaten, die nicht ohnehin schon abgelaufen sind.

4.1.3 Aktuelles Design

Ich habe mich aktuell auf eine vereinfachte Designvariante beschränkt, um die Implementierung eines funktionstüchtigen Prototypes im Rahmen dieser Diplomarbeit zu ermöglichen. In diesem Abschnitt erkläre ich, wo sich das aktuelle Design der Basis-CA⁵ von dem vorgenannten unterscheidet. Das aktuelle Design steht nicht im Widerspruch zu obigen Überlegungen, sondern ist eine vereinfachte Variante davon, welche leicht zu dem zuvor vorgestellten Design ausgebaut werden kann.

- Die CA unterstützt genau einen Issuer d.h. genau einen Algorithmus, einen Provider, ein Schlüsselpaar und einen Administrator (keine Gruppe von Administratoren).
- Der von der CA (also vom einzigen Issuer) verwendete Algorithmus, der Provider, das Schlüsselpaar und der Administrator können allerdings frei gewählt werden.
- Der vom Administrator verwendete Algorithmus und das Schlüsselpaar können ebenfalls frei gewählt werden. Der Provider des Issuers und des Administrators muß allerdings der selbe sein.
- Die Schlüssel und Zertifikate des Issuers und des Administrators werden vom Stock im selben Keystore⁶ erwartet. Die verwendeten Aliase⁷) können frei gewählt werden, allerdings wird für den privaten Schlüssel und das Zertifikat der selben Instanz der selbe Alias benutzt. Das Passwort für den Keystore und den privaten Schlüssel des Issuers sind identisch, aber frei wählbar.
- Reiht sich ein Processable in eine Queue ein, so wandert es immer auch wirklich dort hin. Processors speichern keine abgearbeiteten Processables.
- Die Prioritäten der Processables werden von den Queues ignoriert.
- Es werden nur Zertifikate und Revokationslisten nach dem X509v3-Standard erzeugt.

⁵siehe Abschnitt 11 auf Seite 12

⁶Datenstruktur die Schlüssel und Zertifikate enthält

⁷identifiziert Daten im KeyStore

4 FLEXITRUST

- Um Daten persistent zu halten, werden diese serialisiert (bzw. DER-kodiert, wenn es sich um CODEC-Klassen handelt) und als File auf der Platte abgelegt.
- Auf transaktionales Verhalten der Applikation wird vorerst verzichtet.

4.1.4 Schnittstellen

Aus dem eben vorgestellten Design leiten sich die folgenden Schnittstellen bzw. Berührungspunkte zur Außenwelt ab:

- Jede RA besitzt (mindestens) ein Administratorschlüsselpaar. Aus diesem leitet sich ab, für welchen Issuer sie tätig sein darf.
- Die RA instanziiert für jeden Antrag ein Processable, und verpackt dieses (evtl. zusammen mit anderen) in einem von einem Importer akzeptierten Paket, welches der Kern-CA zur weiteren Verarbeitung übergeben wird.
- Bearbeitete Processables führen die von ihnen erstellten Produkte mit sich. Nach dem Export werden die Produkte von einem weiteren Bearbeitungsschritt entnommen und weiterverarbeitet. Dazu gehören vor allem die Veröffentlichung der Zertifikate und die dauerhafte Speicherung der abgearbeiteten Processables.

4.2 Implementierung

4.2.1 Allgemeines

Da sich das Design von FlexiTRUST auf die Advanced Processing Infrastructure stützt, tut dies die Implementierung natürlich auch. Es wurde ein Vielzahl von Klassen implementiert, die je nach Aufgabe in verschiedene Pakete eingeordnet wurden. Im folgenden werde ich die Pakete und die darin enthaltenen Klassen kurz vorstellen. Genauere Implementierungsdetails zu diesen Packages und den darin enthaltenen Klassen kann man den Javadocs im Anhang A ab Seite 69 entnehmen.

Die Implementierung hält sich an das oben vorgestellte aktuelle Design. Es wurde aber darauf geachtet diese so zu gestalten, daß eine spätere Anpassung an das vollständige Design leicht möglich ist. So wurden z.B. einige Methoden mit Parametern versehen, die für das aktuelle Design nicht erforderlich, aber im vollständigen Design notwendig sind.

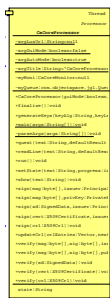
Viele Methoden der im folgenden vorgestellten Klassen bekommen Byte-Arrays als Parameter bzw. haben Byte-Array als Rückgabotyp. Dies war nötig, da bis vor kurzem die Klassen aus der CODEC-Bibliothek nicht serialisierbar, also nicht RMI-fähig waren. Um solche Klassen dennoch benutzen zu können, werden die entsprechenden

Objekte DER-kodiert⁸ übergeben. Da die CODEC-Klassen jetzt serialisierbar sind, werden die Typen der Parameter und Rückgabewerte im Verlauf der weiteren Entwicklung von FlexiTRUST angepaßt werden.

4.2.2 de.tud.cdc.flexiTrust.ca.app

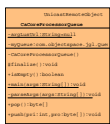
Alle für die Applikation FlexiTRUST spezifischen Klassen (mit Ausnahme der Processables und den Klassen der Initialphase) sind im Package `de.tud.cdc.flexiTrust.ca.app` enthalten. Abbildung 4.1 auf Seite 32 zeigt die Struktur dieses Packages. Im folgenden werden die einzelnen Klassen kurz beschrieben. Genauere Implementierungsdetails zu diesem Package kann man den Javadocs im Anhang A ab Seite 69 entnehmen.

CaCoreProcessor.java



Der `CaCoreProcessor` implementiert das `Processor`-Interface. Er stellt den `Processables` eine Reihe von kryptographischen Grundfunktionalitäten zur Verfügung. Dazu gehören die Signatur und Verifikation von `Byte-Arrays`, `SignedData`-Objekten und `X509Certificate`-Objekten. Außerdem kann er letztgenannte Zertifikate revozieren und Schlüsselpaare erzeugen. Die für einen `Processor` obligatorische Möglichkeit zur Kommunikation mit dem Benutzer bietet er durch Delegation an den `CaCoreMonitor`. Der `CaCoreProcessor` erwartet komprimierte `Processables` in seiner Queue. Um einen Bug in einer der verwendeten Bibliotheken⁹ zu fixen, verschiebt der `CaCoreProcessor` die Gültigkeitszeiträume aller Zertifikate um zwei Stunden in die Vergangenheit. Sobald eine gepatchte Version der betroffenen Bibliothek vorliegt, wird diese Zeitverschiebung aus dem Code entfernt.

CaCoreProcessorQueue.java

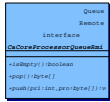


Diese Klasse bietet einige `Remote`-Methoden, und implementiert deshalb das ihr zugeordnete `RMI`-Interface `CaCoreProcessorQueueRmi`, welches seinerseits das `Queue`-Interface erweitert. Die `CaCoreProcessorQueue` ist die dem `CaCoreProcessor` zugeordnete Warteschlange. Die augenblickliche Version ist eine echter `FIFO`, ignoriert also die Prioritäten der eingereichten `Processable`-Instanzen. Ihre `push`- und `pop`-Methoden erwarten bzw. liefern Instanzen vom Typ `byte[]`, da die `Processables` komprimiert eingereicht werden.

⁸[Sec93]

⁹in der CODEC-Bibliothek

CaCoreProcessorQueueRmi.java



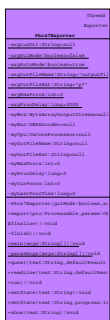
Die Klasse `CaCoreProcessorQueueRmi` ist das der Klasse `CaCoreProcessorQueue` zugeordnete RMI-Interface. Sie erweitert das `Queue-Interface`, und definiert die Remote-Methoden von `CaCoreProcessorQueue`. Dies sind Methoden zum Einreihen und Entnehmen von `Processables` und eine Methode zum Testen, ob die Queue leer ist.

Pkcs7Importer.java



Der `Pkcs7Importer` implementiert das `Importer-Interface` und damit das `Processor-Interface`. Er lädt `Processables` aus einem File, welches ein DER-kodiertes¹⁰ `SignedData` Objekt enthält. Dieses Objekt ist signiert¹¹, und enthält eine Reihe weiterer DER-kodierter `SignedData` Objekte. Jedes davon ist ebenfalls signiert und enthält genau ein serialisiertes `Processable` Objekt. Dieses File-Format ist dasselbe wie das, welches vom `Pkcs7Exporter` produziert wird. Wenn das File eingelesen wird, wird das äußere `SignedData` Objekt dekodiert und seine Signatur verifiziert. Danach werden alle `Processable-Objekte` daraus entnommen und ihr Prozess gestartet. Die für einen `Processor` obligatorische Möglichkeit zur Kommunikation mit dem Benutzer realisiert der `Pkcs7Importer` durch Delegation an den `CaCoreProcessor`.

Pkcs7Exporter.java



Der `Pkcs7Exporter` implementiert das `Exporter-Interface` und damit das `Processor-Interface`. Er entnimmt `Processables` aus der ihm zugeordneten Queue (in welcher sie in komprimierter Form vorliegen), sammelt sie und speichert mehrere zusammen in einem File. Jedes der Queue entnommene `Processable` wird serialisiert und im `Data-Feld` eines frischen `SignedData-Objektes` abgelegt und unterschrieben. Dieses `SignedData-Objekt` wird DER-kodiert und zusammen mit anderen im `Data-Feld` eines übergeordneten `SignedData-Objektes` abgelegt. Sind genug `Processables` gesammelt, so wird das übergeordnete `SignedData-Objekt` unterzeichnet und DER-kodiert als File abgelegt. Dieses resultierende File hat dasselbe Format, welches vom `Pkcs7Importer` erwartet wird. Die Methoden zur Realisierung der Benutzerinteraktion delegieren an den `CaCoreProcessor`.

¹⁰[Sec93]

¹¹von einer vertrauenswürdigen Registration-Authority

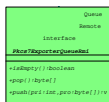
4 FLEXITRUST

Pkcs7ExporterQueue.java



Da diese Klasse verschiedene Remote-Methoden bietet, implementiert sie das ihr zugeordnete RMI-Interface `Pkcs7ExporterQueueRmi` und damit das `Queue`-Interface. Die `Pkcs7ExporterQueue` ist die dem `Pkcs7Exporter` zugeordnete Warteschlange. Die augenblickliche Version ist eine echter FIFO, ignoriert also die Prioritäten, der eingereichten `Processable`-Instanzen. Da diese zudem komprimiert entgegengenommen bzw. verschickt werden erwarten bzw. liefern ihre `push`- und `pop`-Methoden Instanzen vom Typ `byte[]`.

Pkcs7ExporterQueueRmi.java



Die Klasse `Pkcs7ExporterQueueRmi` ist das der Klasse `Pkcs7ExporterQueue` zugeordnete RMI-Interface. Sie erweitert das `Queue`-Interface. Hier werden die Remote-Methoden von `Pkcs7ExporterQueue` definiert. Dies sind Methoden zum Einreihen und Entnehmen von `Processables` und eine Methode zum Testen ob die Queue leer ist.

CaStock.java



Damit diese Klasse Remote-Methoden bieten kann implementiert sie das ihr zugeordnete RMI-Interface `CaStockRmi`. Über dieses implementiert sie das `Stock`-Interface. Der `CaStock` verwaltet alle Ressourcen der CA an zentraler Stelle. Dazu bietet er Methoden um auf Zertifikate, Schlüssel, Algorithmen und andere Properties zugreifen zu können. Die Properties werden dem Properties-File entnommen. Der `CaStock` legt einige Datenstrukturen in Files ab, um diese persistent zu halten. Im Laufe der weiteren Entwicklung von FlexiTRUST wird statt der Files eine Datenbank verwendet. Nachfolgend wird erklärt wie die Daten momentan abgelegt werden.

- Schlüssel und Zertifikate werden in einem `KeyStore` gehalten und mit der `KeyStore` eigenen Methode als File abgelegt.
- Die Seriennummern von Issuer und Administrator sind Instanzen von `BigInteger` und werden serialisiert auf die Platte geschrieben.
- Die Liste aller erstellten Zertifikate ist ein `Hashtable`. Dieser wird ebenfalls serialisiert abgespeichert.
- Die vollständige und die schlanke Revokationsliste¹² sind Instanzen von `X509Crl` und werden DER-kodiert rausgeschrieben.

¹²siehe Abschnitt 34 auf Seite 29

CaStockRmi.java



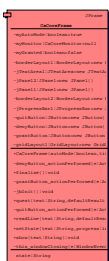
Die Klasse `CaStockRmi` ist das der Klasse `CaStock` zugeordnete RMI-Interface. Sie erweitert das `Remote`- und das `Stock`-Interface. Hier werden die Remote-Methoden von `CaStock` definiert. Dies sind unter anderem Methoden, um Zertifikate, Schlüssel oder die nächste zu verwendende Seriennummer von bestimmten Issuern oder Administratoren zu erfragen. Es können Zertifikate revoziert oder der Liste aller Zertifikate eines Issuers hinzugefügt werden. Es ist auch möglich ein Property aus dem Properties-File zu erfragen.

CaCoreMonitor.java



Der `CaCoreMonitor` ist dazu gedacht von `Processor`-Implementierungen mittels Delegation angesprochen zu werden. Er implementiert die Möglichkeit zur Interaktion mit dem Benutzer. Die Benutzerinteraktion kann über eine graphische Oberfläche oder rein textuell geschehen. Wird der `CaCoreMonitor` für graphische Interaktion instanziiert, so delegieren seine Methoden an den `CaCoreFrame`. Zur rein textuellen Kommunikation benutzt er den `KbdHelper`.

CaCoreFrame.java

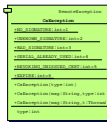


Zweck des `CaCoreFrame` ist es, dem `CaCoreMonitor` eine Delegationsmöglichkeit für die graphische Benutzerinteraktion zu bieten. Durch die Einführung dieser Klasse sind außerdem die Applikation und die graphische Oberfläche voneinander getrennt implementiert. Daher ist es möglich die GUI auszutauschen, ohne an der restlichen Applikation Anpassungen vornehmen zu müssen. Diese Klasse ist sehr einfach gehalten, und bietet nur die nötigste Funktionalität. Die Ausgabe der GUI beschränkt sich auf eine `TextArea` zur Anzeige von Texten und eine `StatusBar` um den aktuellen Status nebst prozentualem Fortschritt anzuzeigen. Als mögliche

Benutzereingaben werden im Moment «grant», «deny» und «quit» als Buttons mit entsprechender Aufschrift angeboten. Mit der weiteren Entwicklung der Applikation wird auch diese Klasse komfortablere Methoden anbieten.

4 FLEXITRUST

CaException.java



Die `CaException` ist ein Wrapper für alle in der Applikation auftretenden Ausnahmen. Da sie auch über RMI eingesetzt werden soll, erbt sie von `RemoteException`. Dies ermöglicht es ihr zudem, die ursprüngliche `Exception` (wenn vorhanden) mitzuführen, da die Klasse `RemoteException` über einen entsprechenden Mechanismus verfügt. Momentan gibt es sechs verschiedene `Exception`-Typen (`no signature`, `unknown signature`, `bad signature`, `serial already used`, `revoking unissued cert`, `expire`), welche durch Integerkonstanten definiert sind. Es steht eine Methode zur Verfügung, um den aktuellen `Exception`-Typ abzufragen.

4.2.3 de.tud.cdc.flexiTrust.ca.init

Alle für die Initialphase von FlexiTRUST spezifischen Klassen sind im Package `de.tud.cdc.flexiTrust.ca.init` enthalten. Zweck der Initialphase ist die Erzeugung aller Daten und Datenstrukturen, die für jede Instanz von FlexiTRUST verschieden sein müssen. Da diese Klassen äußerst sicherheitssensitive Daten erzeugen (z.B. den privaten Schlüssel eines Issuers) arbeiten sie vollkommen autark. Sie sind nicht Teil des CA-Systems, sondern können davon unabhängig gestartet und vollkommen getrennt davon gehalten werden. Im folgenden werden die zu erzeugenden Daten nebst ihren Datenstrukturen vorgestellt.

- Das Schlüsselpaar des Issuers, inklusive dazugehörigem `Selfsigned-Certificate` werden in einem `KeyStore` gespeichert.
- Die nächste zu verwendende Seriennummer des Issuers wird als `BigInteger` abgelegt.
- Das Schlüsselpaar des Administrators und einem von der CA ausgestelltem dazugehörigen Zertifikat werden in einem frischen `KeyStore` gespeichert. Der öffentliche Schlüssel des Administrators wird zusätzlich im `KeyStore` des Issuers hinterlegt.
- Die nächste zu verwendende Seriennummer des Administrators wird als `BigInteger` abgelegt.
- Die Liste aller vom Issuer ausgestellten Zertifikate wird als `Hashtable` realisiert.
- Die vollständige und die schmale Revokationsliste sind Objekte vom Typ `X509Crl`.

Alle diese Daten werden so abgelegt, daß der `CaStock` darauf zugreifen kann. Abbildung 4.2 zeigt die Struktur dieses Packages. Im folgenden werden die einzelnen Klassen kurz beschrieben. Genauere Implementierungsdetails zu diesem Package kann man den Javadocs im Anhang A ab Seite 82 entnehmen.

4.2 IMPLEMENTIERUNG

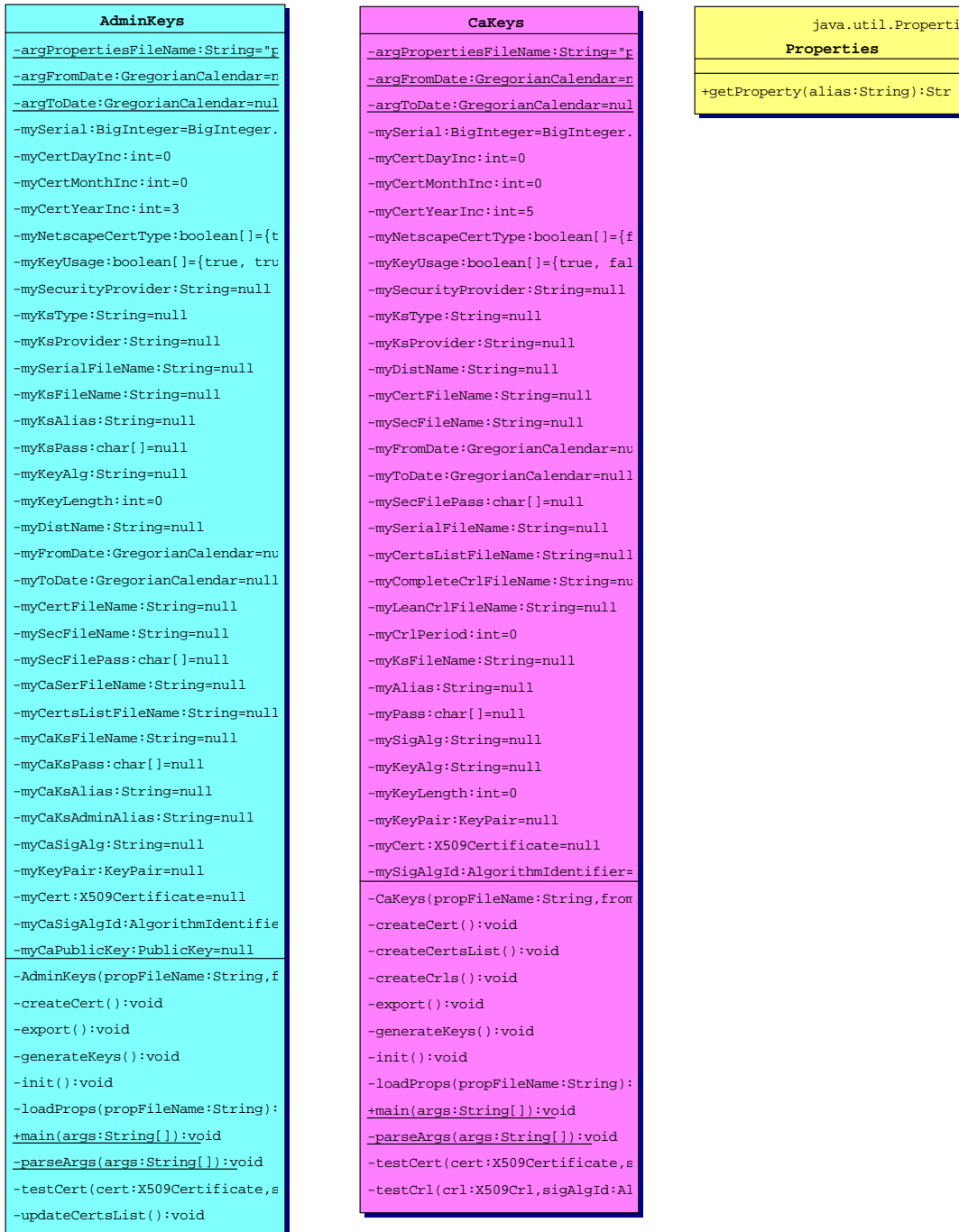
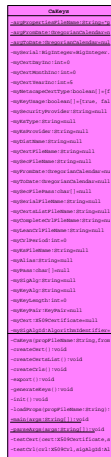


Abbildung 4.2: Das Initiation-Package

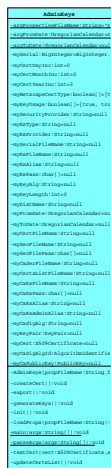
4 FLEXITRUST

CaKeys.java



Diese Klasse erstellt die initialen Daten eines Issuers und speichert diese in entsprechenden Datenstrukturen. Zunächst wird ein neues Schlüsselpaar generiert und ein dazugehöriges Selfsigned-Certificate¹³ erstellt. Privater Schlüssel und Zertifikat werden in ein PKCS#12-Security-File gespeichert. Zum täglichen Gebrauch durch die CA werden diese beiden zusätzlich in einem KeyStore abgelegt und mit der KeyStore eigenen Methode als File rausgeschrieben. Weiterhin wird ein BigInteger mit dem Wert 2 erzeugt und als nächste zu verwendende Seriennummer serialisiert auf die Platte geschrieben. Das Selfsigned-Certificate wird in ein Hashtable gespeichert.¹⁴ Dies ist die Liste aller ausgestellten Zertifikate und wird serialisiert als File abgelegt. Als letztes werden die vollständige und die schmale Version der Revokationsliste¹⁵ erzeugt. Dazu werden zwei neue Objekte der Klasse X509Cr1 instanziiert, DER-kodiert und in Files abgespeichert. Alle erzeugten Daten und Datenstrukturen werden während und nach deren Erzeugung intensiv getestet, um deren Konsistenz zu gewährleisten. Standardmäßig werden alle notwendigen Parameter dem Properties-File entnommen. Es ist aber möglich diese durch Kommandozeilenparameter zu überschreiben.

AdminKeys.java



Diese Klasse erzeugt die initialen Daten eines Administrators und speichert diese in neuen Datenstrukturen bzw. trägt sie in die Datenstrukturen des zugehörigen Issuers ein. Zunächst wird ein neues Schlüsselpaar generiert. Dann wird im Namen des zu diesem Administrator gehörenden Issuers ein Zertifikat erstellt, welches den neuen öffentlichen Schlüssel diesem Administrator zuordnet. Privater Schlüssel und Zertifikat werden als PKCS#12-Security-File aufbewahrt. Zur täglichen Verwendung werden diese beiden außerdem in einem neuen KeyStore abgelegt und dieser auf die Platte geschrieben. Das Zertifikat kommt zusätzlich in den KeyStore des Issuers und in dessen Zertifikatsliste. Es wird ein BigInteger mit dem Wert 1 erzeugt, serialisiert und als nächste zu verwendende Seriennummer dieses Administrators auf die Platte geschrieben. Alle erzeugten Daten und Datenstrukturen werden während und nach deren Erzeugung intensiv getestet, um deren Konsistenz zu gewährleisten. Wie bei CaKeys werden die Defaultparameter dem Properties-File entnommen. Auch hier können diese aber durch Kommandozeilenparameter überschrieben werden.

¹³Seriennummer = 1

¹⁴key = Seriennummer

¹⁵siehe Abschnitt 34 auf Seite 29

Properties.java

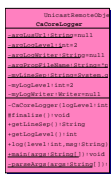


Diese Klasse dient der Robustheit der initialen Phase. Sie beerbt die im JAVA-Core-API enthaltene `Properties`-Klasse und verhält sich, bis auf nachfolgend beschriebene Ausnahme, genau wie diese. Statt beim Zugriff auf ein nicht vorhandenes Property den Wert `null` zu liefern, wirft diese Version eine entsprechende Exception.

4.2.4 de.tud.cdc.flexiTrust.ca.log

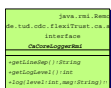
Die im Package `de.tud.cdc.flexiTrust.ca.log` enthaltenen Klassen realisieren ein zentralisiertes Logging. Abbildung 4.3 auf Seite 40 zeigt die Struktur dieses Packages. Im folgenden werden die einzelnen Klassen kurz beschrieben. Genauere Implementierungsdetails zu diesem Package kann man den Javadocs im Anhang A ab Seite 84 entnehmen.

CaCoreLogger.java



Der `CaCoreLogger` ist das Herzstück des gesamten Loggings. Da diese Klasse einige Remote-Methoden bietet, implementiert sie das zugehörige RMI-Interface `CaCoreLoggerRmi`, welches seinerseits das `Logger`-Interface erweitert. Der `CaCoreLogger` ist neben dem `CaStock` die einzige Klasse, die direkt auf das `Properties`-File zugreift. Dies ist notwendig, da der `CaStock` den `Log-Service` nutzt, und folglich der `CaCoreLogger` vor dem `CaStock` instanziiert werden muß. Ganz allgemein sollte der `Log-Service` der erste Service im System sein. Der `Log-Service` kann sich in drei verschiedenen `Log-Level` (`silent`, `normal`, `verbose`) befinden. Eine `Log-Nachricht` kann ebenfalls einen dieser drei Level annehmen, und wird nur dann gespeichert, wenn der Level des `Log-Service` gleich oder geschwätziger ist als derjenige der `Log-Nachricht`. Um verschiedene Arten des Loggings zu ermöglichen, schreibt der `CaCoreLogger` die Nachrichten in ein Objekt der abstrakten Klasse `Writer`, deren konkrete Implementierung zur Laufzeit dynamisch aus dem `Properties`-File ermittelt wird. Es empfiehlt sich zum Zugriff auf den `CaCoreLogger` den `LogHelper` zu verwenden.

CaCoreLoggerRmi.java



Die Klasse `CaCoreLoggerRmi` ist das der Klasse `CaCoreLogger` zugeordnete RMI-Interface. Sie erweitert das `Logger`-Interface. Hier werden die Remote-Methoden von `CaCoreLogger` definiert. Dies sind eine Methode um das vom `Log-Service` benutzte Zeilenendezeichen zu erfragen, eine die den aktuellen `Log-Level` zurückgibt und eine um `Log-Nachrichten` zu übergeben.

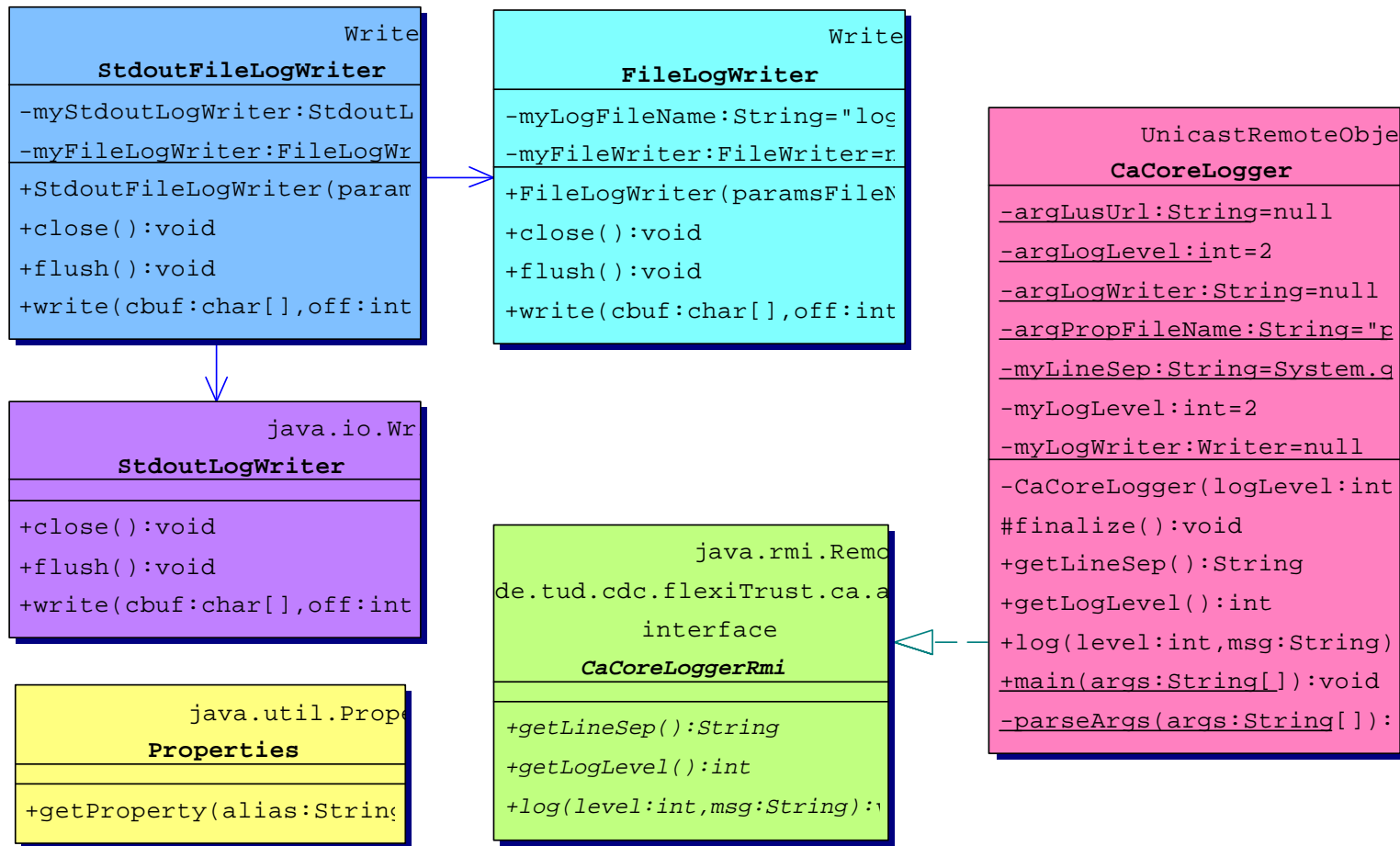
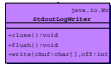


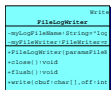
Abbildung 4.3: Das Logging-Package

StdoutLogWriter.java



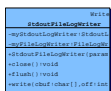
Der `StdoutLogWriter` beerbt die abstrakte `Writer`-Klasse. Er ist zur Benutzung durch `CaCoreLogger` gedacht und schreibt die ihm übergebenen Nachrichten auf die Standard-Ausgabe. Dies geschieht durch Delegation der `flush`- und `write`-Methoden an die entsprechenden Methoden von `System.out`. Die `close`-Methode tut nichts, da die Schließung von `System.out` keinen Sinn macht.

FileLogWriter.java



Der `FileLogWriter` schreibt die ihm übergebenen Nachrichten in ein File, dessen Name er aus dem Properties-File bezieht. Dazu delegiert er seine `close`-, `flush`- und `write`-Methoden an die Klasse `FileWriter`. Da er zur Benutzung durch den `CaCoreLogger` gedacht ist, beerbt er die abstrakte Klasse `Writer`.

StdoutFileLogWriter.java



Der `StdoutFileLogWriter` delegiert alle seine Methoden sowohl an den `StdoutLogWriter` als auch an den `FileLogWriter`. Das bedeutet er loggt auf die Standard-Ausgabe und in ein File. Den Namen des Log-Files entnimmt er dem Properties-File. Um vom `CaCoreLogger` benutzt werden zu können, beerbt er die abstrakte `Writer`-Klasse.

Properties.java



Die Klasse `Properties` ist Erbe der im Java-Core-API enthaltenen `Properties`-Klasse. Ihr Zweck ist es, die Robustheit des Loggings zu erhöhen. Statt wie ihr Vorfahre beim Zugriff auf ein nicht vorhandenes Property den Wert `null` zu liefern, wirft diese Version eine entsprechende Exception.

4.2.5 de.tud.cdc.flexiTrust.ca.pro

Das Package `de.tud.cdc.flexiTrust.ca.pro` enthält alle verfügbaren `Processables`. Abbildung 4.4 auf Seite 42 zeigt die Struktur dieses Packages. Im folgenden werden die einzelnen Klassen kurz beschrieben. Genauere Implementierungsdetails zu diesem Package kann man den Javadocs im Anhang A ab Seite 89 entnehmen.

4 FLEXITRUST

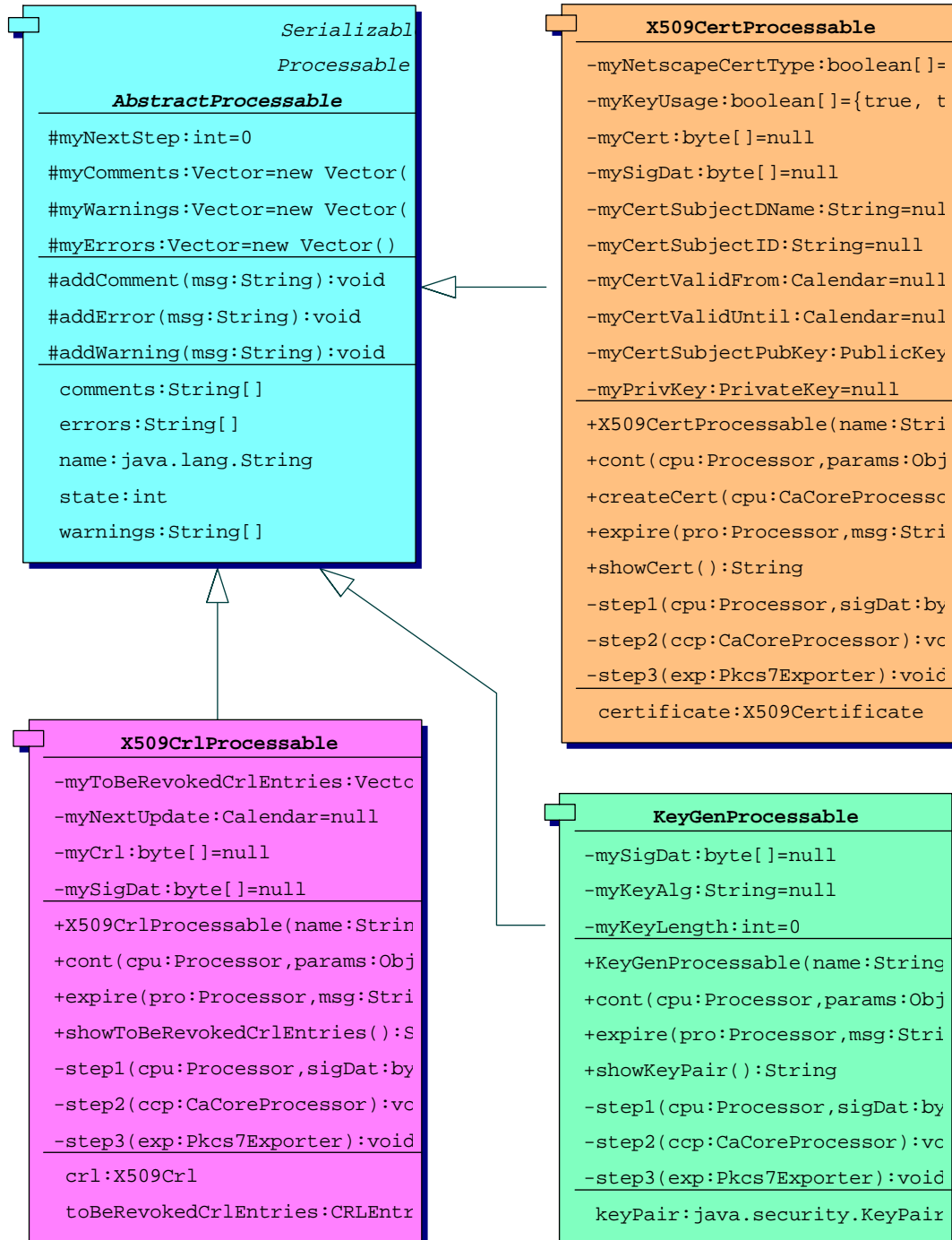


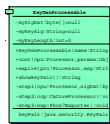
Abbildung 4.4: Das Processables-Package

AbstractProcessable.java



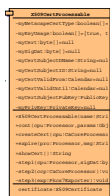
Die abstrakte Klasse `AbstractProcessable` implementiert das `Processable`-Interface. Sie stellt die Methoden, die für alle `Processables` gleich sind (sein können) zur Verfügung. Dazu gehören Methoden, um dem Objekt Kommentare, Warnungen und Fehlermeldungen hinzuzufügen bzw. sich diese ausgeben zu lassen. Außerdem existieren Methoden, um sich den Namen und den aktuellen Status der Instanz zu erfragen. Es empfiehlt sich für jedes `Processable` diese Klasse zu beerben.

KeyGenProcessable.java



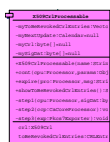
Die Klasse `KeyGenProcessable` beerbt `AbstractProcessable`, und implementiert damit das `Processable`-Interface. Seine Aufgabe ist die Erzeugung eines Schlüsselpaars für einen Algorithmus, der ihm bei der Instanziierung mitgeteilt wird. Alle seine Arbeitsschritte kann es bei einem `CaCoreProcessor` erledigen. Der erste Schritt ist die Verifikation der eigenen Signatur, danach folgt die Erzeugung des Schlüsselpaars und schließlich der Export durch einen `Pkcs7Exporter`.

X509CertProcessable.java



Aufgabe der Klasse `X509CertProcessable` ist die Erzeugung eines X509-Zertifikates. Sie ist Nachfahre von `AbstractProcessable`, und implementiert damit das `Processable`-Interface. Die für ihre Aufgaben benötigten Daten bekommen die Instanzen dieser Klasse im Constructor übergeben. Sie benötigen ausschließlich Methoden des `CaCoreProcessor`. Nach der Verifikation der eigenen Signatur folgt die Erzeugung des Zertifikates und schließlich der Export durch einen `Pkcs7Exporter`. Zu Testzwecken wird momentan das erstellte Zertifikat DER-kodiert auf die Platte geschrieben. Zusätzlich werden der private Schlüssel und das Zertifikat als PKCS#12-Security-File abgelegt.

X509CrlProcessable.java



Als Nachfahre von `AbstractProcessable` implementiert die Klasse `X509CrlProcessable` und damit das `Processable`-Interface. Ihre Aufgabe ist die Revokation von Zertifikaten und die Besorgung einer aktuellen, schlanken Revokationsliste. Die zu revozierenden Zertifikate bekommt eine Objekt dieser Klasse bei der Instanziierung mitgeteilt. Der einzige benötigte Processor-Typ ist `CaCoreProcessor`. Zunächst testet ein `X509CrlProcessable`-Objekt die eigene Signatur. Danach folgt die Revokation der Zertifikate und die Besorgung der neuen Revokationsliste. Schließlich erfolgt der Export durch einen `Pkcs7Exporter`.

4 FLEXITRUST

4.2.6 de.tud.cdc.flexiTrust.ca.util

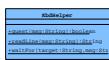
Die im Package `de.tud.cdc.flexiTrust.ca.util` enthaltenen Klassen sind Tools, die verschiedene Aufgaben übernehmen bzw. diese optimieren. Abbildung 4.5 auf Seite 45 zeigt die Struktur dieses Packages. Im folgenden werden die einzelnen Klassen kurz beschrieben. Genauere Implementierungsdetails zu diesem Package kann man den Javadocs im Anhang A ab Seite 93 entnehmen.

CaStockHelper.java



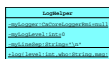
Die Klasse `CaStockHelper` vereinfacht und optimiert die Benutzung des `CaStock`. Sie bietet die selben Methoden an wie dieser, verbirgt aber die nötigen Netzzugriffe und stellt die Methoden lokal zur Verfügung. Damit die Klasse ihre Aufgabe erfüllen kann, ist es nötig, daß alle Instanzen innerhalb einer Java Virtual Machine (JVM) die selbe `CaStockHelper`-Instanz benutzen. Aus diesem Grund sind alle Methoden dieser Klasse `static`. Wenn diese Klasse geladen wird, wird ein Lookup nach dem `CaStock`-Service gemacht, und dessen Proxy geholt. Alle weiteren Aufrufe von `CaStockHelper`-Methoden (z.B. durch vorbeikommende `Processables`) verursachen keine weiteren Jini-Aktivitäten.

KbdHelper.java



Der `KbdHelper` vereinfacht die textuelle Ein- und Ausgabe. Dazu bietet er Methoden zum Lesen von der Standard-Eingabe und zum Schreiben auf die Standard-Ausgabe. Außerdem existiert eine Methode um auf eine bestimmte Eingabe des Benutzers zu warten. Um diesen Helper einfach nutzen zu können, hat er ausschließlich statische Methoden.

LogHelper.java



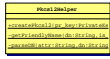
Die Klasse `LogHelper` dient der Optimierung und Vereinfachung der Benutzung des `CaCoreLoggers`. Es ist nötig, daß alle Objekte innerhalb einer JVM die selbe `LogHelper`-Instanz benutzen, damit der `LogHelper` seine Aufgabe erfüllen kann. Deshalb sind alle seine Methoden statisch. Bei der ersten Referenzierung dieser Klasse wird nach dem `CaCoreLogger` gesucht und dessen Proxy geholt. Alle weiteren Aufrufe von `LogHelper`-Methoden (z.B. von durchreisenden `Processables`) verursachen keine weiteren Jini-Aktivitäten. Als weitere Optimierung wird der Level einer ankommenden Log-Nachricht mit dem aktuellen Log-Level des `CaCoreLoggers` verglichen. Die Nachricht wird nur dann weitergegeben, wenn sie vom `CaCoreLogger` auch wirklich gespeichert wird. Dies dient der Senkung der Netzlast und vermindert den Flaschenhalseffekt.



Abbildung 4.5: Das Utilities-Package

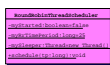
4 FLEXITRUST

Pkcs12Helper.java



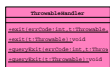
Der `Pkcs12Helper` bietet die Möglichkeit ein PKCS#12–Security–File¹⁶ zu erstellen. Alle Methoden dieser Klasse sind `static`, um den Umgang mit ihr zu vereinfachen. Der Quellcode dieser Klasse wurden fast unverändert aus Markus Taks Sourcen übernommen.

RoundRobinThreadScheduler.java



Da das Java–Laufzeitsystem laut Spezifikation kein Thread–Scheduling implementieren muß, wurde die Klasse `RoundRobinThreadScheduler` eingeführt. Sie teilt allen Threads mit einer Priorität kleiner als `Thread.MAX_PRIORITY` eine vom Benutzer zu bestimmende Zeitscheibe zu. Threads mit niedriger Priorität müssen warten bis solche mit höherer Priorität beendet sind. Innerhalb einer Prioritätsstufe werden die Threads nach dem Round–Robin–Prinzip abgearbeitet. Threads mit der Priorität `Thread.MAX_PRIORITY` haben immer Vorrang, und werden nicht unterbrochen.

ThrowableHandler.java



Die Klasse `ThrowableHandler` bietet Methoden, um die laufende Applikation nach dem Fangen eines `Throwable`s geordnet zu beenden. Dies kann mit oder ohne Rückfrage beim Benutzer geschehen. Es möglich die Ausgaben auf beliebige `PrintStreams` umzuleiten und Eingaben von beliebigen `InputStreams` zu lesen.

¹⁶[Lab00]

Kapitel 5

Betrieb

5.1 Eigenschaften

In diesem Abschnitt benenne ich die Fähigkeiten der zum jetzigen Zeitpunkt (21.01.2001) vorliegenden Version¹ von FlexiTRUST. Was für die weitere Entwicklung der Software noch zu tun ist, kann in Kapitel 6 ab Seite 60 nachgelesen werden.

Die vorliegende Applikation ist in der Lage Schlüssel zu erzeugen, Zertifikate auszustellen und diese wieder zu Revozieren. Alle Aktionen werden geloggt. Die Eigenschaften Skalierbarkeit und dynamische Erweiterbarkeit sind voll ausprogrammiert. Die Anwendung ist eine verteilte Applikation deren einzelne Prozesse in der Lage sind sich selbst zu organisieren. Nachfolgend werden die momentanen Eigenschaften der Software genauer benannt. Wie bereits erwähnt kann und wird das System im Laufe der Zeit dynamisch erweitert werden.

- Allen startbaren Klassen können über die Kommandozeile Parameter mitgegeben werden. Diese Werte werden dann statt den entsprechenden Einträgen im Properties-File benutzt. Eine Beschreibung der möglichen Argumente erhält man durch Angabe des Parameters «-h» beim Aufruf der jeweiligen Klasse.
- Es stehen Tools zur Verfügung, welche die für jede CA einmaligen Daten und Datenstrukturen² erzeugen und in geeigneter Weise bereitstellen. Diese Tools führen intensive Tests auf den von ihnen erzeugten Produkten aus, um deren Konsistenz zu gewährleisten.
- Es gibt ein zentralen Logging-Mechanismus, der von allen Services im System benutzt werden kann. Es ist möglich auf die Standard-Ausgabe, in ein File oder in beide vorgenannten Ziele zu loggen. Es besteht die Möglichkeit weitere Logging-Targets hinzuzufügen.

¹die von mir entwickelte, nicht die von Markus Tak

²z.B. das CA-Schlüsselpaar

5 BETRIEB

- Die zentrale Verwaltung aller von der Applikation benötigten Ressourcen ist durch einen entsprechenden Service realisiert. Dieser liefert auf Anfrage die benötigten Schlüssel, Zertifikate, Algorithmen und andere Properties.
- Der Import von Anträgen bzw. der Export der entstandenen Produkte erfolgt in einem verschachtelten PKCS#7-Format.
- Das System stellt Möglichkeiten (Arbeitsplätze) zur Signatur und Verifikation von Byte-Arrays, SignedData- und X509Certificate-Objekten zur Verfügung. Außerdem können letztgenannte Zertifikate revoziert und Schlüssel-paare erzeugt werden.
- Anträge zur Erzeugung von Schlüsselpaaren (für beliebige Algorithmen), zur Ausstellung von Zertifikaten (nur X509), zu deren Revokation und zur Abholung von Revokationslisten sind implementiert.
- Alle von der CA erstellten Zertifikate und Revokationen werden dauerhaft gespeichert.
- Es existiert eine vollständige und eine schlanke Revokationsliste. Die vollständige enthält alle jemals ausgesprochenen Revokationen. Die schlanke nur Revokationen von Zertifikaten, die nicht ohnehin schon abgelaufen sind.
- Um das System testen zu können, stehen vorgefertigte Testszenarien zur Verfügung, die durch ein einfachen Makefile-Aufruf gestartet werden können.

5.2 Installation

5.2.1 Systemanforderungen

Im folgenden werden die Anforderungen beschrieben, die ein Rechner erfüllen muß, um Host für ein (oder mehrere) FlexiTRUST-Service zu sein:

- Auf jedem Host muß eine Java Virtual Machine in der Version 1.2 (oder höher) installiert sein. Hierzu gehört auch ein korrekt gesetzter Pfad.
- Alle Hosts müssen durch ein TCP/IP-fähiges Netzwerk verbunden sein.
- Um das Makefile nutzen zu können wird eine Bash oder CShell und gmake benötigt.

5.2.2 Installationsvorgang

Das Installieren der Applikation beschränkt sich auf ein einfaches Dekomprimieren des gelieferten Archives. Es ist darauf zu achten, daß beim Entpacken die ursprüngliche Verzeichnisstruktur wieder hergestellt wird.

5.3 Initialphase

In der Initialphase werden alle zum Betrieb der CA benötigten Daten und Datenstrukturen erzeugt. Dies geschieht mittels der beiden Klassen `CaKeys` und `AdminKeys` aus dem Package `de.tud.cdc.flexiTrust.ca.init`. Da diese beiden Tools sicherheitssensitive Daten erzeugen, werden sie getrennt von der restlichen Anwendung als Standalone-Applikationen ausgeführt. Außerdem führen sie während und nach der Generierung ihrer Produkte intensive Test darauf aus, um deren Konsistenz zu gewährleisten. Damit die Applikation lauffähig ist, müssen mindestens ein Satz Issuer-Daten und ein Satz zugehörige Administrator-Daten erzeugt werden.

5.3.1 Issuer

Die Daten und Datenstrukturen eines Issuers werden mittels der Klasse `CaKeys` erzeugt. Der Aufruf dieser Klasse erfolgt über das Makefile-Target `«caKeys»`. Welche Daten und Datenstrukturen erzeugt, und wie diese gespeichert werden, kann in Abschnitt 4.2.3 auf Seite 38 nachgelesen werden. Auf Seite 55 in Abschnitt 5.6.2 ist nachzulesen, welche Properties diese Klasse benötigt. Die Daten des Issuers müssen vor den Daten eines zugehörigen Administrators erzeugt werden.

5.3.2 Administrator

Die Klasse `AdminKeys` erzeugt die Daten und Datenstrukturen eines Administrators. Sie wird mittels des Makefile-Targets `«adminKeys»` gestartet. In Abschnitt 4.2.3 auf Seite 38 ist beschrieben welche Daten und Datenstrukturen erzeugt und wie diese abgespeichert werden. Eine Beschreibung der benutzten Properties ist in Abschnitt 5.6.3 auf Seite 56 zu finden. Die Erzeugung der Administrator-Daten kann nur erfolgen, wenn die Daten des zugehörigen Issuers bereits vorliegen.

5.4 Alltäglicher Betrieb

5.4.1 Start

Der Start der Applikation erfolgt am günstigsten mittels den im Makefile definierten Targets, die in Abschnitt 5.5 ab Seite 52 beschrieben sind. Auf diese werde ich mich auch nachfolgend beziehen, wenn ich den Start der Services erkläre. Da einige Services auf andere Services angewiesen sind, ergibt sich die im folgenden beschriebene optimale Startreihenfolge. Die Services können auch in einer davon abweichenden Reihenfolge gestartet werden. In diesem Fall wartet ein zu früh gestarteter Service auf den von ihm benötigten und vervollständigt sein Startup bei dessen Erscheinen. Dies kann (besonders bei Start der Applikation auf einem einzelnen Rechner) zu kurzzeiti-

5 BETRIEB

gen Belastungsspitzen führen. Erscheint ein benötigter Service zu spät in System, so beenden sich auch der auf ihn wartende.

Lookup-Service

Laut Jini-Philosophie³ gehört das Vorhandensein eines Lookup-Service zur Default-Infrastruktur eines Netzwerkes. Da dies aber nicht garantiert (und in der Praxis meist nicht der Fall) ist, habe ich die Möglichkeit einen Lookup-Service zu starten mit in das Makefile aufgenommen. Das aufzurufende Target lautet «lus».

Singletons

Es gibt eine Reihe von Services, von den jeweils nur eine einzige Instanz der gleichen Klasse zur selben Zeit im System vorhanden sein darf. Dies sind alle Stock-, Logger- und Queue-Services. Diese sollten zuerst gestartet werden. Das Target «sgl» ruft diese in sinnvoller Reihenfolge auf.

Processors

Von den Processors dürfen beliebig viele Instanzen jeder Klasse vorhanden sein. Das Target «prc» startet jeweils eine Instanz von CaCoreProcessor und Pkcs7Exporter. Diese können auch einzeln mittels «ccProc» bzw. «p7Exp» gestartet werden.

Importer

Um schließlich auch Processables ins System bringen zu können, wird noch (mindestens) ein Importer benötigt. Das Target «p7Imp» startet einen Pkcs7Importer. Es dürfen beliebig viele Instanzen jeder Importer-Klasse gleichzeitig gestartet werden.

5.4.2 Betrieb

Registration-Authority

Damit die Kern-CA⁴ Arbeit hat, müssen Processables ins System gebracht werden. Diese werden von (mindestens) einer Registration-Authority (RA) erzeugt und in einem vom Importer akzeptierten Format in den Bereich der Kern-CA gebracht. Entsprechend werden abgearbeitete Processables in einem geeigneten Format aus dem Bereich der Kern-CA heraustransportiert. Im folgenden wird davon ausgegangen, daß frische Processables geliefert und bearbeitete abgeholt werden. Die Kern-CA selber kümmert sich darum nicht.

³[Mic00d]

⁴siehe Abschnitt 8 auf Seite 11

Auto-Mode

Alle `Processors` (konkret `CaCoreProcessor`, `Pkcs7Importer` und `Pkcs7Exporter`) können im interaktiven oder im automatischen Modus gestartet werden. In welchem Modus die jeweilige Instanz startet, wird durch einen entsprechenden Kommandozeilenparameter festgelegt. Im interaktiven Modus wird der Benutzer vor wichtigen Schritten (z.B. Ausstellung eines Zertifikates, Revokation eines Zertifikates, ...) um Zustimmung zu deren Durchführung gebeten. Im automatischen Modus, wird der Benutzer lediglich darüber informiert. Für den `Pkcs7Importer` gibt es noch weitere Auswirkungen. Wird dieser im «autoMode» gestartet, so lädt er ein einziges mal ein PKCS#7-Security-File (dessen Name er als Kommandozeilenparameter bekommen hat), entpackt die darin enthaltenen `Processables`, startet diese und beendet sich. Im interaktiven Modus kann der Benutzer das Laden verschiedener PKCS#7-Security-Files beliebig oft anstoßen. Der `Pkcs7Importer` muß dann explizit beendet werden.

GUI-Mode

`Processors` können mit graphischer Oberfläche oder im Kommandozeilenmodus gestartet werden. Der Modus, in dem die Instanz laufen soll, wird ihr beim Start per Kommandozeilenparameter mitgeteilt. Bei der Arbeit mit graphischer Oberfläche werden dem Benutzer die Daten in einem `Frame` angezeigt, über den im interaktiven Modus auch die Benutzereingaben entgegengenommen werden. Im Kommandozeilenmodus werden die Daten auf die Standard-Ausgabe geschrieben bzw. von der Standard-Eingabe gelesen. Auto-Mode und GUI-Mode sind voneinander unabhängig. Alle Kombinationen sind möglich.

5.4.3 Beendigung

Momentan gibt es noch keine Möglichkeit das System von zentraler Stelle aus geordnet herunterzufahren. Jeder einzelne Service kann jedoch regulär beendet werden. Um also die Applikation zu beenden, ist es nötig jeden Service einzeln zu stoppen. Dies geschieht am besten in der umgekehrten optimalen Startreihenfolge.⁵ Es ist zu beachten, daß die Beendigung eines Service im Augenblick den sofortigen Abbruch all seiner Aktivitäten zur Folge hat. Gerade dort in Arbeit befindliche `Processables` gehen verloren und müssen beim nächsten Systemstart neu importiert werden. Dies wird sich ändern, wenn dem System transaktionales Verhalten hinzugefügt wird.

⁵siehe Abschnitt 5.4.1 auf Seite 49

5.5 Das Makefile

Das Makefile dient dem vereinfachten Aufruf von Compilierungs- und Startvorgängen. Compilierung und Start der Applikation sind auch ohne das Makefile möglich, gestalten sich dann aber erheblich komplizierter.

5.5.1 Variablen

Dieser Abschnitt beschreibt die durch den Benutzer anpassbaren Variablen des Makefiles. Es existieren weitere Variablen, die aber vom Benutzer nicht geändert werden sollten.

lusHost: Wird dieser Variablen ein Wert zugewiesen, so wird dieser dem `JiniHelper` (bei dessen Start) als URL des Lookup-Service übergeben. Dies ist notwendig, wenn der LUS statt wie üblich mittels Multicast-Paketen mit Unicast-Paketen kontaktiert werden muß. Eine Notwendigkeit dazu besteht, wenn das Subnetz nicht Multicast-fähig ist, oder sich die Applikation über mehrere Subnetze verteilt.⁶

policy: Hier wird festgelegt nach welcher Policy der RMI-Dämon⁷ und der Lookup-Service⁸ arbeiten sollen. Gültige Policy-Files befinden sich im Verzeichnis «policy» unterhalb des Root-Verzeichnisses.

guiMode: Die Applikation kann mit oder ohne graphische Oberfläche gestartet werden. Bei «guiMode=true» wird mit, bei «GuiMode=false» ohne GUI gestartet.

autoMode: Die CA kann automatisch oder interaktiv arbeiten. «autoMode=true» stellt den automatischen Modus, «autoMode=false» den interaktiven Modus ein.

propFile: Diese Variable enthält den Namen des zu verwendenden Properties-File. Näheres zu den Properties kann man in Abschnitt 5.6 auf Seite 55 nachlesen.

logLevel: Hier kann der initiale Level des `CaCoreLoggers` eingestellt werden. Erlaubte Werte sind: 1 (silent), 2 (normal), und 3 (verbose).

logWriter: Der vom `CaCoreLogger` benutzte `Writer` wird über diese Variable bestimmt. Hier wird der voll qualifizierte Name einer von `Writer` erbedenden Klasse erwartet.

5.5.2 Targets

In diesem Abschnitt werden die wichtigsten im Makefile definierten Targets erklärt. Zunächst wird auf Compileraufrufe eingegangen, danach auf Startvorgänge.

⁶Multicast-Pakete verlassen ihr Subnetz normalerweise nicht

⁷[Mic00f]

⁸[Mic00d]

Compileraufrufe

javac: Beim Aufruf dieses Targets werden mittels des Java-Compilers alle Quellen zu Klassen kompiliert.

rmic: Dies stößt die Erzeugung aller Stubs und Skeletons durch den RMI-Compiler an.

cls: Hiermit werden zuerst alle Klassen und danach alle Stubs und Skeletons erzeugt. Der Aufruf dieses Targets entspricht dem sequentiellen Aufruf der beiden vorgenannten Targets.

doc: Erzeugt die Javadocs aller Klassen mittels des Javadoc-Compilers.

Startvorgänge

caKeys: Startet die Klasse `CaKeys`. Diese erzeugt alle für eine Issuer-Instanz exklusiven Daten.

adminKeys: Bewirkt die Generierung aller für eine Administrator-Instanz spezifischen Daten mittels Aufruf der `AdminKeys`-Klasse. Es ist zu beachten, daß zuvor mindestens ein Satz Issuer-Daten (siehe vorherigen Punkt) erzeugt wurde, da diese zur Erzeugung der Administrator-Daten benötigt werden.

ini: Entspricht der aufeinanderfolgenden Ausführung der Targets «`caKeys`» und «`adminKeys`».

rmid: Dieses Target startet den RMI-Dämon, welcher von `Reggie` (siehe nächsten Punkt) benötigt wird.

reggie: Hierdurch wird `Reggie`, die Implementierung eines Jini-Lookup-Service von SUN Microsystems, aufgerufen. Da `Reggie` ein aktivierbarer RMI-Server ist, benötigt er einen laufenden RMI-Dämon (siehe vorherigen Punkt) um starten zu können.

lus: Führt zuerst das Target «`rmid`» und danach «`reggie`» aus.

logger: Startet den `CaCoreLogger`. Dies sollte immer der erste gestartete `FlexiTRUST`-Service sein, da alle anderen `FlexiTRUST`-Klassen darauf zugreifen. Von jeder `Logger`-Klasse darf immer nur eine Instanz im System existieren.

caStock: Als zweiter `FlexiTRUST`-Service sollte immer der `CaStock` gestartet werden. Mit Ausnahme des `CaCoreLoggers` wird er von allen übrigen `FlexiTRUST`-Klassen benutzt. Es ist nicht erlaubt mehrere `CaStock`-Instanzen gleichzeitig zu starten.

5 BETRIEB

- procQueue:** Ruft die Klasse `CaCoreProcessorQueue` auf. Dieser Service sollte vor allen `CaCoreProcessor`-Instanzen existieren, da letztere auf ihn zugreifen. Im System darf es zu einer Zeit maximal eine `CaCoreProcessorQueue`-Instanz geben.
- p7ExpQueue:** Bewirkt die Ausführung von `Pkcs7ExporterQueue`. Da alle `Pkcs7Exporter`-Objekte darauf zugreifen, ist darauf zu achten die `Pkcs7ExporterQueue` vor diesen zu starten. Es macht keinen Sinn mehr als eine Instanz von `Pkcs7ExporterQueue` innerhalb einer CA zu haben.
- sgl:** Führt die Targets `«logger»`, `«caStock»`, `«procQueue»` und `«p7ExpQueue»` in genannter Reihenfolge aus, d.h. startet alle Singleton-Services in geeigneter Reihenfolge.
- ccProc:** Startet die Klasse `CaCoreProcessor`. Diese greift bei ihrer Arbeit auf den `CaCoreProcessorQueue`-Service zu und sollte deshalb nach diesem aufgerufen werden. Es können beliebig viele Instanzen von `CaCoreProcessor` gleichzeitig vorhanden sein.
- p7Exp:** Dieses Target dient der Instanziierung eines `Pkcs7Exproter`-Objektes. Da dieses auf den `Pkcs7ExporterQueue`-Service zugreift, empfiehlt es sich, diesen zuvor zu starten.
- prc:** Entspricht der sequentiellen Ausführung der Targets `«ccProc»` und `«p7Exp»`.
- p7Imp:** Hiermit wird die Klasse `Pkcs7Importer` gestartet. Es können beliebig viele `Pkcs7Importer`-Instanzen gleichzeitig im System vorhanden sein.
- tst:** Testet die Klassen des FlexiTRUST-Systems. Dazu muß die CA bis auf den `Pkcs7Importer` vollständig gestartet sein. Es werden drei PKCS#7-Pakete mit `Processables` (eines mit `X509CertProcessables`, eines mit `X509CrlProcessables` und eines mit `KeyGenProcessables`) geschnürt. Für jedes Paket wird (sequentiell) ein eigener `Pkcs7Importer` gestartet. Dieser liest sein Paket ein und startet die darin enthaltenen `Processables`.
- all:** Initialisiert und startet das FlexiTRUST-System. Danach wird der im vorherigen Punkt beschriebene Test durchgeführt. Dies geschieht durch den sequentiellen Aufruf der Targets `«ini»`, `«env»`, `«prc»` und `«tst»`. Bei Ausführung diese Targets werden also alle Klassen des Systems getestet.⁹

⁹die Klassen der Initialphase führen Selbsttests durch

5.6 Die Properties

Im Properties-File werden die vom System benötigten (Default-)Properties eingetragen. Es wird ein Properties-File namens «properties» mitgeliefert. Es ist zu empfehlen die dort eingetragenen Werte unverändert zu lassen, und abweichende Parameter per Kommandozeile zu übergeben. Der Name des Properties-Files ist variabel, und wird den betreffenden Klassen per Kommandozeilenparameter mitgeteilt. Im folgenden werden die Properties nach Gruppen sortiert vorgestellt. Wenn nicht anders vermerkt wird eine JCA-konforme¹⁰ Syntax erwartet.

5.6.1 Global

Die in diesem Abschnitt vorgestellten Properties gelten global für alle Klassen.

keyStoreType: Typ der verwendeten KeyStores.

keyStoreProvider: Provider der verwendeten KeyStores.

securityProvider: Voll qualifizierter Klassenname des erwünschten Security-Providers.

5.6.2 CaKeys

Hier werden die von der Klasse `CaKeys` benötigten Parameter vorgestellt. Diese werden zum Teil auch von der Klasse `AdminKeys` benutzt¹¹. Die Klasse `CaKeys` benutzt auch einige der Parameter, die der Certification-Authority zugeordnet sind.

caKeys_KeyAlg: Algorithmus für den ein Schlüsselpaar erzeugt werden soll.

caKeys_KeyLength: Länge der zu erzeugenden Schlüssel in Bit.

caKeys_IssuerName: Name des neuen Issuers (für den ein Schlüsselpaar erzeugt wird) als RFC1779-String.

caKeys_CertFileName: String, der den Namen des zu erstellenden Zertifikats-Files enthält.

caKeys_SecFileName: String, der den Namen des zu erstellenden PKCS#12-Security-Files enthält.

caKeys_SecFilePass: Passwort zum Schutz des PKCS#12-Security-Files.

¹⁰[Knu98]

¹¹z.B. um an den Issuer-Schlüssel zu gelangen

5.6.3 AdminKeys

Dies sind die der Klasse `AdminKeys` zugeordneten Properties. Außer diesen benutzt `AdminKeys` auch einige Parameter, die der Klasse `CaKeys`, der CA und dem Administrator zugeordnet sind.

adminKeys_KeyAlg: Algorithmus für den ein Schlüsselpaar erzeugt werden soll.

adminKeys_KeyLength: Länge der zu erzeugenden Schlüssel in Bit.

adminKeys_IssuerName: Name des neuen Administrators (für den ein Schlüsselpaar erzeugt wird) als RFC1779-String.

adminKeys_CertFileName: String, der den Namen des zu erstellenden Zertifikats-Files enthält.

adminKeys_SecFileName: String, der den Namen des zu erstellenden PKCS#12-Security-Files enthält.

adminKeys_SecFilePass: Passwort zum Schutz des PKCS#12-Security-Files.

5.6.4 Certification-Authority

Hier werden die Parameter erklärt, welche zur CA (bzw. zu deren einzigen Issuer) gehören.

ca_SerialFileName: Name des Files, der die nächste von der CA zu verwendende Seriennummer enthält.

ca_CertsListFileName: Name des Files, der die Liste aller erstellten Zertifikate enthält.

ca_ComplCrlFileName: Name des Files, der die komplette Revokationsliste enthält.

ca_LeanCrlFileName: Name des Files, der die schlanke Revokationsliste enthält.

ca_CrlPeriod: Anzahl der Tage nach denen spätestens eine neue Revokationsliste erscheint.

ca_KeyStoreFileName: Name des Keystore-Files.

ca_Pass: Passwort für `KeyStore` und privaten Schlüssel.

ca_Alias: Alias für privaten Schlüssel und das Zertifikat des Issuers.

ca_AdminAlias: Alias für das Zertifikat des Administrators.

ca_SigAlg: Signatur-Algorithmus des Issuers.

5.6.5 Administrator

Nachfolgend aufgeführte Properties werden vom (bisher einzigen) Administrator benutzt.

ad_SerialFileName: Name des Files, der die nächste vom zu Administrator zu verwendende Seriennummer enthält.

ad_KeystoreFileName: Name des Keystore-Files des Administrators.

ad_Pass: Passwort für `Keystore` und privaten Schlüssel.

ad_Alias: Alias für privaten Schlüssel und das Zertifikat des Issuers.

ad_SigAlg: Signatur-Algorithmus des Administrators.

5.6.6 Logging

Diese Parameter werden vom Logging-Mechanismus benötigt.

lw_LogFileName: Name des zu erstellenden Log-Files.

5.7 Erweiterung

In diesem Abschnitt wird erklärt, was zu tun ist, um das System zu erweitern. Dazu geben wir uns eine neue Aufgabe vor, und erklären anhand dieses Beispiels was getan werden muß, um sie dem System hinzuzufügen.

Die neue Aufgabe der CA sei es, aus gegebenem Zertifikat und privatem Schlüssel ein PKCS#12-Security-Paket zu erzeugen. PKCS#12 ist ein von der CA noch nicht unterstützter Standard. Wir überlegen uns, die PKCS#12-spezifischen Algorithmen in einer neuen `Processor`-Klasse zu implementieren, um diese beliebigen `Processables` zugänglich machen zu können. Zu jeder `Processor`-Klasse gehört eine zugeordnete `Queue`-Klasse (inklusive RMI-Interface). Diese müssen wir ebenfalls implementieren. Schließlich fehlt noch eine `Processable`-Klasse, welche die Rohdaten ins System bringt, und die erstellten Pakete mit nach außen nimmt. Zur Umsetzung eben genannter Punkte, gehen wir wie nachfolgend beschrieben vor.

1. Zunächst implementieren wir die neue `Queue`. Dazu kopieren wir uns den Quell-Code von `CaCoreProcessorQueue` und benennen die Klasse in `Pkcs7ProcessorQueue` um. Statt des `CaCoreProcessorQueueRmi`-Interfaces lassen das `Pkcs7ProcessorQueueRmi`-Interface beerben, welches wir aus ersterem durch Kopieren und Umbenennen erhalten haben. Falls wir nicht vergessen haben den Konstruktor ebenfalls umzubenennen, ist die neue Klasse fertig.

2. Nun folgt der `Pkcs7Processor`. Dazu beerben wir die Klasse `CaCoreProcessor`. Dies erspart uns einiges an Implementierungsarbeit, und ermöglicht es der neuen Klasse als `CaCoreProcessor` zu fungieren, wenn keine PKCS#12-spezifischen Aufgaben vorliegen. Die neue Klasse bedient sich vorrangig aus der `Pkcs7ProcessorQueue`. Liegen dort keine `Processables`, so weicht sie auf die `CaCoreProcessorQueue` aus. Es wird eine neue Methode implementiert, die aus den übergebenen Parametern ein PKCS#12-Security-Paket erzeugt und dieses zurückgibt.
3. Schließlich fehlt noch das `Pkcs7Processable`. Dieses bekommen wir durch Kopieren und Umbenennen des `KeyGenProcessables`. Wir verändern die Klasse zusätzlich so, daß sich ihre Instanzen in der `Pkcs7ProcessorQueue` einreihen. Wird ein Objekt dort von einem `Pkcs7Processor` entnommen, läßt es sich von diesem das PKCS#12-Security-Paket erstellen. Die Methode zum Verlassen des Systems kann unverändert bleiben. Natürlich müssen der Konstruktor, und die Methode zum entnehmen des erstellten Produktes angepaßt werden.

Alles was jetzt noch zu tun bleibt, ist der Start einer `Pkcs7ProcessorQueue` und beliebig vieler `Pkcs7Processors`. Diese fügen sich nahtlos in das bereits laufende System ein, ohne das sonstige Aktivitäten notwendig sind. Die bereits laufenden Services müssen weder angehalten, noch sonst irgendwie umkonfiguriert werden. Von nun an ist die CA in der Lage PKCS#12-Security-Pakete zu erstellen, wenn sie von der RA entsprechende Anträge¹² bekommt.

In diesem Beispiel wurden alle neuen Klassen durch Kopieren oder Beerben bereits bestehender Klassen implementiert. Sollte dies einmal nicht möglich sein, so können alle jini-spezifischen Aufgaben an den `JiniHelper` delegiert werden. Der Programmierer muß keine tiefen Jini-Kenntnisse besitzen, es genügt ein grobe Überblick über den Aufbau eines Jini-Systems. Ähnlich verhält es sich mit RMI. Neue Services mit Remote-Methoden beerben `UnicastRemoteObject` und implementieren ihr Remote Method Interface. Dieses erweitert das Remote-Interface und deklariert alle entfernt zugreifbaren Methoden.

5.8 Testsysteme

Das System wurde umfangreich auf unterschiedlichen Plattformen in verschiedenen Kombinationen getestet. Es wurden folgende Rechner-Systeme verwendet:

Win98: Pentium-II-PC mit Windows 98

WinNT: Pentium-II-PC mit Windows NT

¹²die eben implementierten `Pkcs12Processables`

Win2K: Pentium-II-PC mit Windows 2000

Linux: Pentium-II-PC mit Suse Linux 7.0

Solaris: Ultra-Sparc mit Solaris 7

Das Netzwerk bestand aus 3 Subnetzen. Innerhalb jedes Subnetzes waren die Rechner mittels eines 10MBit-Ethernet-Netzwerkes verbunden.

Subnetz 1: 2*Win98, 1*Linux

Subnetz 2: 2*WinNT, 1*Win2K

Subnetz 3: 2*Solaris

Subnetz 2 und 3 waren mittels eines 100MBit-Ethernet-Netzwerkes verbunden. Subnetz 1 war über das Internet mittels eines T-DSL-Anschlusses mit den anderen Subnetzen verbunden. Die CA wurde in folgenden Konfigurationen getestet:

- Jeweils auf einem einzigen Rechner mit jedem Betriebssystem.
- Verteilt auf die Rechner innerhalb jedes Subnetzes.
- Verteilt über alle Rechner aller Subnetze.

Kapitel 6

Ausblick

Der Schwerpunkt dieser Diplomarbeit lag wie bereits erwähnt auf dem Design der Software. Bei der Implementierung handelt es sich um einen Prototyp, der zwar voll lauffähig ist, sich aber an einem etwas vereinfachten Design orientiert. Nachfolgend werde ich weitere Schritte in der Entwicklung der Software aufzählen. Die Reihenfolge dieser Aufzählung muß nicht die zeitliche Reihenfolge sein, in der diese Punkte angegangen werden.

- Implementierung eines zentral gesteuerten, geordneten Shutdowns des Systems. Hierzu kann das `Synchronizable`-Interface genutzt werden. Es ist zu überlegen, wie mit den gerade in Arbeit befindlichen oder im System gespeicherten `Processables` zu verfahren ist.
- `Singleton-Service`s sollten das System auf das Vorhandensein von unerwünschten Klonen überprüfen, und für Abhilfe sorgen.
- Alle `Services` aktivierbar im Sinne des `RMI-Activation-Systems` machen.
- Implementierung der simultanen Benutzung mehrerer Issuer mit jeweils mehreren ihnen zugeordneten Administratoren.
- Um dynamischen Wechsel des `Log-Levels` zu vereinfachen sollte ein `LogLevelListener`-Interface eingeführt werden.
- Implementierung einer dynamischen Zuordnung von `Exportern` zu `Processables`. Diese könnte vom `Importer` vorgenommen werden, im `Properties-File` festgelegt sein oder über das `Synchronizable`-Interface geregelt werden.
- Unterstützung des `Peer-to-Peer-Transports` der `Processables` zwischen den `Processors`.
- Ausstattung des `AbstractProcessables` mit Methoden zum Setzen und Prüfen der eigenen Signatur.

- Da die CODEC-Klassen jetzt serialisierbar sind, die Typisierung der betroffenen Methoden anpassen.
- Es sollte ein Property eingeführt werden, welches den aktuellen Komprimierungsgrad für den Netzverkehr angibt. Eventuell könnte dies sogar eine dynamische Größe sein.
- Auch die komprimierte Speicherung der Daten kann sinnvoll sein.
- Es ist sinnvoll ein `QueueListener`-Interface zu implementieren, um bei leeren Queues nicht auf Busy-Waiting-Algorithmen angewiesen zu sein.
- Die Implementierung geeigneter Cache-Mechanismen bei den Klassen `CaStockHelper` und `LogHelper` steht noch aus.
- Das System verhält sich nicht transaktional. Um dies zu erreichen könnten der Advanced Processing Infrastructure zwei neue Konzepte «Transaction-Manager» und «Transaction» hinzugefügt werden. Zu deren Realisierung könnte dann auf die gleichnamigen Klassen des Jini-Frameworks zurückgegriffen werden.
- Die Persistenz mittels Files ist unsicher und langsam. Sie sollte mittels Java-Spaces (einer pro JVM?) oder Datenbank-Anbindung (eine DB pro Host?) realisiert werden.
- Um das Ablaufen der Applikation auf einem einzelnen Rechner zu optimieren, könnte jeder Service auch gemäß dem Singleton-Pattern¹ implementiert sein. Die `getInstance()`-Methode würde dann bei verteilter Ausführung den entsprechenden Jini-Service, bei nicht verteilter Ausführung die Singleton-Instanz liefern.
- Zwischen dem voll automatischen und dem voll interaktiven Modus könnten noch weitere Modi existieren. Beispielsweise wäre ein Stichproben-Modus denkbar.

¹das Design-Pattern, nicht die Singleton-Services

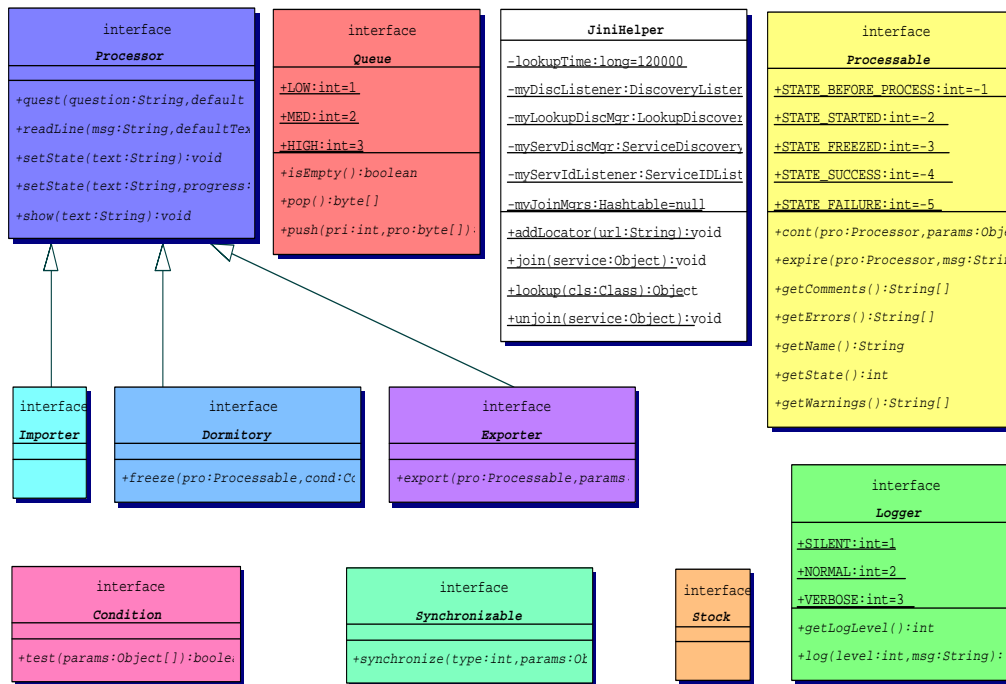
Anhang A

Javadocs



Abbildung A.1: Übersicht über alle Packages

A.1 de.tud.cdc.flexiTrust.cs.api



[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.api Interface Condition

public interface Condition

A Condition is used by a [Processable](#) to tell a [Dormitory](#) what it is waiting for when calling the [Dormitory.freeze\(Processable, Condition\)](#) method.

Version:
SId: Condition.java,v 1.10 2000/12/22 17:24:44 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

boolean	test (java.lang.Object[] params)
---------	--

Tests whether the condition is fulfilled.

Method Detail

test

```
public boolean test(java.lang.Object[] params)
    throws java.lang.Exception
    Tests whether the condition is fulfilled.
Parameters:
    params - Some parameters. The implementors must agree about type and structure of these parameters.
Returns:
    true if the condition is fulfilled, false otherwise.
Throws:
    java.lang.Exception - Implementors may throw some Exceptions.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.api Interface Dormitory

public interface Dormitory
extends [Processor](#)

In a Dormitory a given [Processable](#) is frozen until the given [Condition](#) is fulfilled. The [Condition](#) is tested regularly. If the [Condition](#) cannot be fulfilled within an adequate time the [Processable](#) will be expired.

Version:
SId: Dormitory.java,v 1.18 2000/12/22 17:24:44 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

void	freeze (Processable pro, Condition cond)
------	---

Freezes the given [Processable](#) until the [Condition.test\(Object\[\]\)](#) method returns true.

Methods inherited from interface de.tud.cdc.flexiTrust.ca.api.Processor

[quest](#), [readLine](#), [setState](#), [setState](#), [show](#)

Method Detail

freeze

```
public void freeze(Processable pro,
    Condition cond)
    throws java.lang.Exception
    Freezes the given Processable until the Condition.test\(Object\[\]\) method returns true. The Processable is expired (by calling Processable.expire\(Processor, String, Object\[\]\)) if the Condition isn't fulfilled in an adequate time.
Parameters:
    pro - The Processable to freeze.
    cond - The condition to wait for.
```

A JAVADOCS

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)

[SUMMARY](#)
[INNER](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)
[DETAIL](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Exporter

All Known Implementing Classes:
[Pkcs7Exporter](#)

public interface **Exporter**
extends [Processor](#)

Exporters are the last station [Processables](#) roam to (the export is last step in their process).

Version:
\$Id: Exporter.java,v 1.25 2000/12/22 17:24:44 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

void	export (Processable pro, java.lang.Object[] params)
	Exports the given Processable .

Methods inherited from interface de.tud.cdc.flexiTrust.ca.api.Processor

[quest](#), [readLine](#), [setState](#), [setState](#), [show](#)

Method Detail

export

```
public void export(Processable pro,
                  java.lang.Object[] params)
    throws java.lang.Exception;
    Exports the given Processable.
```

Parameters:
pro - The instance to export.
params - Some parameters. The implementors must agree about type and structure of these parameters.

Throws:
java.lang.Exception - Implementors may throw some Exceptions.

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)

[SUMMARY](#)
[INNER](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)
[DETAIL](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Importer

All Known Implementing Classes:
[Pkcs7Importer](#)

public interface **Importer**
extends [Processor](#)

An Importer imports [Processables](#) into the Advanced Processing Infrastructure. The Importer receives the [Processable](#) from somewhere and calls [Processable.cont\(Processor, Object\[\]\)](#) to start the process.

Version:
\$Id: Importer.java,v 1.8 2000/12/22 17:24:44 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Methods inherited from interface de.tud.cdc.flexiTrust.ca.api.Processor

[quest](#), [readLine](#), [setState](#), [setState](#), [show](#)

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)

[SUMMARY](#)
[INNER](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)
[DETAIL](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)

[SUMMARY](#)
[INNER](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)
[DETAIL](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Class JiniHelper

```
java.lang.Object
|
+-de.tud.cdc.flexiTrust.ca.api.JiniHelper
```

public class **JiniHelper**
extends java.lang.Object

Eases and optimizes the usage of JINI. It is necessary for [JiniHelper](#)'s task that all classes within one JVM use the same instance of [JiniHelper](#). Therefore and to ease the use of this class all [JiniHelper](#) methods are static. When this class is loaded (when first referenced) a JINI lookup for the Lookup Service is done and the LUS proxy is fetched. All further calls of [JiniHelper](#) methods (e.g. by roaming [Processables](#)) do not cause further lookups for the LUS. In addition a caching mechanism is used to optimize the lookup of frequently used services.

Version:
\$Id: JiniHelper.java,v 1.3 2001/01/17 16:18:28 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

JiniHelper ()	
-------------------------------	--

Method Summary

static void	addLocator (java.lang.String url)	Adds a LookupLocator pointing to given URL.
static void	join (java.lang.Object service)	Joins the given service to DJINN.
static java.lang.Object	lookup (java.lang.Class cls)	Looks up the given service in DJINN.
static void	unJoin (java.lang.Object service)	Unjoins the given service form DJINN.

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

JiniHelper

```
public JiniHelper()
```

Method Detail

addLocator

```
public static void addLocator(java.lang.String url)
    Adds a LookupLocator pointing to given URL.
```

Parameters:
url - The URL where the LUS runs.

join

```
public static void join(java.lang.Object service)
    Joins the given service to DJINN.
```

Parameters:
service - The service to join.

lookup

```
public static java.lang.Object lookup(java.lang.Class cls)
    Looks up the given service in DJINN.
```

Parameters:
cls - The service to look up.

unjoin

```
public static void unJoin(java.lang.Object service)
    Unjoins the given service form DJINN.
```

Parameters:
service - The service to unjoin.

[Overview](#)
[Package](#)
[Class](#)
[Tree](#)
[Deprecated](#)
[Index](#)
[Help](#)

[PREV CLASS](#)
[NEXT CLASS](#)
[FRAMES](#)
[NO FRAMES](#)

[SUMMARY](#)
[INNER](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)
[DETAIL](#)
[FIELD](#)
[CONSTR](#)
[METHOD](#)

A.1 DE.TUD.CDC.FLEXITRUST.CS.API

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Logger

All Known Subinterfaces:
[CaCoreLoggerRmi](#)

public interface **Logger**

Provides a centralized logging mechanism for the Advanced Processing Infrastructure.

Version:
 \$Id: Logger.java,v 1.22 2000/12/25 13:55:12 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Field Summary

static int	NORMAL	Mnemonic for logging mode normal.
static int	SILENT	Mnemonic for logging mode silent.
static int	VERBOSE	Mnemonic for logging mode verbose.

Method Summary

int	getLogLevel()	Returns the current log level.
void	log(int level, java.lang.String msg)	Adds the given message to log.

Field Detail

SILENT

public static final int **SILENT**
 Mnemonic for logging mode silent.

NORMAL

public static final int **NORMAL**
 Mnemonic for logging mode normal.

VERBOSE

public static final int **VERBOSE**
 Mnemonic for logging mode verbose.

Method Detail

getLogLevel

public int [getLogLevel\(\)](#)
 throws java.lang.Exception
 Returns the current log level. Possible log levels are described in [Field Summary](#).
Returns:
 The current log level.
Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

log

public void [log\(int level, java.lang.String msg\)](#)
 throws java.lang.Exception
 Adds the given message to log.
Parameters:
 level - The level of the given message. If the message's level is greater than this Logger's level it will not be logged. Possible levels are described in [Field Summary](#).
 msg - The message to log.
Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Processable

All Known Implementing Classes:
[AbstractProcessable](#)

public interface **Processable**

All instances to be processed by a [Processor](#) must implement this interface. A [Processable](#) roams from one [Processor](#) to another to execute his tasks on them. A [Processable](#)'s process normally consists of several steps which might be done on different [Processors](#). Then all steps are coded within one [Processable](#) class, and a stepcounter is used to determine the next step. It is a good idea to increment the step counter within the [cont\(Processor, Object\[\]\)](#) method to ensure the accurate incrementation of the step counter. The [Processable](#) is responsible for the accurate workflow and error handling.

Version:
 \$Id: Processable.java,v 1.34 2000/12/25 13:55:12 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Field Summary

static int	STATE_BEFORE_PROCESS	Mnemonic for state: Processable ready for processing.
static int	STATE_FAILURE	Mnemonic for state: Processable could not be processed due to some error.
static int	STATE_FROZZED	Mnemonic for state: Processable in process but waiting for something.
static int	STATE_STARTED	Mnemonic for state: Processable in process.
static int	STATE_SUCCESS	Mnemonic for state: Processable successfully processed.

Method Summary

void	cont(Processor pro, java.lang.Object[] params)	Continues the process.
void	expire(Processor pro, java.lang.String msg, java.lang.Object[] params)	Expires the process.

java.lang.String[]	getComments()	Returns all comments.
java.lang.String[]	getErrors()	Returns all errors.
java.lang.String	getName()	Returns the name of the current object.
int	getState()	Returns the current state of process.
java.lang.String[]	getWarnings()	Returns all warnings.

Field Detail

STATE_BEFORE_PROCESS

public static final int **STATE_BEFORE_PROCESS**
 Mnemonic for state: Processable ready for processing.

STATE_STARTED

public static final int **STATE_STARTED**
 Mnemonic for state: Processable in process.

STATE_FROZZED

public static final int **STATE_FROZZED**
 Mnemonic for state: Processable in process but waiting for something.

STATE_SUCCESS

public static final int **STATE_SUCCESS**
 Mnemonic for state: Processable successfully processed.

STATE_FAILURE

public static final int **STATE_FAILURE**
 Mnemonic for state: Processable could not be processed due to some error.

Method Detail

A JAVADOCS

cont

```
public void cont(Processor pro,
               java.lang.Object[] params)
    throws java.lang.Exception
```

Continues the process. This method is called by a [Processor](#) when this [Processable](#) is scheduled for being processed.

Parameters:
 pro - The current [Processor](#).
 params - Some parameters. The implementors must agree about type and structure of these parameters.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

expire

```
public void expire(Processor pro,
                  java.lang.String msg,
                  java.lang.Object[] params)
    throws java.lang.Exception
```

Expires the process. Called when an unrecoverable error occurred, or called by a [Dormitory](#) when the [Processable](#) is frozen too long (while waiting for a [Condition](#) to be fulfilled). Some cleanup is done and the [Processable](#) exports itself with an adequate error message.

Parameters:
 pro - The current [Processor](#).
 msg - A message describing the cause of expiration.
 params - Some parameters. The implementors must agree about type and structure of these parameters.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

getComments

```
public java.lang.String[] getComments()
    throws java.lang.Exception
```

Returns all comments. A comment is an assessment free message.

Returns:
 All comments.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

getErrors

```
public java.lang.String[] getErrors()
    throws java.lang.Exception
```

Returns all errors. An error is a message about some occurrence which causes the process to fail.

Returns:
 All errors.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

getName

```
public java.lang.String getName()
    throws java.lang.Exception
```

Returns the name of the current object.

Returns:
 The name of the current object.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

getState

```
public int getState()
    throws java.lang.Exception
```

Returns the current state of process. Possible states are defined in [Field Summary](#).

Returns:
 The current state.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

getWarnings

```
public java.lang.String[] getWarnings()
    throws java.lang.Exception
```

Returns all warnings. A warning is a message about some occurrence which usually should not happen, but does not cause the fail of the process.

Returns:
 All warnings.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Processor

All Known Subinterfaces:
[Dormitory](#), [Exporter](#), [Importer](#)

All Known Implementing Classes:
[CaCoreProcessor](#)

public interface Processor

Processors are able to process [Processables](#) by calling [Processable.cont\(Processor, Object\[\]\)](#). The [Processables](#) roam from one [Processor](#) to another to execute their tasks on them.

Version:
 \$Id: Processor.java,v 1.24 2000/12/28 18:15:14 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

boolean	quest (java.lang.String question, boolean defaultAnswer)	Displays the given question and returns the answer.
java.lang.String	readLine (java.lang.String msg, java.lang.String defaultText)	Displays the given message and reads a line of text.
void	setState (java.lang.String text)	Displays the given text as the current state
void	setState (java.lang.String text, int progress)	Displays the given text as the current state and the given progress as current progress.
void	show (java.lang.String text)	Displays the given text.

Method Detail

quest

```
public boolean quest(java.lang.String question,
                    boolean defaultAnswer)
    throws java.lang.Exception
```

Displays the given question and returns the answer.

Parameters:
 question - The question to be displayed.
 defaultAnswer - The default result (used in auto mode).

Returns:
 The answer.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

readLine

```
public java.lang.String readLine(java.lang.String msg,
                                java.lang.String defaultText)
    throws java.lang.Exception
```

Displays the given message and reads a line of text.

Parameters:
 msg - The message to display.
 defaultText - The default result (used in auto mode).

Returns:
 The line of text read in.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

setState

```
public void setState(java.lang.String text)
    throws java.lang.Exception
```

Displays the given text as the current state

Parameters:
 text - The current state.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

setState

```
public void setState(java.lang.String text,
                    int progress)
    throws java.lang.Exception
```

Displays the given text as the current state and the given progress as current progress.

Parameters:
 text - The current state.
 progress - The current progress.

Throws:
 java.lang.Exception - Implementors may throw some Exceptions.

A.1 DE.TUD.CDC.FLEXITRUST.CS.API

show

```
public void show(java.lang.String text)
    throws java.lang.Exception
    Displays the given text.
Parameters:
    The - text to display.
Throws:
    java.lang.Exception - Implementors may throw some Exceptions.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Queue

All Known Subinterfaces:
[CaCoreProcessorQueueRmi](#), [Pkcs7ExporterQueueRmi](#)

public interface **Queue**

For each type of [Processor](#) there is an associated queue.

Version:
SId: Queue.java,v 1.39 2000/12/28 18:15:14 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Field Summary

static int	HIGH	Mnemonic for low priority.
static int	LOW	Mnemonic for low priority.
static int	MED	Mnemonic for low priority.

Method Summary

boolean	isEmpty()	Returns wether the queue is empty or not.
byte[]	pop()	Dequeues the next Processable .
void	push(int pri, byte[] pro)	Enqueues a given Processable .

Field Detail

LOW

```
public static final int LOW
```

Mnemonic for low priority.

MED

```
public static final int MED
    Mnemonic for low priority.
```

HIGH

```
public static final int HIGH
    Mnemonic for low priority.
```

Method Detail

isEmpty

```
public boolean isEmpty()
    throws java.lang.Exception
    Returns wether the queue is empty or not.
Returns:
    true if the queue is empty, false otherwise.
Throws:
    java.lang.Exception - Implementors may throw some Exceptions.
```

pop

```
public byte[] pop()
    throws java.lang.Exception
    Dequeues the next Processable.
Returns:
    The instance to process next.
Throws:
    java.lang.Exception - Implementors may throw some Exceptions.
```

push

```
public void push(int pri,
    byte[] pro)
    throws java.lang.Exception
    Enqueues a given Processable. Possible priorities are described in Field Summary.
Parameters:
    pri - The priority of given Processable
    pro - The instance to enqueue for processing.
Throws:
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Stock

All Known Subinterfaces:
[CaStockRmi](#)

public interface **Stock**

Manages centralized resources.

Version:
SId: Stock.java,v 1.9 2000/12/22 17:24:45 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

A JAVADOCS

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.api
Interface Synchronizable

public interface **Synchronizable**

Provides a way to synchronize objects in the Advanced Processing Infrastructure.

Version:
SId: Synchronizable.java,v 1.7 2000/12/22 17:24:45 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

void synchronize (int type, java.lang.Object[] params)
--

Causes the object to synchronize with system.

Method Detail

synchronize

public void **synchronize**(int type,
 java.lang.Object[] params)
 throws java.lang.Exception

Causes the object to synchronize with system. The implementors must agree about synchronization types and parameters.

Parameters:
type - Determines the type of synchronisation. The implementors must agree about synchronization types.
params - Some parameters. The implementors must agree about type and structure of these parameters.

Throws:
java.lang.Exception - Implementors may throw some Exceptions.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

Methods inherited from class java.awt.Window

addWindowListener, applyResourceBundle, applyResourceBundle, dispose, getFocusOwner, getInputContext, getLocale, getOwnedWindows, getOwner, getToolkit, getWarningString, hide, isShowing, pack, postEvent, processEvent, removeWindowListener, setCursor, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFont, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processMouseEvent, processMouseMotionEvent, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CaCoreFrame

```
public CaCoreFrame(boolean autoMode,
                  java.lang.String title,
                  CaCoreMonitor moni)
    Constructs a new instance.
Parameters:
    autoMode - Determines whether to run in auto mode or not.
    title - The title of the frame.
    moni - The calling instance.
```

Method Detail

finalize

```
public void finalize()
    Destroys the instance.
Overrides:
    finalize in class java.awt.Frame
```

quest

```
public boolean quest(java.lang.String text,
                    boolean defaultResult)
    Displays the given question and returns the answer.
Parameters:
    text - The question to be displayed.
    defaultResult - The default result (used in auto mode).
Returns:
    The answer.
```

readLine

```
public java.lang.String readLine(java.lang.String text,
                               java.lang.String defaultResult)
    Displays the given message and reads a line of text.
Parameters:
    text - The message to display.
    defaultResult - The default result (used in auto mode).
Returns:
    The line of text read in.
```

setState

```
public void setState(java.lang.String text)
    Displays the given text as the current state
Parameters:
    text - The current state.
```

setState

```
public void setState(java.lang.String text,
                    int progress)
    Displays the given text as the current state and the given progress as current progress.
Parameters:
    text - The current state.
    progress - The current progress.
```

show

```
public void show(java.lang.String text)
    Displays the given text.
Parameters:
    the - text to display.
```

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.app
Class CaCoreMonitor

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.app.CaCoreMonitor
```

public class CaCoreMonitor
extends java.lang.Object

Provides a mechanism for graphical and textual interaction with the user.

Version:
\$Id: CaCoreMonitor.java,v 1.73 2000/12/28 18:15:14 wiesmaier Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

```
CaCoreMonitor(boolean guiMode, boolean autoMode, java.lang.String title,
              CaCoreProcessor proc)
    Constructs a new instance.
```

Method Summary

void	finalize ()	Destroys the instance.
boolean	quest (java.lang.String text, boolean defaultResult)	Displays the given question and returns the answer.
java.lang.String	readLine (java.lang.String text, java.lang.String defaultResult)	Displays the given message and reads a line of text.
void	setState (java.lang.String text)	Displays the given text as the current state
void	setState (java.lang.String text, int progress)	Displays the given text as the current state and the given progress as current progress.
void	show (java.lang.String text)	Displays the given text.

Methods inherited from class java.lang.Object

```
clone, equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

CaCoreMonitor

```
public CaCoreMonitor(boolean guiMode,
                    boolean autoMode,
                    java.lang.String title,
                    CaCoreProcessor proc)
    Constructs a new instance.
    Parameters:
    guiMode - Indicates the GUI mode.
    autoMode - Indicates the aut mode.
    title - The title of this monitor.
    proc - The calling instance.
```

Method Detail

finalize

```
public void finalize()
    Destroys the instance.
    Overrides:
    finalize in class java.lang.Object
```

quest

```
public boolean quest(java.lang.String text,
                    boolean defaultResult)
    Displays the given question and returns the answer.
    Parameters:
    text - The question to be displayed.
    defaultResult - The default result (used in auto mode).
    Returns:
    The answer.
```

readLine

```
public java.lang.String readLine(java.lang.String text,
                                java.lang.String defaultResult)
    Displays the given message and reads a line of text.
    Parameters:
    text - The message to display.
```

```
defaultResult - The default result (used in auto mode).
Returns:
The line of text read in.
```

setState

```
public void setState(java.lang.String text)
    Displays the given text as the current state.
    Parameters:
    text - The current state.
```

setState

```
public void setState(java.lang.String text,
                    int progress)
    Displays the given text as the current state and the given progress as current progress.
    Parameters:
    text - The current state.
    progress - The current progress.
```

show

```
public void show(java.lang.String text)
    Displays the given text.
    Parameters:
    text - text to display.
```

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL: FIELD](#) [CONSTR](#) [METHOD](#)

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL: FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.app
Class CaCoreProcessor

```
java.lang.Object
|-- java.lang.Thread
   |-- de.tud.cdc.flexiTrust.ca.app.CaCoreProcessor
```

public class CaCoreProcessor
 extends java.lang.Thread
 implements Processor

Implements some cryptographic core functionality.

Version:
 \$Id: CaCoreProcessor.java,v 1.151 2001/01/07 15:51:19 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

```
CaCoreProcessor(boolean guiMode, boolean autoMode, java.lang.String title)
    Constructs a new instance.
```

Method Summary

```
void finalize()
    Destroys the instance.
java.security.KeyPair generateKeys(java.lang.String keyAlg, int keyLength)
    Generates a key pair for given algorithm with given key length.
static void main(java.lang.String[] args)
    Invoked by runtime system when this class is executed.
boolean quest(java.lang.String text, boolean defaultResult)
    Displays the given question and returns the answer.
java.lang.String readLine(java.lang.String text, java.lang.String defaultResult)
```

```
Displays the given message and reads a line of text.
void run()
    Looks up the CaCoreProcessorQueue and pops Processable's from it.
void setState(java.lang.String text)
    Displays the given text as the current state.
void setState(java.lang.String text, int progress)
    Displays the given text as the current state and the given progress as current progress.
void show(java.lang.String text)
    Displays the given text.
byte[] sign(byte[] msg, java.security.Principal issuer)
    Signs a binary message.
byte[] sign(byte[] msg, java.security.PrivateKey privKey, java.lang.String sigAlg, java.lang.String secProv)
    Signs a binary message.
void sign(codec.pkcs7.SignedData sd, java.security.Principal issuer)
    Signs the data found in given SignedData object.
void sign(codec.x509.X509Certificate cert, java.security.Principal issuer)
    Signs the given certificate.
void sign(codec.x509.X509Crl crl)
    Signs the given certificate revocation list.
byte[] updateCrl(java.util.Vector crlEntries, java.util.Calendar nextUpdate)
    Adds the given CrlEntries to crl.
void verify(byte[] msg, byte[] sig, java.security.Principal issuer)
    Verifies a binary message.
void verify(byte[] msg, byte[] sig, java.security.PublicKey pubKey, java.lang.String sigAlg, java.lang.String secProv)
    Verifies a binary message.
void verify(codec.pkcs7.SignedData sd)
    Verifies the data found in the given SignedData object.
void verify(codec.x509.X509Certificate cert)
    Verifies the given certificate.
void verify(codec.x509.X509Crl crl)
    Verifies the given certificate revocation list.
```

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

A JAVADOCS

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CaCoreProcessor

```
public CaCoreProcessor(boolean guiMode,
                      boolean autoMode,
                      java.lang.String title)
    Constructs a new instance.
Parameters:
    guiMode - Determines whether this instance has a GUI.
    title - The title of the frame, if to be created.
```

Method Detail

finalize

```
public void finalize()
    Destroys the instance.
Overrides:
    finalize in class java.lang.Object
```

generateKeys

```
public java.security.KeyPair generateKeys(java.lang.String keyAlg,
                                         int keyLength)
    Generates a key pair for given algorithm with given key length.
Parameters:
    keyAlg - The Algorithm to generate a key pair for.
    keyLength - How may bits the new key will have.
```

main

```
public static void main(java.lang.String[] args)
    Invoked by runtime system when this class is executed. Creates a new instance of this
    class, starts it and waits for user's command to end the service.
Parameters:
    args - The command line arguments.
```

quest

```
public boolean quest(java.lang.String text,
                    boolean defaultResult)
    Displays the given question and returns the answer.
Specified by:
    quest in interface Processor
Parameters:
    text - The question to be displayed.
    defaultResult - The default result (used in auto mode).
Returns:
    The answer.
```

readLine

```
public java.lang.String readLine(java.lang.String text,
                                java.lang.String defaultResult)
    Displays the given message and reads a line of text.
Specified by:
    readLine in interface Processor
Parameters:
    text - The message to display.
    defaultResult - The default result (used in auto mode).
Returns:
    The line of text read in.
```

run

```
public void run()
    Looks up the CaCoreProcessorQueue and popps Processable's from it. Invoked by
    runtime system when this Thread is started.
Overrides:
    run in class java.lang.Thread
```

setState

```
public void setState(java.lang.String text)
    Displays the given text as the current state
Specified by:
    setState in interface Processor
Parameters:
    text - The current state.
```

setState

```
public void setState(java.lang.String text,
                    int progress)
    Displays the given text as the current state and the given progress as current progress.
Specified by:
    setState in interface Processor
Parameters:
    text - The current state.
    progress - The current progress.
```

show

```
public void show(java.lang.String text)
    Displays the given text.
Specified by:
    show in interface Processor
Parameters:
    text - text to display.
```

sign

```
public byte[] sign(byte[] msg,
                  java.security.Principal issuer)
    Signs a binary message.
Parameters:
    msg - The message to sign.
    issuer - The desired signer.
Returns:
    The generated signature.
Throws:
    java.lang.Exception - If signing failed.
```

sign

```
public byte[] sign(byte[] msg,
                  java.security.PrivateKey privKey,
                  java.lang.String sigAlg,
                  java.lang.String secProv)
    Signs a binary message.
Parameters:
    msg - The message to sign.
    privKey - The PrivateKey to sign with.
    sigAlg - The algorithm to sign with.
    sigProv - The provider of the engine to sign with. Giving null leads to using the
    default provider.
```

Returns:
The generated signature.
Throws:
java.lang.Exception - If signing failed.

sign

```
public void sign(codec.pkcs7.SignedData sd,
                java.security.Principal issuer)
    throws java.lang.Exception
    Signs the data found in given SignedData object. Adds a signerInfo with a valid
    signature and the certificate notarizing the authenticity of the used key to given
    SignedData object.
Parameters:
    sd - The instance to sign.
    si - The desired signer.
Throws:
    java.lang.Exception - If signing failed.
```

sign

```
public void sign(codec.x509.X509Certificate cert,
                java.security.Principal issuer)
    throws java.lang.Exception
    Signs the given certificate. All issuer dependent fields and the serial number are set
    to correct values before signing. All other values have to be set to correct values by the
    method's caller before calling this method.
Parameters:
    cert - The certificate to sign.
    issuer - The desired signer.
Throws:
    java.lang.Exception - If signing failed.
```

sign

```
public void sign(codec.x509.X509Crl crl)
    throws java.lang.Exception
    Signs the given certificate revocation list. All issuer dependent fields and the serial
    number are set to correct values before signing. All other values have to be set to
    correct values by the method's caller before calling this method.
Throws:
    java.lang.Exception - If signing failed.
```

updateCrl

```
public byte[] updateCrl(java.util.Vector crlEntries,
    java.util.Calendar nextUpdate)
    throws java.rmi.RemoteException
```

Adds the given CrlEntries to crl. The current crl is updated and a new lean crl is returned.

Parameters:
 crlEntries - The new CrlEntries.
 nextUpdate - When the crl will be updated next time (worst case).

Throws:
 java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

verify

```
public void verify(byte[] msg,
    byte[] sig,
    java.security.Principal issuer)
    throws CaException,
    java.lang.Exception
```

Verifies a binary message.

Parameters:
 msg - The signed message.
 sig - The signature to verify.
 issuer - The signer.

Throws:
 CaException - If given signature is not valid for some reason.
 java.lang.Exception - If verification failed. **ATTENTION:** This exception does NOT imply an invalid signature! It ONLY implies a failure in the verification process.

verify

```
public void verify(byte[] msg,
    byte[] sig,
    java.security.PublicKey pubkey,
    java.lang.String sigAlg,
    java.lang.String secProv)
    throws CaException,
    java.lang.Exception
```

Verifies a binary message.

Parameters:
 msg - The signed message.
 sig - The signature to verify.
 pubkey - The PublicKey to verify with.
 sigAlg - The algorithm to verify with.
 sigProv - The Provider of the signature engine to verify with. Giving null leads to using the default provider.

Throws:
 CaException - If given signature is not valid for some reason.
 java.lang.Exception - If verification failed. **ATTENTION:** This exception does NOT imply an invalid signature! It ONLY implies a failure in the verification process.

verify

```
public void verify(Codec.pkcs7.SignedData sd)
    throws CaException,
    java.lang.Exception
```

Verifies the data found in the given SignedData object. There can be many signatures (SignerInfos) included. For each one the associated issuer's X509Certificate is verified and than the signature itself.

Parameters:
 sd - The instance to be verified.

Throws:
 CaException - If given instance is not valid for some reason.
 java.lang.Exception - If verification failed. **ATTENTION:** This does NOT imply an invalid signature! It ONLY implies a failure in the verification process.

verify

```
public void verify(Codec.x509.X509Certificate cert)
    throws CaException,
    java.lang.Exception
```

Verifies the given certificate.

Parameters:
 cert - The certificate to verify.

Throws:
 CaException - If given instance is not valid for some reason.
 java.lang.Exception - If verification failed. **ATTENTION:** This does NOT imply an invalid signature! It ONLY implies a failure in the verification process.

verify

```
public void verify(Codec.x509.X509Crl crl)
    throws CaException,
    java.lang.Exception
```

Verifies the given certificate revocation list.

Parameters:
 crl - The crl to verify.

Throws:
 CaException - If given instance is not valid for some reason.
 java.lang.Exception - If verification failed. **ATTENTION:** This does NOT imply an invalid signature! It ONLY implies a failure in the verification process.

[Overview Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS	NEXT CLASS	FRAMES	NO FRAMES
SUMMARY	INNER FIELD CONSTR METHOD	DETAIL	FIELD CONSTR METHOD

[Overview Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS	NEXT CLASS	FRAMES	NO FRAMES
SUMMARY	INNER FIELD CONSTR METHOD	DETAIL	FIELD CONSTR METHOD

de.tud.cdc.flexiTrust.ca.app
Class CaCoreProcessorQueue

```
java.lang.Object
|--java.rmi.server.RemoteObject
    |--java.rmi.server.RemoteServer
        --java.rmi.server.UnicastRemoteObject
            --de.tud.cdc.flexiTrust.ca.app.CaCoreProcessorQueue
```

public class CaCoreProcessorQueue
 extends java.rmi.server.UnicastRemoteObject
 implements CaCoreProcessorQueueRmi

The [Queue](#) for [CaCoreProcessors](#).

Version:
 \$Id: CaCoreProcessorQueue.java,v 1.101 2000/12/28 18:15:14 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class java.rmi.server.RemoteObject

ref

Fields inherited from interface de.tud.cdc.flexiTrust.ca.api.Queue

[HIGH](#), [LOW](#), [MED](#)

Method Summary

protected void	finalize()	Destructs the instance.
boolean	isEmpty()	Returns wether the queue is empty or not.
static void	main (java.lang.String[] args)	Invoked by runtime system when this class is executed.

```
byte[] pop\(\)
    Dequeues the next Processable.
void push(int pri, byte[] pro)
    Enqueues a given Processable.
```

Methods inherited from class java.rmi.server.UnicastRemoteObject

clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer

getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject

equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Method Detail

finalize

```
protected void finalize()
    Destructs the instance.
Overrides:
    finalize in class java.lang.Object
```

isEmpty

```
public boolean isEmpty()
    Returns wether the queue is empty or not.
Specified by:
    isEmpty in interface CaCoreProcessorQueueRmi
Returns:
    true if the queue is empty, false otherwise.
```

main

```
public static void main(java.lang.String[] args)
    Invoked by runtime system when this class is executed. Creates a new instance of this class, joins it to DJINN and waits for user's command to end the service.
Parameters:
    args - The command line arguments.
```

pop

```
public byte[] pop()
    Dequeues the next Processable.
Specified by:
    pop in interface CaCoreProcessorQueueRmi
Returns:
    The instance to process next.
```

push

```
public void push(int pri,
                byte[] pro)
    Enqueues a given Processable. Possible priorities are described in Field Summary.
Specified by:
    push in interface CaCoreProcessorQueueRmi
Parameters:
    pri - The priority of given Processable
    pro - The instance to enqueue for processing.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.app Interface CaCoreProcessorQueueRmi

All Known Implementing Classes:
[CaCoreProcessorQueue](#)

```
public interface CaCoreProcessorQueueRmi
    extends Queue, java.rmi.Remote
```

The RMI-interface for [CaCoreProcessorQueue](#).

Version:
\$Id: CaCoreProcessorQueueRmi.java,v 1.30 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from interface [de.tud.cdc.flexiTrust.ca.api.Queue](#)

[HIGH](#), [LOW](#), [MED](#)

Method Summary

boolean	isEmpty()	Returns wether the queue is empty or not.
byte[]	pop()	Dequeues the next Processable .
void	push(int pri, byte[] pro)	Enqueues a given Processable .

Method Detail

isEmpty

```
public boolean isEmpty()
    throws java.rmi.RemoteException
    Returns wether the queue is empty or not.
Specified by:
    isEmpty in interface Queue
Returns:
```

true if the queue is empty, false otherwise.

Throws:
java.rmi.RemoteException - Any occuring Exception is rethrown as nested Exception of a RemoteException.

pop

```
public byte[] pop()
    throws java.rmi.RemoteException
    Dequeues the next Processable.
Specified by:
    pop in interface Queue
Returns:
    The instance to process next.
Throws:
    java.rmi.RemoteException - Any occuring Exception is rethrown as nested Exception of a RemoteException.
```

push

```
public void push(int pri,
                byte[] pro)
    throws java.rmi.RemoteException
    Enqueues a given Processable. Possible priorities are described in Field Summary.
Specified by:
    push in interface Queue
Parameters:
    pri - The priority of given Processable
    pro - The instance to enqueue for processing.
Throws:
    java.rmi.RemoteException - Any occuring Exception is rethrown as nested Exception of a RemoteException.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.app Class CaException

```
java.lang.Object
|--java.lang.Throwable
   |--java.lang.Exception
      |--java.io.IOException
         |--java.rmi.RemoteException
            |--de.tud.cdc.flexiTrust.ca.app.CaException
```

```
public class CaException
    extends java.rmi.RemoteException
```

Covers all Exceptions which might be thrown originally by FlexiTrustCA classes.

Version:
\$Id: CaException.java,v 1.42 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Field Summary

static int	BAD_SIGNATURE	Mnemonic for exception type: Given Siganture is bad.
static int	EXPIRE	Mnemonic for exception type: Procesable must be expired.
static int	NO_SIGNATURE	Mnemonic for exception type: No signature found in SignedData.
static int	REVOKING_UNISSUED_CERT	Mnemonic for exception type: Revoking unissued cert.
static int	SERIAL_ALREADY_USED	Mnemonic for exception type: Serial already used.
static int	UNKNOWN_SIGNATURE	Mnemonic for exception type: Unknown signature found in SignedData.

Fields inherited from class [java.rmi.RemoteException](#)

detail

Constructor Summary

[CaException](#)(int type)
Constructs new instance with given type.

[CaException](#)(java.lang.String msg, int type)
Creates and initializes a new Exception with given type.

[CaException](#)(java.lang.String msg, java.lang.Throwable t, int type)
Creates and initializes a new Exception with given type.

Method Summary

int [getType](#)()
Returns the type of this Exception.

Methods inherited from class java.rmi.RemoteException

getMessage, printStackTrace, printStackTrace, printStackTrace

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, toString

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

NO_SIGNATURE

public static final int **NO_SIGNATURE**
Mnemonic for exception type: No signature found in SignedData.

UNKNOWN_SIGNATURE

public static final int **UNKNOWN_SIGNATURE**
Mnemonic for exception type: Unknown signature found in SignedData.

BAD_SIGNATURE

public static final int **BAD_SIGNATURE**
Mnemonic for exception type: Given Signature is bad.

SERIAL_ALREADY_USED

public static final int **SERIAL_ALREADY_USED**
Mnemonic for exception type: Serial already used.

REVOKING_UNISSUED_CERT

public static final int **REVOKING_UNISSUED_CERT**
Mnemonic for exception type: Revoking unissued cert.

EXPIRE

public static final int **EXPIRE**
Mnemonic for exception type: Processable must be expired.

Constructor Detail

CaException

public **CaException**(int type)
int type)
Constructs new instance with given type. Possible types are defined in [Field Summary](#).

Parameters:
type - The Exception's type.

CaException

public **CaException**(java.lang.String msg,
int type)
Creates and initializes a new Exception with given type. Possible types are defined in [Field Summary](#).

Parameters:
msg - The Exception's message.

type - The Exception's type.

CaException

public **CaException**(java.lang.String msg,
java.lang.Throwable t,
int type)
Creates and initializes a new Exception with given type. Possible types are defined in [Field Summary](#).

Parameters:
msg - The Exception's message.
t - The Throwable that caused this Exception.
type - The Exception's type.

Method Detail

getType

public int **getType**()
Returns the type of this Exception. Possible types are defined in [Field Summary](#).

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexitrust.ca.app
Class CaStock

java.lang.Object
|--java.rmi.server.RemoteObject
 |--java.rmi.server.RemoteServer
 |--java.rmi.server.UnicastRemoteObject
 |--de.tud.cdc.flexitrust.ca.app.CaStock

public class **CaStock**
extends java.rmi.server.UnicastRemoteObject
implements [CaStockRmi](#)

Manages centralized resources as keys, certs and so on.

Version:
\$Id: CaStock.java,v 1.111 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Field Summary

static codec.x501.Name	AD
static codec.x501.Name	CA

Fields inherited from class java.rmi.server.RemoteObject

ref

Method Summary

void	addCert (java.math.BigInteger serial, byte[] cert)	Adds the given certificate to list of all certificates.
protected void	finalize ()	Destructs the instance.

A JAVADOCS

byte[]	getCert (java.lang.String subject) Returns the certificate belonging to given subject.
java.math.BigInteger	getNextSerial (java.lang.String issuer) Returns the serial next to be used for given issuer.
java.security.PrivateKey	getPrivKey (java.lang.String issuer) Returns the private key of given issuer.
java.lang.String	getProperty (java.lang.String alias) Returns the property with given alias.
java.security.PublicKey	getPubKey (java.lang.String principal) Returns the public key for given principal.
java.security.Provider	getSecProv (java.lang.String issuer) Returns the security provider used by given issuer.
java.security.Provider[]	getSecProvs () Returns all security providers used by known issuers.
java.lang.String	getSigAlg (java.lang.String issuer) Returns the signature algorithm of given issuer.
static void	main (java.lang.String[] args) Invoked by runtime system when this class is executed.
byte[]	updateCrl (java.util.Vector crlEntries, java.util.Calendar nextUpdate) Updates the crl with given crl entries and returns the new lean crl (DER encoded).

Methods inherited from class java.rmi.server.UnicastRemoteObject
clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer
getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject
equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Field Detail

CA

```
public static final codec.x501.Name CA
```

AD

```
public static final codec.x501.Name AD
```

Method Detail

addCert

```
public void addCert(java.math.BigInteger serial,
                   byte[] cert)
    throws java.rmi.RemoteException
```

Adds the given certificate to list of all certificates.
Specified by:
[addCert](#) in interface [CaStockRmi](#)
Parameters:
serial - The serial number of the new cert.
cert - The new cert (DER encoded).
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

finalize

```
protected void finalize()
    Destructs the instance.
```

Overrides:
finalize in class java.lang.Object

getCert

```
public byte[] getCert(java.lang.String subject)
    throws java.rmi.RemoteException
```

Returns the certificate belonging to given subject. The certificate is returned DER encoded.
Specified by:
[getCert](#) in interface [CaStockRmi](#)
Parameters:
subject - Return the cert for whom?
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getNextSerial

```
public java.math.BigInteger getNextSerial(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the serial next to be used for given issuer.
Specified by:
[getNextSerial](#) in interface [CaStockRmi](#)
Parameters:
issuer - The issuer to get the next serial for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPrivKey

```
public java.security.PrivateKey getPrivKey(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the private key of given issuer.
Specified by:
[getPrivKey](#) in interface [CaStockRmi](#)
Parameters:
issuer - The issuer to get the private key for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getProperty

```
public java.lang.String getProperty(java.lang.String alias)
    throws java.rmi.RemoteException
```

Returns the property with given alias.
Specified by:
[getProperty](#) in interface [CaStockRmi](#)
Parameters:
alias - The alias to get the property for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPubKey

```
public java.security.PublicKey getPubKey(java.lang.String principal)
    throws java.rmi.RemoteException
```

Returns the public key for given principal.

Specified by:
[getPubKey](#) in interface [CaStockRmi](#)
Parameters:
principal - The principal to fetch the public key for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProv

```
public java.security.Provider getSecProv(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the security provider used by given issuer.
Specified by:
[getSecProv](#) in interface [CaStockRmi](#)
Parameters:
issuer - The issuer to fetch the provider for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProvs

```
public java.security.Provider[] getSecProvs()
    throws java.rmi.RemoteException
```

Returns all security providers used by known issuers.
Specified by:
[getSecProvs](#) in interface [CaStockRmi](#)
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSigAlg

```
public java.lang.String getSigAlg(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the signature algorithm of given issuer.
Specified by:
[getSigAlg](#) in interface [CaStockRmi](#)
Parameters:
issuer - The issuer to fetch the signature algorithm for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

main

```
public static void main(java.lang.String[] args)
```

Invoked by runtime system when this class is executed. Creates a new instance of this class, joins it to DJINN and waits for user's command to end the service.

Parameters:
args - The command line arguments.

updateCrl

```
public byte[] updateCrl(java.util.Vector crlEntries,
                       java.util.Calendar nextUpdate)
    throws java.rmi.RemoteException
```

Updates the crl with given crl entries and returns the new lean crl (DER encoded).

Specified by:
[updateCrl](#) in interface [CaStockRmi](#)

Parameters:
crlEntries - The new revocations.
nextUpdate - The time when next update will be made (worst case).

Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.app
Interface CaStockRmi

All Known Implementing Classes:
[CaStock](#)

public interface **CaStockRmi**
extends java.rmi.Remote, [Stock](#)

The RMI-Interface for [CaStock](#).

Version:
\$Id: CaStockRmi.java,v 1.30 2000/12/27 18:10:53 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Method Summary

void	addCert (java.math.BigInteger serial, byte[] cert)	Adds the given certificate to list of all certificates.
byte[]	getCert (java.lang.String subject)	Returns the certificate belonging to given subject.
java.math.BigInteger	getNextSerial (java.lang.String issuer)	Returns the serial next to be used for given issuer.
java.security.PrivateKey	getPrivKey (java.lang.String issuer)	Returns the private key of given issuer.
java.lang.String	getProperty (java.lang.String alias)	Returns the property with given alias.
java.security.PublicKey	getPubKey (java.lang.String principal)	Returns the public key for given principal.
java.security.Provider	getSecProv (java.lang.String issuer)	Returns the security provider used by given issuer.
java.security.Provider[]	getSecProvs ()	Returns all security providers used by known issuers.
java.lang.String	getSigAlg (java.lang.String principal)	Returns the signature algorithm of given issuer.
byte[]	updateCrl (java.util.Vector crlEntries, java.util.Calendar nextUpdate)	Updates the crl with given crl entries and returns the new lean crl (DER encoded).

Method Detail

addCert

```
public void addCert(java.math.BigInteger serial,
                   byte[] cert)
    throws java.rmi.RemoteException
```

Adds the given certificate to list of all certificates.

Parameters:
serial - The serial number of the new cert.
cert - The new cert (DER encoded).

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getCert

```
public byte[] getCert(java.lang.String subject)
    throws java.rmi.RemoteException
```

Returns the certificate belonging to given subject. The certificate is returned DER encoded.

Parameters:
subject - Return the cert for whom?

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getNextSerial

```
public java.math.BigInteger getNextSerial(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the serial next to be used for given issuer.

Parameters:
issuer - The issuer to get the next serial for.

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPrivKey

```
public java.security.PrivateKey getPrivKey(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the private key of given issuer.

Parameters:
issuer - The issuer to get the private key for.

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getProperty

```
public java.lang.String getProperty(java.lang.String alias)
    throws java.rmi.RemoteException
```

Returns the property with given alias.

Parameters:
alias - The alias to get the property for.

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPubKey

```
public java.security.PublicKey getPubKey(java.lang.String principal)
    throws java.rmi.RemoteException
```

Returns the public key for given principal.

Parameters:
principal - The principal to fetch the public key for.

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProv

```
public java.security.Provider getSecProv(java.lang.String issuer)
    throws java.rmi.RemoteException
```

Returns the security provider used by given issuer.

Parameters:
issuer - The issuer to fetch the provider for.

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProvs

```
public java.security.Provider[] getSecProvs()
    throws java.rmi.RemoteException
```

Returns all security providers used by known issuers.

Throws:

RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSigAlg

```
public java.lang.String getSigAlg(java.lang.String principal)
    throws java.xml.RemoteException
```

Returns the signature algorithm of given issuer.

Parameters:
 issuer - The issuer to fetch the signature algorithm for.

Throws:
 RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

updateCrl

```
public byte[] updateCrl(java.util.Vector crlEntries,
    java.util.Calendar nextUpdate)
    throws java.xml.RemoteException
```

Updates the crl with given crl entries and returns the new lean crl (DER encoded).

Parameters:
 crlEntries - The new revocations.
 nextUpdate - The time when next update will be made (worst case).

Throws:
 RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL](#) [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) [DETAIL](#) [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.app Class Pkcs7Exporter

```
java.lang.Object
|
|-- java.lang.Thread
|   |
|   |-- de.tud.cdc.flexiTrust.ca.app.Pkcs7Exporter
```

public class **Pkcs7Exporter**
 extends java.lang.Thread
 implements [Exporter](#)

Exports [Processables](#). Each [Processable](#) is serialized into an own codec.pkcs7.SignedData object which is signed by this [Exporter](#). The signedData itself is DER encoded and stored together with others into a senior signedData. The senior signedData is signed by this [Exporter](#), too. Then the senior signedData is DER encoded and stored into a file which is the very export. The resulting file has a format that could be read in by [Pkcs7Importer](#).

Version:
 Std: Pkcs7Exporter.java,v 1.100 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class java.lang.Thread

MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Method Summary

void	export (Processable pro, java.lang.Object[] params)	Exports the given Processable .
protected void	finalize ()	Destructs the instance.
static void	main (java.lang.String[] args)	Invoked by runtime system when this class is executed.
boolean	quest (java.lang.String text, boolean defaultResult)	Displays the given question and returns the answer.
java.lang.String	readLine (java.lang.String text, java.lang.String defaultResult)	Displays the given message and reads a line of text.
void	run ()	

	Looks up the Pkcs7ExporterQueue and pops Processable 's from it.
void	setState (java.lang.String text) Displays the given text as the current state
void	setState (java.lang.String text, int progress) Displays the given text as the current state and the given progress as current progress.
void	show (java.lang.String text) Displays the given text.

Methods inherited from class java.lang.Thread

activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

export

```
public void export(Processable pro,
    java.lang.Object[] params)
```

Exports the given [Processable](#). This is done by packing the given processable into a new codec.pkcs7.SignedData object, signing it and adding it to senior SignedData.

Specified by:
[export](#) in interface [Exporter](#)

Parameters:
 pro - The instance to export.
 params - Ignored.

finalize

```
protected void finalize()
```

Destructs the instance.

Overrides:
 finalize in class java.lang.Object

main

```
public static void main(java.lang.String[] args)
```

Invoked by runtime system when this class is executed. Creates a new instance of this class, starts it and waits for user's command to end the service.

Parameters:
 args - The command line arguments.

quest

```
public boolean quest(java.lang.String text,
    boolean defaultResult)
```

Displays the given question and returns the answer.

Parameters:
 text - The question to be displayed.
 defaultResult - The default result (used in auto mode).

Returns:
 The answer.

readLine

```
public java.lang.String readLine(java.lang.String text,
    java.lang.String defaultResult)
```

Displays the given message and reads a line of text.

Parameters:
 text - The message to display.
 defaultResult - The default result (used in auto mode).

Returns:
 The line of text read in.

run

```
public void run()
```

Looks up the [Pkcs7ExporterQueue](#) and pops [Processable](#)'s from it. Invoked by runtime system when this Thread is started.

Overrides:
 run in class java.lang.Thread

setState

```
public void setState(java.lang.String text)
```

Displays the given text as the current state
Parameters:
 text - The current state.

setState

```
public void setState(java.lang.String text,
                    int progress)
    Displays the given text as the current state and the given progress as current progress.
Parameters:
    text - The current state.
    progress - The current progress.
```

show

```
public void show(java.lang.String text)
    Displays the given text.
Parameters:
    The - text to display.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.app
Class Pkcs7ExporterQueue

```
java.lang.Object
|--java.rmi.server.RemoteObject
   |--java.rmi.server.RemoteServer
      |--java.rmi.server.UnicastRemoteObject
         |--de.tud.cdc.flexiTrust.ca.app.Pkcs7ExporterQueue
```

public class **Pkcs7ExporterQueue**
 extends java.rmi.server.UnicastRemoteObject
 implements [Pkcs7ExporterQueueRmi](#)

The [Queue](#) for [Pkcs7Exporters](#).

Version:
 SId: Pkcs7ExporterQueue.java,v 1.60 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class java.rmi.server.RemoteObject

ref

Fields inherited from interface de.tud.cdc.flexiTrust.ca.api.Queue

HIGH , LOW , MED
--

Method Summary

protected void	finalize()	Destructs the instance.
boolean	isEmpty()	Returns whether the queue is empty or not.
static void	main (java.lang.String[] args)	Invoked by runtime system when this class is executed.

byte[]	pop()	Dequeues the next Processable .
void	push (int pri, byte[] pro)	Enqueues a given Processable .

Methods inherited from class java.rmi.server.UnicastRemoteObject

clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer

getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject

equals, getRef, hashCode, toString, toStub
--

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Method Detail

finalize

```
protected void finalize()
    Destructs the instance.
Overrides:
    finalize in class java.lang.Object
```

isEmpty

```
public boolean isEmpty()
    Returns whether the queue is empty or not.
Specified by:
    isEmpty in interface Pkcs7ExporterQueueRmi
Returns:
    true if the queue is empty, false otherwise.
```

main

```
public static void main(java.lang.String[] args)
    Invoked by runtime system when this class is executed. Creates a new instance of this class, joins it to DJINN and waits for user's command to end the service.
Parameters:
    args - The command line arguments.
```

pop

```
public byte[] pop()
    Dequeues the next Processable.
Specified by:
    pop in interface Pkcs7ExporterQueueRmi
Returns:
    The instance to process next.
```

push

```
public void push(int pri,
                byte[] pro)
    Enqueues a given Processable. Possible priorities are described in Field Summary.
Specified by:
    push in interface Pkcs7ExporterQueueRmi
Parameters:
    pri - The priority of given Processable
    pro - The instance to enqueue for processing.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [FRAMES](#) [NO FRAMES](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.app Interface Pkcs7ExporterQueueRmi

All Known Implementing Classes:
[Pkcs7ExporterQueue](#)

public interface **Pkcs7ExporterQueueRmi**
extends [Queue](#), [java.rmi.Remote](#)

The RMI-Interface for [Pkcs7ExporterQueue](#).

Version:
SId: Pkcs7ExporterQueueRmi.java, v 1.27 2000/12/28 18:15:15 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from interface [de.tud.cdc.flexiTrust.ca.api.Queue](#)

[HIGH](#), [LOW](#), [MED](#)

Method Summary

boolean	isEmpty()	Returns whether the queue is empty or not.
byte[]	pop()	Dequeues the next Processable .
void	push(int pri, byte[] pro)	Enqueues a given Processable .

Method Detail

isEmpty

public boolean **isEmpty()**
throws [java.rmi.RemoteException](#)

Returns whether the queue is empty or not.

Specified by:
[isEmpty](#) in interface [Queue](#)

Returns:

true if the queue is empty, false otherwise.

Throws:
[java.rmi.RemoteException](#) - Any occurring [Exception](#) is rethrown as nested [Exception](#) of a [RemoteException](#).

pop

public byte[] **pop()**
throws [java.rmi.RemoteException](#)

Dequeues the next [Processable](#).

Specified by:
[pop](#) in interface [Queue](#)

Returns:
The instance to process next.

Throws:
[java.rmi.RemoteException](#) - Any occurring [Exception](#) is rethrown as nested [Exception](#) of a [RemoteException](#).

push

public void **push(int pri, byte[] pro)**
throws [java.rmi.RemoteException](#)

Enqueues a given [Processable](#). Possible priorities are described in [Field Summary](#).

Specified by:
[push](#) in interface [Queue](#)

Parameters:
pri - The priority of given [Processable](#)
pro - The instance to enqueue for processing.

Throws:
[java.rmi.RemoteException](#) - Any occurring [Exception](#) is rethrown as nested [Exception](#) of a [RemoteException](#).

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [FRAMES](#) [NO FRAMES](#) [DETAIL: FIELD | CONSTR | METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [FRAMES](#) [NO FRAMES](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.app Class Pkcs7Importer

java.lang.Object
|
+- java.lang.Thread
|
+- de.tud.cdc.flexiTrust.ca.app.Pkcs7Importer

public class **Pkcs7Importer**
extends [java.lang.Thread](#)
implements [Importer](#)

Loads a number of [Processables](#) stored serialized in a [codec.pkcs7.SignedData](#) object which itself is stored DER encoded in a file. This [codec.pkcs7.SignedData](#) object is signed by a trusted registration authority and contains a number of further DER encoded [codec.pkcs7.SignedData](#) objects. Each of those is signed as well and contains one serialized [Processable](#) object. This file format is the same as this produced by [Pkcs7ExportApp](#).

When the file is read in, the outer [codec.pkcs7.SignedData](#) object is decoded and its signature is verified. After that each of the [Processable](#) objects is deserialized and its process is started. The first step in each [Processable](#)'s process should be the verification of the own signature (which is given to it as a parameter).

Version:
SId: Pkcs7Importer.java, v 1.114 2001/01/11 16:44:34 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class [java.lang.Thread](#)

[MAX_PRIORITY](#), [MIN_PRIORITY](#), [NORM_PRIORITY](#)

Method Summary

protected void	finalize()	Destructs the instance.
static void	main(java.lang.String[] args)	Invoked by runtime system when this class is executed.
boolean	quiet(java.lang.String text, boolean defaultResult)	Displays the given question and returns the answer.
java.lang.String	readLine(java.lang.String text, java.lang.String defaultResult)	

	run()	Displays the given message and reads a line of text.
void	run()	Loads Processables and imports them into the Advanced Processing Infrastructure.
void	setState(java.lang.String text)	Displays the given text as the current state
void	setState(java.lang.String text, int progress)	Displays the given text as the current state and the given progress as current progress.
void	show(java.lang.String text)	Displays the given text.

Methods inherited from class [java.lang.Thread](#)

[activeCount](#), [checkAccess](#), [countStackFrames](#), [currentThread](#), [destroy](#), [dumpStack](#), [enumerate](#), [getContextClassLoader](#), [getName](#), [getPriority](#), [getThreadGroup](#), [interrupt](#), [interrupted](#), [isAlive](#), [isDaemon](#), [isInterrupted](#), [join](#), [join](#), [resume](#), [setContextClassLoader](#), [setDaemon](#), [setName](#), [setPriority](#), [sleep](#), [sleep](#), [start](#), [stop](#), [stop](#), [suspend](#), [toString](#), [yield](#)

Methods inherited from class [java.lang.Object](#)

[clone](#), [equals](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [wait](#), [wait](#), [wait](#)

Method Detail

finalize

protected void **finalize()**
Destructs the instance.

Overrides:
[finalize](#) in class [java.lang.Object](#)

main

public static void **main(java.lang.String[] args)**
Invoked by runtime system when this class is executed. Creates a new instance of this class, starts it.

Parameters:
args - The command line arguments.

quest

```
public boolean quest(java.lang.String text,
                    boolean defaultResult)
    Displays the given question and returns the answer.
Parameters:
    text - The question to be displayed.
    defaultResult - The default result (used in auto mode).
Returns:
    The answer.
```

readLine

```
public java.lang.String readLine(java.lang.String text,
                                 java.lang.String defaultResult)
    Displays the given message and reads a line of text.
Parameters:
    text - The message to display.
    defaultResult - The default result (used in auto mode).
Returns:
    The line of text read in.
```

run

```
public void run()
    Loads processables and imports them into the Advanced Processing Infrastructure.
    If in auto mode this is done exactly one time. If not in auto mode this is done until
    the user terminates the program. Invoked by runtime system when this Thread is
    started.
Overrides:
    run in class java.lang.Thread
```

setState

```
public void setState(java.lang.String text)
    Displays the given text as the current state
Parameters:
    text - The current state.
```

setState

```
public void setState(java.lang.String text,
                    int progress)
    Displays the given text as the current state and the given progress as current progress.
```

Parameters:

text - The current state.
 progress - The current progress.

show

```
public void show(java.lang.String text)
    Displays the given text.
Parameters:
    The - text to display.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

A.3 DE.TUD.CDC.FLEXITRUST.CA.INIT

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
SUMMARY INNER FIELD CONSTR METHOD
FRAMES NO FRAMES
DETAIL FIELD CONSTR METHOD

de.tud.cdc.flexiTrust.ca.init
Class Properties

```

java.lang.Object
|
|--java.util.Dictionary
|   |--java.util.Hashtable
|       |--java.util.Properties
|           |--de.tud.cdc.flexiTrust.ca.init.Properties

```

public class **Properties**
 extends java.util.Properties

Behaves like java.util.Properties except that an unknown alias will result in program termination.

Version:
 \$Id: Properties.java,v 1.3 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class java.util.Properties defaults

Constructor Summary Properties()
--

Method Summary java.lang.String getProperty (java.lang.String alias) Returns the property corresponding to the given alias.
--

Methods inherited from class java.util.Properties
--

getProperty, list, list, load, propertyNamees, save, setProperty, store

Methods inherited from class java.util.Hashtable clear, clone, contains, containsKey, containsValue, elements, entrySet, equals, get, hashCode, isEmpty, keys, keySet, put, putAll, rehash, remove, size, toString, values
--

Methods inherited from class java.lang.Object finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Properties

```

public Properties()

```

Method Detail

getProperty

```

public java.lang.String getProperty(java.lang.String alias)

```

Returns the property corresponding to the given alias.

Overrides:
 getProperty in class java.util.Properties

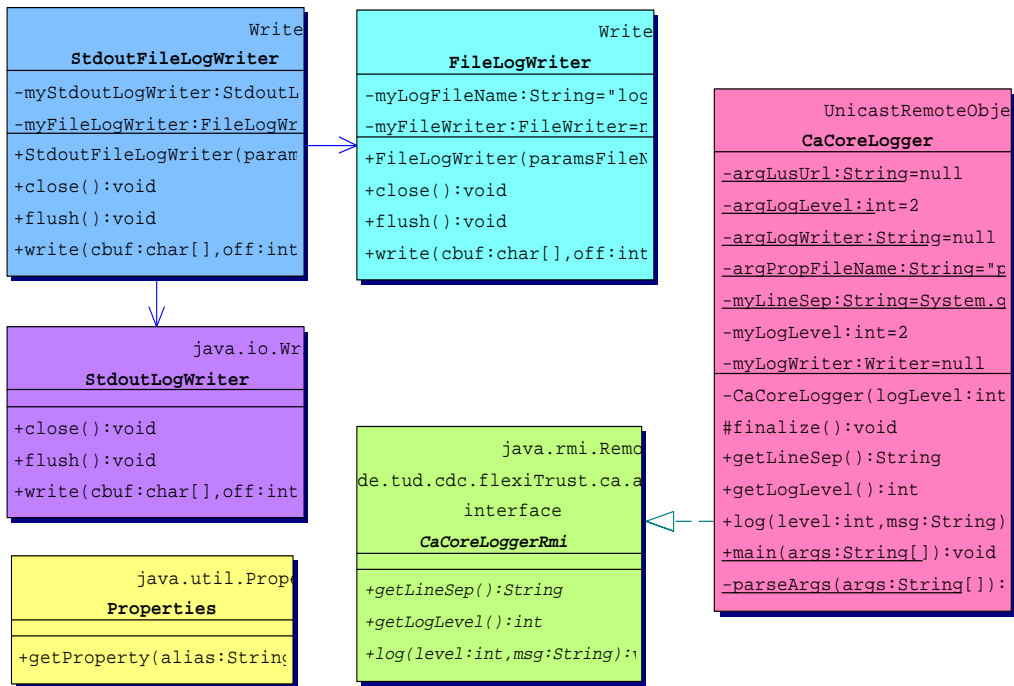
Parameters:
 alias - The alias to give the property for.

Returns:
 the property for given alias.

Throws:
 java.lang.Error - If given alias corresponds to no property.

Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
SUMMARY INNER FIELD CONSTR METHOD
FRAMES NO FRAMES
DETAIL FIELD CONSTR METHOD

A.4 de.tud.cdc.flexiTrust.ca.log



[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL: FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.log

Class CaCoreLogger

```

java.lang.Object
|
|-- java.rmi.server.RemoteObject
|   |-- java.rmi.server.RemoteServer
|       |-- java.rmi.server.UnicastRemoteObject
|           |-- de.tud.cdc.flexiTrust.ca.log.CaCoreLogger
    
```

public class **CaCoreLogger**
extends java.rmi.server.UnicastRemoteObject
implements [CaCoreLoggerRmi](#)

Provides a mechanism for centralized logging.

Version:
\$Id: CaCoreLogger.java,v 1.5 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class java.rmi.server.RemoteObject
ref

Fields inherited from interface de.tud.cdc.flexiTrust.ca.api.Logger
[NORMAL](#), [SILENT](#), [VERBOSE](#)

Method Summary

protected void	finalize()	Destructs the instance.
java.lang.String	getLineSep()	Returns the platform dependent line separator of the CaCoreLoggers host computer.
int	getLogLevel()	Returns the current log level.

void	log (int level, java.lang.String msg)	Adds the given message to log.
static void	main (java.lang.String[] args)	Invoked by runtime system when this class is executed.

Methods inherited from class java.rmi.server.UnicastRemoteObject
clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer
getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject
equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object
getClass, notify, notifyAll, wait, wait, wait

Method Detail

finalize

protected void **finalize()**
Destructs the instance.
Overrides:
finalize in class java.lang.Object

getLineSep

public java.lang.String **getLineSep()**
Returns the platform dependent line separator of the CaCoreLoggers host computer.
Specified by:
[getLineSep](#) in interface [CaCoreLoggerRmi](#)
Returns:
The platform dependent line separator.

A.4 DE.TUD.CDC.FLEXITRUST.CA.LOG

getLogLevel

```
public int getLogLevel()
    Returns the current log level. Possible log levels are described in Logger's Field Summary.
Specified by:
getLogLevel in interface CaCoreLoggerRmi
Returns:
    The current log level.
```

log

```
public void log(int level,
               java.lang.String msg)
    throws java.rmi.RemoteException
    Adds the given message to log.
Specified by:
log in interface CaCoreLoggerRmi
Parameters:
    level - The level of the given message. If the message's level is greater than this
    Logger's level it will not be logged. Possible levels are described in Loggers's Field Summary.
    msg - The message to log.
Throws:
    java.rmi.RemoteException - Any occurring exception is rethrown as nested
    Exception of a RemoteException.
```

main

```
public static void main(java.lang.String[] args)
    throws java.rmi.RemoteException
    Invoked by runtime system when this class is executed. Creates a new instance of this
    class, joins it to DJINN and waits for user's command to end the service.
Parameters:
    args - The command line arguments.
Throws:
    java.rmi.RemoteException - Any occurring exception is rethrown as nested
    Exception of a RemoteException.
```

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.log
Interface CaCoreLoggerRmi

All Known Implementing Classes:
[CaCoreLogger](#)

public interface **CaCoreLoggerRmi**
extends java.rmi.Remote, [Logger](#)

The RMI-Interface for [CaCoreLogger](#).

Version:
\$Id: CaCoreLoggerRmi.java,v 1.4 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from interface de.tud.cdc.flexiTrust.ca.api.Logger

[NORMAL](#), [SILENT](#), [VERBOSE](#)

Method Summary

java.lang.String	getLineSep()	Returns the platform dependent line separator of the CaCoreLoggers host computer.
int	getLogLevel()	Returns the current log level.
void	log (int level, java.lang.String msg)	Adds the given message to log.

Method Detail

getLineSep

```
public java.lang.String getLineSep()
    throws java.rmi.RemoteException
    Returns the platform dependent line separator of the CaCoreLoggers host computer.
Returns:
    The platform dependent line separator.
```

Throws:
RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getLogLevel

```
public int getLogLevel()
    throws java.rmi.RemoteException
    Returns the current log level. Possible log levels are described in Logger's Field Summary.
Specified by:
getLogLevel in interface Logger
Returns:
    The current log level.
Throws:
    RemoteException - Any occurring Exception is rethrown as nested Exception of a
    RemoteException.
```

log

```
public void log(int level,
               java.lang.String msg)
    throws java.rmi.RemoteException
    Adds the given message to log.
Specified by:
log in interface Logger
Parameters:
    level - The level of the given message. If the message's level is greater than
    Logger's level it will not be logged. Possible levels are described in Loggers's Field Summary.
    msg - The message to log.
Throws:
    RemoteException - Any occurring Exception is rethrown as nested Exception of a
    RemoteException.
```

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.log
Class FileLogWriter

```
java.lang.Object
|
|--java.io.Writer
|
|--de.tud.cdc.flexiTrust.ca.log.FileLogWriter
```

public class **FileLogWriter**
extends java.io.Writer

Writes log entries to a file.

Version:
\$Id: FileLogWriter.java,v 1.2 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class java.io.Writer

[lock](#)

Constructor Summary

FileLogWriter (java.lang.String paramsFileName)	Constructs a new instance.
---	----------------------------

Method Summary

void	close()	Closes this writer
void	flush()	Flushes this writer
void	write (char[] cbuf, int off, int len)	Writes some chars.

Methods inherited from class java.io.Writer

[write](#), [write](#), [write](#), [write](#)

Methods inherited from class java.lang.Object
 clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

FileLogWriter

public **FileLogWriter**(java.lang.String paramsFileName)
 Constructs a new instance.
Parameters:
 paramsFileName - The name of the parameters file.

Method Detail

close

public void **close**()
 throws java.io.IOException
 Closes this writer.
Overrides:
 close in class java.io.Writer
Throws:
 java.io.IOException - Inherited from [Writer](#).

flush

public void **flush**()
 throws java.io.IOException
 Flushes this writer.
Overrides:
 flush in class java.io.Writer
Throws:
 java.io.IOException - Inherited from [Writer](#).

write

public void **write**(char[] cbuf,
 int off,
 int len)
 throws java.io.IOException
 Writes some chars.

Overrides:
 write in class java.io.Writer
Parameters:
 cbuf - The buffer holding the chars.
 off - The offset. Where to begin reading chars.
 len - The length. How many chars are to be written.
Throws:
 java.io.IOException - Inherited from [Writer](#).

Overview Package Class Tree Deprecated Index Help
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Overview Package Class Tree Deprecated Index Help
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.log
Class Properties

```

java.lang.Object
|
+-java.util.Dictionary
|   +-java.util.Hashtable
|       +-java.util.Properties
|           +-de.tud.cdc.flexiTrust.ca.log.Properties
    
```

public class **Properties**
 extends java.util.Properties

Behaves like java.util.Properties except that an unknown alias will result in program termination.

Version:
 SId: Properties.java,v 1.2 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class java.util.Properties
 defaults

Constructor Summary

[Properties](#)()

Method Summary

java.lang.String [getProperty](#)(java.lang.String alias)
 Returns the property corresponding the given alias.

Methods inherited from class java.util.Properties

[getProperty](#), [list](#), [list](#), [load](#), [propertyName](#), [save](#), [setProperty](#), [store](#)

Methods inherited from class java.util.Hashtable
 clear, clone, contains, containsKey, containsValue, elements, entrySet, equals, get, hashCode, isEmpty, keys, keySet, put, putAll, rehash, remove, size, toString, values

Methods inherited from class java.lang.Object
 finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

Properties

public **Properties**()

Method Detail

getProperty

public java.lang.String **getProperty**(java.lang.String alias)
 Returns the property corresponding the given alias.
Overrides:
 getProperty in class java.util.Properties
Parameters:
 alias - The alias to give the property for.
Returns:
 the property for given alias.
Throws:
 java.lang.Error - If given alias corresponds to no property.

Overview Package Class Tree Deprecated Index Help
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
 SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

A.4 DE.TUD.CDC.FLEXITRUST.CA.LOG

Overview	Package	Class	Tree	Deprecated	Index	Help
PREV CLASS	NEXT CLASS			FRAMES	NO FRAMES	
SUMMARY		INNER	FIELD	CONSTR	METHOD	
				DETAIL	FIELD	CONSTR METHOD

de.tud.cdc.flexiTrust.ca.log
Class StdoutFileLogWriter

```

java.lang.Object
|
|--java.io.Writer
|   |
|   |--de.tud.cdc.flexiTrust.ca.log.StdoutFileLogWriter

```

public class **StdoutFileLogWriter**
 extends java.io.Writer

Writes log entries to a file and to standard out.

Version:
 SId: StdoutFileLogWriter.java,v 1.3 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class java.io.Writer
lock

Constructor Summary
StdoutFileLogWriter (java.lang.String paramsFileName) Constructs a new instance.

Method Summary
void close () Closes this writer
void flush () Flushes this writer
void write (char[] cbuf, int off, int len) Writes some chars.

Methods inherited from class java.io.Writer
write, write, write, write

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail
StdoutFileLogWriter
public StdoutFileLogWriter (java.lang.String paramsFileName) Constructs a new instance. Parameters: paramsFileName - The name of the parameters file.

Method Detail
close
public void close () throws java.io.IOException Closes this writer. Overrides: close in class java.io.Writer Throws: java.io.IOException - Inherited from Writer.
flush
public void flush () throws java.io.IOException Flushes this writer. Overrides: flush in class java.io.Writer Throws: java.io.IOException - Inherited from Writer.
write
public void write (char[] cbuf, int off, int len) throws java.io.IOException Writes some chars.

Overrides:
 write in class java.io.Writer

Parameters:
 cbuf - The buffer holding the chars.
 off - The offset. Where to begin reading chars.
 len - The length. How many chars are to be written.

Throws:
 java.io.IOException - Inherited from Writer.

Overview	Package	Class	Tree	Deprecated	Index	Help
PREV CLASS	NEXT CLASS			FRAMES	NO FRAMES	
SUMMARY		INNER	FIELD	CONSTR	METHOD	
				DETAIL	FIELD	CONSTR METHOD

Overview	Package	Class	Tree	Deprecated	Index	Help
PREV CLASS	NEXT CLASS			FRAMES	NO FRAMES	
SUMMARY		INNER	FIELD	CONSTR	METHOD	
				DETAIL	FIELD	CONSTR METHOD

de.tud.cdc.flexiTrust.ca.log
Class StdoutLogWriter

```

java.lang.Object
|
|--java.io.Writer
|   |
|   |--de.tud.cdc.flexiTrust.ca.log.StdoutLogWriter

```

public class **StdoutLogWriter**
 extends java.io.Writer

Writes log entries to standard out.

Version:
 SId: StdoutLogWriter.java,v 1.2 2000/12/28 18:15:16 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Fields inherited from class java.io.Writer
lock

Constructor Summary
StdoutLogWriter ()

Method Summary
void close () Closes this writer.
void flush () Flushes this writer
void write (char[] cbuf, int off, int len) Writes some chars.

Methods inherited from class java.io.Writer
write, write, write, write

A JAVADOCS

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

StdoutLogWriter

public **StdoutLogWriter**()

Method Detail

close

public void **close**()
throws java.io.IOException
Closes this writer. Does nothing.
Overrides:
close in class java.io.Writer
Throws:
java.io.IOException - Inherited from `Writer`.

flush

public void **flush**()
throws java.io.IOException
Flushes this writer
Overrides:
flush in class java.io.Writer
Throws:
java.io.IOException - Inherited from `Writer`.

write

public void **write**(char[] cbuf,
int off,
int len)
throws java.io.IOException
Writes some chars.
Overrides:
write in class java.io.Writer
Parameters:

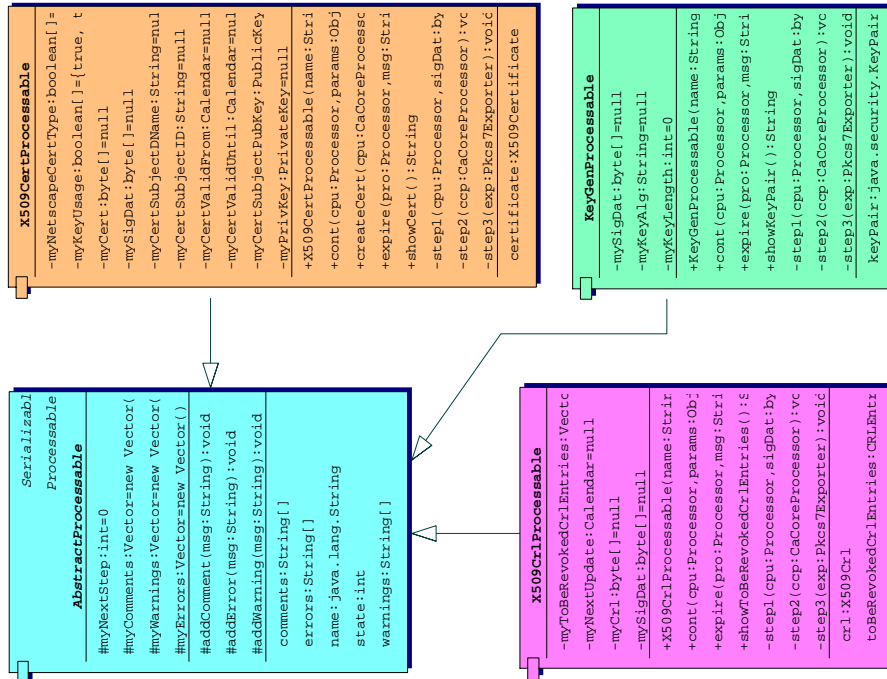
cbuf - The buffer holding the chars.
off - The offset. Where to begin reading chars.
len - The length. How many chars are to be written.
Throws:
java.io.IOException - Inherited from `Writer`.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

A.5 de.tud.cdc.flexiTrust.ca.pro



Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER FIELD CONSTR METHOD DETAIL: FIELD CONSTR METHOD

de.tud.cdc.flexiTrust.ca.pro Class AbstractProcessable

java.lang.Object
|-- de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable

Direct Known Subclasses:
[KeyGenProcessable](#), [X509CertificateProcessable](#), [X509CrlProcessable](#)

public abstract class **AbstractProcessable**
extends java.lang.Object
implements java.io.Serializable, [Processable](#)

Provides a [Processable](#)'s core functionality.

Version:
\$Id: AbstractProcessable.java,v 1.14 2000/12/29 11:48:33 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Field Summary	
protected java.util.Vector	myComments Holds all comments.
protected java.util.Vector	myErrors Holds all errors.
protected java.lang.String	myName Holds the {de.tud.cdc.flexiTrust.ca.api.Processable Processable's} name
protected int	myNextStep Holds the next step to be processed.
protected int	myState Holds the current state of the Processable .
protected java.util.Vector	myWarnings Holds all warnings.

Fields inherited from interface de.tud.cdc.flexiTrust.ca.api.Processable
[STATE_BEFORE_PROCESS](#), [STATE_FAILURE](#), [STATE_FROZEN](#), [STATE_STARTED](#), [STATE_SUCCESS](#)

Constructor Summary

[AbstractProcessable](#)()

Method Summary

protected void	addComment (java.lang.String msg) Adds the given message to my comments.
protected void	addError (java.lang.String msg) Adds the given message to my errors.
protected void	addWarning (java.lang.String msg) Adds the given message to my warnings.
java.lang.String[]	getComments () Returns all comments.
java.lang.String[]	getErrors () Returns all errors.
java.lang.String	getName () Returns the name of the current object.
int	getState () Returns the current state of process.
java.lang.String[]	getWarnings () Returns all warnings.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

myState

protected int **myState**
Holds the current state of the [Processable](#).

myNextStep

protected int **myNextStep**
Holds the next step to be processed.

A JAVADOCS

myName

protected java.lang.String **myName**
 Holds the (de.tud.cdc.flexiTrust.ca.api.Processable Processable's) name

myComments

protected java.util.Vector **myComments**
 Holds all comments. A comment is an assessment free message.

myWarnings

protected java.util.Vector **myWarnings**
 Holds all warnings. A warning is a message about some occurrence which usually should not happen, but does not cause the fail of the process.

myErrors

protected java.util.Vector **myErrors**
 Holds all errors. An error is a message about some occurrence which causes the process to fail.

Constructor Detail

AbstractProcessable

public **AbstractProcessable**()

Method Detail

addComment

protected void **addComment**(java.lang.String msg)
 Adds the given message to my comments. A comment is an assessment free message.
Parameters:
 msg - The new comment.

addError

protected void **addError**(java.lang.String msg)

Adds the given message to my errors. An error is a message about some occurrence which causes the process to fail.
Parameters:
 msg - The new error.

addWarning

protected void **addWarning**(java.lang.String msg)
 Adds the given message to my warnings. A warning is a message about some occurrence which usually should not happen, but does not cause the fail of the process.
Parameters:
 msg - The new warning.

getComments

public java.lang.String[] **getComments**()
 Returns all comments. A comment is an assessment free message.
Specified by:
[getComments](#) in interface [Processable](#)
Returns:
 All comments.

getErrors

public java.lang.String[] **getErrors**()
 Returns all errors. An error is a message about some occurrence which causes the process to fail.
Specified by:
[getErrors](#) in interface [Processable](#)
Returns:
 All errors.

getName

public java.lang.String **getName**()
 Returns the name of the current object.
Specified by:
[getName](#) in interface [Processable](#)
Returns:
 The name of the current object.

getState

public int **getState**()
 Returns the current state of process. Possible states are defined in [Processable's Field Summary](#).
Specified by:
[getState](#) in interface [Processable](#)
Returns:
 The current state.

getWarnings

public java.lang.String[] **getWarnings**()
 Returns all warnings. A warning is a message about some occurrence which usually should not happen, but does not cause the fail of the process.
Specified by:
[getWarnings](#) in interface [Processable](#)
Returns:
 All warnings.

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
 SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
 SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.pro
Class KeyGenProcessable

java.lang.Object
 |
 +-de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable
 |
 +-de.tud.cdc.flexiTrust.ca.pro.KeyGenProcessable

public class **KeyGenProcessable**
 extends [AbstractProcessable](#)

Generates a key pair for given algorithm with given key length.

Version:
 \$Id: KeyGenProcessable.java,v 1.16 2000/12/29 21:53:20 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable

[myComments](#), [myErrors](#), [myName](#), [myNextStep](#), [myState](#), [myWarnings](#)

Constructor Summary

[KeyGenProcessable](#)(java.lang.String name, java.lang.String keyAlg, int keyLength)
 Constructs a new instance.

Method Summary

void	cont (Processor cpu, java.lang.Object[] params)
	Continues the process.
void	expire (Processor pro, java.lang.String msg, java.lang.Object[] params)
	Expires the process.
java.security.KeyPair	getKeyPair ()
	Returns the generated key pair.
java.lang.String	showKeyPair ()
	Returns a string representation of generated key pair.

A.5 DE.TUD.CDC.FLEXITRUST.CA.PRO

Methods inherited from class [de.tud.cdc.flexitrust.ca.pro.AbstractProcessable](#)
[addComment](#), [addError](#), [addWarning](#), [getComments](#), [getErrors](#), [getName](#), [getState](#), [getWarnings](#)

Methods inherited from class [java.lang.Object](#)
[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Constructor Detail

KeyGenProcessable

```
public KeyGenProcessable(java.lang.String name,
                        java.lang.String keyAlg,
                        int keyLength)
```

Constructs a new instance.

Parameters:
 name - The name of this instance.
 keyAlg - The algorithm to generate keys for.
 keyLength - The length of the new keys.

Method Detail

cont

```
public void cont(Processor cpu,
                java.lang.Object[] params)
```

Continues the process. This method is called by a [Processor](#) when this Processable is scheduled for being processed.

Parameters:
 pro - The current [Processor](#).
 params - These parameters vary with the current step.

- step = 1:
 - params[0] = byte[], my signature as DER encoded codec.pkcs7.SignedData object.
- step = 2:
 - params is ignored.
- step = 3:
 - params is ignored.

expire

```
public void expire(Processor pro,
                  java.lang.String msg,
                  java.lang.Object[] params)
```

Expires the process. Called when an unrecoverable error occurred, or called by a [Dormitory](#) when the Processable is frozen too long (while waiting for a [Condition](#) to be fulfilled). Some cleanup is done and the Processable exports itself with an adequate error message.

Parameters:
 pro - The current [Processor](#).
 msg - A message describing the cause of expiration.
 params - Ignored.

getKeyPair

```
public java.security.KeyPair getKeyPair()
```

Returns the generated key pair. If key pair was not generated yet null is returned.

Returns:
 The generated key pair.

showKeyPair

```
public java.lang.String showKeyPair()
```

Returns a string representation of generated key pair.

Returns:
 A string representation of generated key pair.

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexitrust.ca.pro
Class X509CertProcessable

```
java.lang.Object
|
|--de.tud.cdc.flexitrust.ca.pro.AbstractProcessable
|   |
|   |--de.tud.cdc.flexitrust.ca.pro.X509CertProcessable
```

public class **X509CertProcessable**
 extends [AbstractProcessable](#)

Generates a X509Certificate.

Version:
 \$Id: X509CertProcessable.java,v 1.17 2000/12/29 21:53:21 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class [de.tud.cdc.flexitrust.ca.pro.AbstractProcessable](#)
[myComments](#), [myErrors](#), [myName](#), [myNextStep](#), [myState](#), [myWarnings](#)

Constructor Summary

```
X509CertProcessable(java.lang.String name, java.lang.String id,
                    java.lang.String dn, java.util.Calendar validF, java.util.Calendar validU,
                    java.security.PublicKey pubk, java.security.PrivateKey privKey)
```

Creates new instance of this class.

Method Summary

void	cont (Processor cpu, java.lang.Object[] params)
	Continues the process.
void	createCert (CaCoreProcessor cpu)
	Creates a new X509Certificate.
void	expire (Processor pro, java.lang.String msg, java.lang.Object[] params)
	Expires the process.
codec.x509.X509Certificate	getCertificate ()

	Returns the created certificate.
java.lang.String	showCert ()
	Returns a string representation of created cert.

Methods inherited from class [de.tud.cdc.flexitrust.ca.pro.AbstractProcessable](#)
[addComment](#), [addError](#), [addWarning](#), [getComments](#), [getErrors](#), [getName](#), [getState](#), [getWarnings](#)

Methods inherited from class [java.lang.Object](#)
[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Constructor Detail

X509CertProcessable

```
public X509CertProcessable(java.lang.String name,
                        java.lang.String id,
                        java.lang.String dn,
                        java.util.Calendar validF,
                        java.util.Calendar validU,
                        java.security.PublicKey pubk,
                        java.security.PrivateKey privKey)
```

Creates new instance of this class.

Parameters:
 id - Unique ID of new cert's subject.
 dn - Distinguished name of new cert's subject.
 validF - New cert's valid from.
 validU - New cert's valid to.
 pubk - The public key of new cert.

Method Detail

cont

```
public void cont(Processor cpu,
                java.lang.Object[] params)
```

Continues the process. This method is called by a [Processor](#) when this Processable is scheduled for being processed.

Parameters:
 pro - The current [Processor](#).
 params - These parameters vary with the current step.

A JAVADOCS

- step = 1:
 - params[0] = byte[], my signature as DER encoded codec.pkcs7.SignedData object.
- step = 2:
 - params is ignored.
- step = 3:
 - params is ignored.

createCert

```
public void createCert(CaCoreProcessor cpu)
    throws java.lang.Exception
    Creates a new X509Certificate.
Parameters:
cpu - The hosting {de.tud.cdc.flexiTrust.ca.app.CaCoreProcessor}.
Throws:
java.lang.Exception - Exception handling is delegated to caller.
```

expire

```
public void expire(Processor pro,
    java.lang.String msg,
    java.lang.Object[] params)
    Expires the process. Called when an unrecoverable error occurred, or called by a
    Dormitory when the Processable is frozen too long (while waiting for a Condition
    to be fulfilled). Some cleanup is done and the Processable exports itself with an
    adequate error message.
Parameters:
pro - The current Processor.
msg - A message describing the cause of expiration.
params - Ignored.
```

getCertificate

```
public codec.x509.X509Certificate getCertificate()
    Returns the created certificate.
Returns:
The created cert.
```

showCert

```
public java.lang.String showCert()
    Returns a string representation of created cert.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.pro
Class X509CrlProcessable

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable
|
|--de.tud.cdc.flexiTrust.ca.pro.X509CrlProcessable
```

public class X509CrlProcessable
extends [AbstractProcessable](#)

Revokes some X509Certificates and fetches an updated CRL.

Version:
Std: X509CrlProcessable.java,v 1.15 2000/12/29 21:53:21 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

See Also:
[Serialized Form](#)

Fields inherited from class de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable
[myComments](#), [myErrors](#), [myName](#), [myNextStep](#), [myState](#), [myWarnings](#)

Constructor Summary

```
X509CrlProcessable(java.lang.String name,
    codec.x509.CRLEntry[] crlEntries, java.util.Calendar nextUpdate)
    Constructs a new instance.
```

Method Summary

void	cont (Processor cpu, java.lang.Object[] params)	Continues the process.
void	expire (Processor pro, java.lang.String msg, java.lang.Object[] params)	Expires the process.
codec.x509.X509Crl	getCrl ()	Returns the new lean crl.
codec.x509.CRLEntry[]	getToBeRevokedCrlEntries ()	Returns the new revocations.

```
java.lang.String showToBeRevokedCrlEntries()
    Returns a String representation of new revocations.
```

Methods inherited from class de.tud.cdc.flexiTrust.ca.pro.AbstractProcessable
[addComment](#), [addError](#), [addWarning](#), [getComments](#), [getErrors](#), [getName](#), [getState](#), [getWarnings](#)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

X509CrlProcessable

```
public X509CrlProcessable(java.lang.String name,
    codec.x509.CRLEntry[] crlEntries,
    java.util.Calendar nextUpdate)
    Constructs a new instance.
Parameters:
name - The name of the new instance.
crlEntries - The new revocations.
```

Method Detail

cont

```
public void cont(Processor cpu,
    java.lang.Object[] params)
    Continues the process. This method is called by a Processor when this Processable
    is scheduled for being processed.
Parameters:
pro - The current Processor.
params - These parameters vary with the current step.
```

- step = 1:
 - params[0] = byte[], my signature as DER encoded codec.pkcs7.SignedData object.
- step = 2:
 - params is ignored.
- step = 3:
 - params is ignored.

expire

```
public void expire(Processor pro,
    java.lang.String msg,
    java.lang.Object[] params)
    Expires the process. Called when an unrecoverable error occurred, or called by a
    Dormitory when the Processable is frozen too long (while waiting for a Condition
    to be fulfilled). Some cleanup is done and the Processable exports itself with an
    adequate error message.
Parameters:
pro - The current Processor.
msg - A message describing the cause of expiration.
params - Ignored.
```

getCrl

```
public codec.x509.X509Crl getCrl()
    Returns the new lean crl.
Returns:
The new lean crl.
```

getToBeRevokedCrlEntries

```
public codec.x509.CRLEntry[] getToBeRevokedCrlEntries()
    Returns the new revocations.
Returns:
The new revocations.
```

showToBeRevokedCrlEntries

```
public java.lang.String showToBeRevokedCrlEntries()
    Returns a String representation of new revocations.
Returns:
A String representation of new revocations.
```

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)

[SUMMARY](#) [INNER](#) [FIELD](#) [CONSTR](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONSTR](#) [METHOD](#)

A.6 de.tud.cdc.flexiTrust.ca.util

```

RoundRobinThreadScheduler
-myStarted:boolean=false
-myRtTimePeriod:long=25
-mySleepers:Thread=new Thread()
+schedule(tp:long):void
    
```

```

KbdHelper
+quest(msg:String):boolean
+readLine(msg:String):String
+waitFor(target:String,msg:Str
    
```

```

CaStockHelper
-myCaStock:CaStockRmi=null
+addCert(serial:BigInteger,cer
+getCert(subject:String):byte[]
+getNextSerial(issuer:String):
+getPrivateKey(issuer:String):Pri
+getProperty(alias:String):Str
+getPubKey(principal:String):F
+getSecProv(principal:String):
+getSecProvs():Provider[]
+getSigAlg(principal:String):S
+updateCrl(crlEntries:Vector,n
    
```

```

LogHelper
-myLogger:CaCoreLoggerRmi=null
-myLogLevel:int=0
-myLinesSep:String="\n"
+log(level:int,who:String,msg:
    
```

```

Pkcs12Helper
+createPkcs12(pr_key:PrivateKe
-getFriendlyName(dn:String,is_
-parseDN(attr:String,dn:String
    
```

```

ThrowableHandler
+exit(errCode:int,t:Throwable,
+exit(t:Throwable):void
+queryExit(errCode:int,t:Thro
+queryExit(t:Throwable):void
    
```

Overview Package [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER FIELD](#) [CONST](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONST](#) [METHOD](#)

de.tud.cdc.flexiTrust.ca.util

Class CaStockHelper

```

java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.util.CaStockHelper
    
```

public class **CaStockHelper**
extends java.lang.Object

Eases and optimizes the usage of [CaStock](#). It is necessary for CaStockHelper's task that all classes within one JVM use the same instance of CaStockHelper. Therefore and to ease the use of this class all CaStockHelper methods are static. When this class is loaded (when first referenced) a JINI lookup for [CaStock](#) is done and the service is fetched. All further calls of CaStockHelper methods (e.g. by roaming [Processables](#)) do not cause further JINI activities.

Version:
Sid: CaStockHelper.java,v 1.39 2001/01/17 16:18:28 wiesmaie Exp S

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

CaStockHelper()

Method Summary

static void	addCert (java.math.BigInteger serial, byte[] cert) Adds the given certificate to list of all certificates.
static byte[]	getCert (java.lang.String subject) Returns the certificate belonging to given subject.
static java.math.BigInteger	getNextSerial (java.lang.String issuer) Returns the serial next to be used for given issuer.
static java.security.PrivateKey	getPrivateKey (java.lang.String issuer) Returns the private key of given issuer.
static java.lang.String	getProperty (java.lang.String alias) Returns the property with given alias.

static java.security.PublicKey	getPubKey (java.lang.String principal) Returns the public key for given principal.
static java.security.Provider	getSecProv (java.lang.String principal) Returns the security provider used by given issuer.
static java.security.Provider[]	getSecProvs () Returns all security providers used by known issuers.
static java.lang.String	getSigAlg (java.lang.String principal) Returns the signature algorithm of given issuer.
static byte[]	updateCrl (java.util.Vector crlEntries, java.util.Calendar nextUpdate) Updates the crl with given crl entries and returns the new lean crl (DER encoded).

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CaStockHelper

```

public CaStockHelper()
    
```

Method Detail

addCert

```

public static void addCert(java.math.BigInteger serial,
                           byte[] cert)
                           throws java.rmi.RemoteException
    
```

Adds the given certificate to list of all certificates.
Parameters:
serial - The serial number of the new cert.
cert - The new cert (DER encoded).
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getCert

```

public static byte[] getCert(java.lang.String subject)
    
```

throws java.rmi.RemoteException
Returns the certificate belonging to given subject. The certificate is returned DER encoded.
Parameters:
subject - Return the cert for whom?
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getNextSerial

public static java.math.BigInteger **getNextSerial**(java.lang.String issuer) throws java.rmi.RemoteException
Returns the serial next to be used for given issuer.
Parameters:
issuer - The issuer to get the next serial for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPrivKey

public static java.security.PrivateKey **getPrivKey**(java.lang.String issuer) throws java.rmi.RemoteException
Returns the private key of given issuer.
Parameters:
issuer - The issuer to get the private key for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getProperty

public static java.lang.String **getProperty**(java.lang.String alias) throws java.rmi.RemoteException
Returns the property with given alias.
Parameters:
alias - The alias to get the property for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getPubKey

public static java.security.PublicKey **getPubKey**(java.lang.String principal) throws java.rmi.RemoteException
Returns the public key for given principal.
Parameters:
principal - The principal to fetch the public key for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProv

public static java.security.Provider **getSecProv**(java.lang.String principal) throws java.rmi.RemoteException
Returns the security provider used by given issuer.
Parameters:
issuer - The issuer to fetch the provider for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSecProvs

public static java.security.Provider[] **getSecProvs**() throws java.rmi.RemoteException
Returns all security providers used by known issuers.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

getSigAlg

public static java.lang.String **getSigAlg**(java.lang.String principal) throws java.rmi.RemoteException
Returns the signature algorithm of given issuer.
Parameters:
issuer - The issuer to fetch the signature algorithm for.
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

updateCrl

public static byte[] **updateCrl**(java.util.Vector crlEntries, java.util.Calendar nextUpdate) throws java.rmi.RemoteException

Updates the crl with given crl entries and returns the new lean crl (DER encoded).
Parameters:
crlEntries - The new revocations.
nextUpdate - The time when next update will be made (worst case).
Throws:
java.rmi.RemoteException - Any occurring Exception is rethrown as nested Exception of a RemoteException.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

de.tud.cdc.flexiTrust.ca.util Class KbdHelper

java.lang.Object
|--de.tud.cdc.flexiTrust.ca.util.KbdHelper

public class KbdHelper
extends java.lang.Object

Provides some functionality for IO via standard in and standard out. In order to ease the use of this class all kbdHelper methods are static.

Version:
Std: KbdHelper.java, v 1.19 2001/01/17 16:18:28 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

KbdHelper()

Method Summary

static boolean	quest (java.lang.String msg)	Displays the given question and returns the answer.
static java.lang.String	reading (java.lang.String msg)	Displays the given message and reads a line of text.
static void	waitFor (java.lang.String target, java.lang.String msg)	Blocks until given target is read in.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

A.6 DE.TUD.CDC.FLEXITRUST.CA.UTIL

KbdHelper

```
public KbdHelper()
```

Method Detail

quest

```
public static boolean quest(java.lang.String msg)
```

Displays the given question and returns the answer.

Parameters:
msg - The question to be displayed.

Returns:
The answer.

readLine

```
public static java.lang.String readLine(java.lang.String msg)
```

Displays the given message and reads a line of text.

Parameters:
msg - The message to display.

Returns:
The line of text read in.

waitFor

```
public static void waitFor(java.lang.String target,
                          java.lang.String msg)
```

Blocks until given target is read in.

Parameters:
target - The target to wait for.
msg - The message to display.

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.util
Class LogHelper

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.util.LogHelper
```

public class LogHelper
extends java.lang.Object

Eases and optimizes the usage of [CaCoreLogger](#). It is necessary for LogHelper's task that all classes within one JVM use the same instance of LogHelper. Therefore and to ease the use of this class all LogHelper methods are static. When this class is loaded (when first referenced) a JNI lookup for [CaCoreLogger](#) is done and the service is fetched. All further calls of LogHelper methods (e.g. by roaming [Processables](#)) do not cause further JNI activities. In addition the level of the log message is tested and the message is given to the [CaCoreLogger](#) only if it will really be saved.

Version:
\$Id: LogHelper.java,v 1.24 2001/01/17 16:18:28 wiesmaie Exp \$

Author:
Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

LogHelper()	
-----------------------------	--

Method Summary

static void log (int level, java.lang.String who, java.lang.String msg)	Adds the given message to log.
---	--------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

LogHelper

```
public LogHelper()
```

Method Detail

log

```
public static void log(int level,
                      java.lang.String who,
                      java.lang.String msg)
```

Adds the given message to log.

Parameters:
level - The level of the given message. If the message's level is greater than the [Logger](#)'s level it will not be logged.
who - Who wants to log the message.
msg - The message to log.

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Overview](#) [Package](#) [Class Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
SUMMARY: [INNER](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#) DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

de.tud.cdc.flexiTrust.ca.util
Class Pkcs12Helper

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.util.Pkcs12Helper
```

public class Pkcs12Helper
extends java.lang.Object

Provides a mechanism to create a PKCS#12 security file. In order to ease the use of this class all Pkcs12Helper methods are static.

Version:
\$Id: Pkcs12Helper.java,v 1.17 2001/01/17 16:18:28 wiesmaie Exp \$

Author:
Markus Tak - TU Darmstadt/CDC, Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

Pkcs12Helper()	
--------------------------------	--

Method Summary

static void createPkcs12 (java.security.PrivateKey pr_key, codec.X509.X509Certificate cert, codec.X509.X509Certificate[] chain, char[] pin, java.lang.String filename)	Creates a PKCS#12 security file.
--	----------------------------------

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Pkcs12Helper

A JAVADOCS

```
public Pkcs12Helper()
```

Method Detail

createPkcs12

```
public static void createPkcs12(java.security.PrivateKey pr_key,
                               codec.X509Certificate cert,
                               codec.X509Certificate[] chain,
                               char[] pin,
                               java.lang.String filename)
```

Creates a PKCS#12 security file.

Parameters:
 pr_key - the private key.
 cert - The certificate.
 chain - The certificate chain.
 pin - The PIN for privacy.
 filename - The name of the new security file.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.util
Class RoundRobinThreadScheduler

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.util.RoundRobinThreadScheduler
```

public class **RoundRobinThreadScheduler**
 extends java.lang.Object

Schedules all Threads in system having priority less than Thread.MAX_PRIORITY in a round robin manner. In order to ease the use of this class all RoundRobinThreadScheduler methods are static.

Version:
 \$Id: RoundRobinThreadScheduler.java,v 1.44 2001/01/17 16:18:28 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

[RoundRobinThreadScheduler\(\)](#)

Method Summary

static void [schedule](#)(long tp)
 Starts the scheduling.

Methods inherited from class java.lang.Object
 clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

RoundRobinThreadScheduler

```
public RoundRobinThreadScheduler()
```

Method Detail

schedule

```
public static void schedule(long tp)
    throws java.lang.Exception
```

Starts the scheduling.

Parameters:
 tp - The time slice every >CODE>Thread is given.

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

[Overview](#) [Package](#) [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)
[PREV CLASS](#) [NEXT CLASS](#) [FRAMES](#) [NO FRAMES](#)
[SUMMARY: INNER | FIELD | CONSTR | METHOD](#) [DETAIL: FIELD | CONSTR | METHOD](#)

de.tud.cdc.flexiTrust.ca.util
Class ThrowableHandler

```
java.lang.Object
|
|--de.tud.cdc.flexiTrust.ca.util.ThrowableHandler
```

public class **ThrowableHandler**
 extends java.lang.Object

Provides some methods for handling Throwables. In order to ease the use of this class all ThrowableHandler methods are static.

Version:
 \$Id: ThrowableHandler.java,v 1.17 2001/01/17 16:18:29 wiesmaie Exp \$

Author:
 Alex Wiesmaier - TU Darmstadt/CDC

Constructor Summary

[ThrowableHandler\(\)](#)

Method Summary

static void [exit](#)(int errCode, java.lang.Throwable t, java.io.PrintStream ps)
 Prints the throwable's message to given PrintStream and exits the program.

static void [exit](#)(java.lang.Throwable t)
 Prints the throwable's message to System.err and exits the program.

static void [queryExit](#)(int errCode, java.lang.Throwable t, java.io.PrintStream ps, java.io.InputStream is)
 Blocks the current Thread, prints the throwables message to given PrintStream, asks on given InputStream whether the user wants to exit or continue the program and acts due to user's answer.

static void [queryExit](#)(java.lang.Throwable t)
 Blocks the current Thread, prints the throwable's message to System.err, asks on system.out whether the user wants to exit or continue the program and acts due to user's answer.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
ThrowableHandler
public ThrowableHandler ()
Method Detail
exit
public static void exit (int errCode, java.lang.Throwable t, java.io.PrintStream ps)
Prints the throwable's message to given PrintStream and exits the program.
Parameters: errCode - The error code given to shell when system exits. t - A Throwable. ps - A PrintStream for printing messages.
exit
public static void exit (java.lang.Throwable t)
Prints the throwable's message to System.err and exits the program.
Parameters: t - A Throwable.
queryExit
public static void queryExit (int errCode, java.lang.Throwable t, java.io.PrintStream ps, java.io.InputStream is)
Blocks the current Thread, prints the throwables message to given PrintStream, asks on given InputStream whether the user wants to exit or continue the program and acts due to user's answer.
Parameters: errCode - The error code given to shell when system exits. t - A Throwable. ps - A PrintStream for printing messages. is - An InputStream for reading user commands.

queryExit
public static void queryExit (java.lang.Throwable t)
Blocks the current Thread, prints the throwable's message to System.err, asks on system.out whether the user wants to exit or continue the program and acts due to user's answer.
Parameters: t - A Throwable.
Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER FIELD CONSTR METHOD DETAIL: FIELD CONSTR METHOD

Literaturverzeichnis

- [Buc99] Johannes Buchmann. *Einführung in die Kryptographie*. Springer-Verlag, 1999.
- [Car00] Open Card. *The OpenCard Framework*. <http://www.opencard.org>, 2000.
- [DS00] Jens Dambruch and Markus Schuster. *Entwurf und Implementierung einer offenen Schnittstelle für die Online-Registrierung (ORA)*. <http://www.student.informatik.tu-darmstadt.de:8080/jensd/krypto/>, 2000.
- [Fla98] David Flanagan. *Java in a Nutshell*. O'Reilly, 1998.
- [For92] Internet Engineering Task Force. *Technical Overview of Directory Services: Using the X.500 Protocol*. <http://www.ietf.cnri.reston.va.us/rfc/rfc1309.txt>, 1992.
- [For99] Internet Engineering Task Force. *Internet X.509 Public Key Infrastructure: Certificate and CRL Profile*. <http://www.ietf.cnri.reston.va.us/rfc/rfc2459.txt>, 1999.
- [FS98] M. Fowler and K. Scott. *UML Distilled*. Addison–Wesley, 1998.
- [fSidI00] Bundesamt für Sicherheit in der Informationstechnik. *Maßnahmenkatalog des BSI*. <http://www.bsi.de/aufgaben/projekte/pbdigsig/index.htm>, 2000.
- [fWuT97] Bundesministerium für Wirtschaft und Technik. *Deutsches Signaturgesetz, Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste und Signaturverordnung*. <http://www.iid.de/iukdg>, 1997.
- [IT97] ITU-T. *OSI Networking and System Aspects: Abstract Syntax Notation One (ASN.1)*. <http://www.itu.int/publications/online.htm>, 1997.
- [jGu00] jGuru. *Fundamentals of RMI — Short Course*. <http://developer.java.sun.com/developer/onlineTraining/Downloads/rmi.zip>, 2000.

LITERATURVERZEICHNIS

- [Knu98] Jonathan Knudsen. *Java Cryptography*. O'Reilly & Associates, Inc., 1998.
- [Lab00] RSA Laboratories. *Public-Key Cryptography Standards*. <http://www.rsalabs.com/pkcs>, 2000.
- [Mic99a] SUN Microsystems. *Jini Technology Core Platform Compatibility Kit Harness Specification*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> JiniTM Technology Core Platform Compatibility Kit, 1999.
- [Mic99b] SUN Microsystems. *Jini Technology Core Platform Compatibility Kit Test Specification*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> JiniTM Technology Core Platform Compatibility Kit, 1999.
- [Mic99c] SUN Microsystems. *Passing the TCK*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> JiniTM Technology Core Platform Compatibility Kit, 1999.
- [Mic00a] SUN Microsystems. *A Collection of Jini Technology Helper Utilities and Services Specifications*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> Jini 1.1Beta, 2000.
- [Mic00b] SUN Microsystems. *The Java Tutorial — A practical guide for programmers*. <http://java.sun.com/docs/books/tutorial/index.html>, 2000.
- [Mic00c] SUN Microsystems. *JavaSpaces Service Specification*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> Jini 1.1Beta, 2000.
- [Mic00d] SUN Microsystems. *Jini Technology Core Platform Specification*. <http://developer.java.sun.com/developer/restricted/jinisite/choicejini.html#> Jini 1.1Beta, 2000.
- [Mic00e] SUN Microsystems. *Jini Technology Manual Pages, v1.1*. <http://www.sun.com/jini/specs/>, 2000.
- [Mic00f] SUN Microsystems. *rmid - The Java RMI Activation System Daemon*. <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/rmid.html>, 2000.
- [MZ00a] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series I, Part I: What is Jiro Technology?* http://www.jiro.com/education/tutorials/articles/what_is_jiro.pdf, 2000.

LITERATURVERZEICHNIS

- [MZ00b] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series I, Part II: What is a Management Server?*
http://www.jiro.com/education/tutorials/articles/management_server.pdf, 2000.
- [MZ00c] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series I, Part III: What are Management Components?*
http://www.jiro.com/education/tutorials/articles/articles/management_component.pdf, 2000.
- [MZ00d] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series I, Sidebar I: Jini Technology for Jiro Technology Developers.*
http://www.jiro.com/education/tutorials/articles/jini_for_jiro.pdf, 2000.
- [MZ00e] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series II, Part I: What are Jiro Technology Base Services?*
http://www.jiro.com/education/tutorials/articles/base_services_intro.pdf, 2000.
- [MZ00f] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series II, Part II: What is the Jiro Log Service?*
http://www.jiro.com/education/tutorials/articles/log_service.pdf, 2000.
- [MZ00g] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series II, Part III: What is the Jiro Technology Event Service?*
http://www.jiro.com/education/tutorials/articles/event_service.pdf, 2000.
- [MZ00h] Jay Miller and Kelby Zorgdrager. *The Jiro Technology On-Line Tutorial — Series II, Part IV: The Jiro Technology Scheduling Service.*
http://www.jiro.com/education/tutorials/articles/scheduling_service.pdf, 2000.
- [Oak98] Scott Oaks. *Java Security*. O'Reilly & Associates, Inc., 1998.
- [Sat00] Vladislav Satanovski. *KeyStore Manager*. Technische Universität Darmstadt, Fachbereich Theoretische Informatik, 2000.
- [Sec93] RSA Security. *A Layman's Guide to a Subset of ASN.1, BER, and DER*.
<ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc>, 1993.
- [Tak99] Markus Tak. *Public Key Infrastrukturen: Ein Java-basiertes Trustcenter*.
<ftp://ftp.informatik.tu-darmstadt.de/pub/TI/reports/tak.diplom.ps.gz>, 1999.
- [TUD98] Fachgebiet Theoretische Informatik Technische Universität Darmstadt, Fachbereich Informatik. *Seneca, eine Java-Bibliothek für den gleichzeitigen Zugriff auf software- und hardwarebasierte*

LITERATURVERZEICHNIS

Kryptographiemodule.

<http://www.informatik.tu-darmstadt.de/TI/Welcome.html>, 1998.

[Wor00] PC/SC Workgroup. *PC/SC Specifications.*

<http://www.pcscworkgroup.com>, 2000.

Index

- AbstractProcessable, 43
 - Implementierung, 43
- AbstractProcessable.java, 43
- Abstraktion, 15
- Administrator, 49
 - Initialphase, 49
- AdminKeys, 38
 - Implementierung, 38
 - Initialphase, 49
- adminKeys, 53
- AdminKeys.java, 38
- Advanced Processing Infrastructure,
 - 16, 22, 63
 - Design, 16
 - Implementierung, 21
 - Javadoc, 63
 - Package, 22
- all, 54
- Analyse, 12
- API, 16
- Application, 31
 - Implementierung, 31
 - Package, 31
- Application–Package, 31
 - Javadoc, 69
- Aufgabenstellung, 12
- Ausblick, 60
- Auto–Mode, 51
- autoMode, 52
- Basis–CA, 12
- Beendigung, 51
- Betrieb, 47, 49, 50
 - Auto–Mode, 51
 - Beendigung, 51
 - Betrieb, 50
- GUI–Mode, 51
- RA, 50
- Registration–Authority, 50
- Start, 49
- CaCoreFrame, 35
 - Implementierung, 35
- CaCoreFrame.java, 35
- CaCoreLogger, 39
 - Implementierung, 39
- CaCoreLogger.java, 39
- CaCoreLoggerRmi, 39
 - Implementierung, 39
- CaCoreLoggerRmi.java, 39
- CaCoreMonitor
 - Implementierung, 35
- CaCoreMonitor.java, 35
- CaCoreProcessor, 31
 - Implementierung, 31
- CaCoreProcessor.java, 31
- CaCoreProcessorQueue, 31
 - Implementierung, 31
- CaCoreProcessorQueue.java, 31
- CaCoreProcessorQueueRmi, 33
 - Implementierung, 33
- CaCoreProcessorQueueRmi.java, 33
- CaException, 36
 - Implementierung, 36
- CaException.java, 36
- CaKeys, 38
 - Implementierung, 38
 - Initialphase, 49
- caKeys, 53
- CaKeys.java, 38
- CaStock, 34
 - Implementierung, 34

- caStock, 53
- CaStock.java, 34
- CaStockHelper, 44
 - Implementierung, 44
- CaStockHelper.java, 44
- CaStockRmi, 35
 - Implementierung, 35
- CaStockRmi.java, 35
- ccProc, 54
- Certification–Authority, 10
- cls, 53
- Compileraufrufe, 53
 - Makefile, 53
- Condition, 25
 - Design, 20
 - Implementierung, 25
- Condition.java, 25

- de.tud.cdc.flexiTrust.ca.api
 - Implementierung, 22
 - Javadoc, 63
- de.tud.cdc.flexiTrust.ca.app
 - Implementierung, 31
 - Javadoc, 69
- de.tud.cdc.flexiTrust.ca.init
 - Implementierung, 36
 - Javadoc, 82
- de.tud.cdc.flexiTrust.ca.log
 - Implementierung, 39
 - Javadoc, 84
- de.tud.cdc.flexiTrust.ca.pro
 - Implementierung, 41
 - Javadoc, 89
- de.tud.cdc.flexiTrust.ca.util
 - Implementierung, 44
 - Javadoc, 93
- Design
 - Advanced Processing Infrastructure, 16
 - Condition, 20
 - Dormitory, 19
 - Exporter, 19
 - FlexiTRUST, 27
 - Importer, 19
 - Kompression, 21
 - Logger, 20
 - Processable, 16
 - Processor, 18
 - Queue, 18
 - Stock, 20
 - Synchronizable, 21
- doc, 53
- Dormitory, 25
 - Design, 19
 - Implementierung, 25
- Dormitory.java, 25

- Eigenschaften, 47
- Einleitung, 8
- Erweiterbarkeit, 15
- Erweiterung, 57
- Exporter, 25
 - Design, 19
 - Implementierung, 25
- Exporter.java, 25

- Fähigkeiten, 47
- FileLogWriter, 41
 - Implementierung, 41
- FileLogWriter.java, 41
- Flexibilität, 14
- FlexiTRUST, 27
 - Design, 27
 - Implementierung, 30
- Fusion, 15

- gmake, 52
- GUI–Mode, 51
- guiMode, 52

- Implementierung, 21, 30
 - AbstractProcessable, 43
 - AdminKeys, 38
 - Advanced Processing Infrastructure, 21
 - Application, 31
 - CaCoreFrame, 35

INDEX

- CaCoreLogger, 39
- CaCoreLoggerRmi, 39
- CaCoreMonitor, 35
- CaCoreProcessor, 31
- CaCoreProcessorQueue, 31
- CaCoreProcessorQueueRmi, 33
- CaException, 36
- CaKeys, 38
- CaStock, 34
- CaStockHelper, 44
- CaStockRmi, 35
- Condition, 25
- de.tud.cdc.flexiTrust.ca.api, 22
- Dormitory, 25
- Exporter, 25
- FileLogWriter, 41
- FlexiTRUST, 30
- Importer, 24
- Initiation, 36
- JiniHelper, 26
- KbdHelper, 44
- KeyGenProcessable, 43
- Logger, 26
- Logging, 39
- LogHelper, 44
- Pkcs12Helper, 46
- Pkcs7Exporter, 33
- Pkcs7ExporterQueue, 34
- Pkcs7ExporterQueueRmi, 34
- Pkcs7Importer, 33
- Processable.java, 24
- Processables, 41
- Processor, 24
- Properties, 39, 41
- Queue, 24
- RoundRobinThreadScheduler, 46
- StdoutFileLogWriter, 41
- StdoutLogWriter, 41
- Stock, 25
- Synchronizable, 26
- ThrowableHandler, 46
- Utilities, 44
- X509CertProcessable, 43
- Implementierung
 - de.tud.cdc.flexiTrust.ca.app, 31
 - de.tud.cdc.flexiTrust.ca.init, 36
 - de.tud.cdc.flexiTrust.ca.log, 39
 - de.tud.cdc.flexiTrust.ca.pro, 41
 - de.tud.cdc.flexiTrust.ca.util, 44
- Importer, 24
 - Design, 19
 - Implementierung, 24
- Importer.java, 24
- Index, 102
- ini, 53
- Initialphase, 36, 49
 - Administrator, 49
 - AdminKeys, 49
 - CaKeys, 49
 - Issuer, 49
 - Package, 36
- Initiation, 36
 - Implementierung, 36
 - Package, 36
- Initiation–Package, 36
- Installation, 48
- Installationsvorgang, 48
- Issuer
 - Initialphase, 49
- Istzustand, 12
- javac, 53
- Javadoc
 - Advanced Processing Infrastructure, 63
 - Application–Package, 69
 - de.tud.cdc.flexiTrust.ca.api, 63
 - de.tud.cdc.flexiTrust.ca.app, 69
 - de.tud.cdc.flexiTrust.ca.init, 82
 - de.tud.cdc.flexiTrust.ca.log, 84
 - de.tud.cdc.flexiTrust.ca.pro, 89
 - de.tud.cdc.flexiTrust.ca.util, 93
- Javadocs, 62
- JiniHelper, 26
 - Implementierung, 26
- JiniHelper.java, 26

- KbdHelper, 44
 - Implementierung, 44
- KbdHelper.java, 44
- Kern-CA, 11, 13
- KeyGenProcessable, 43
 - Implementierung, 43
- KeyGenProcessable.java, 43
- Kompression
 - Design, 21
- Konventionen, 8

- Leseanweisung, 8
- Literaturverzeichnis, 97
- Logger, 26
 - Design, 20
 - Implementierung, 26
- logger, 53
- Logger.java, 26
- Logging, 39
 - Implementierung, 39
 - Package, 39
- Logging-Package, 39
- LogHelper, 44
 - Implementierung, 44
- LogHelper.java, 44
- logLevel, 52
- logWriter, 52
- lus, 53
- lusHost, 52

- make, 52
- Makefile, 52
 - adminKeys, 53
 - all, 54
 - autoMode, 52
 - caKeys, 53
 - caStock, 53
 - ccProc, 54
 - cls, 53
 - Compileraufrufe, 53
 - doc, 53
 - guiMode, 52
 - ini, 53
 - javac, 53
 - logger, 53
 - logLevel, 52
 - logWriter, 52
 - lus, 53
 - lusHost, 52
 - p7Exp, 54
 - p7ExpQueue, 54
 - p7Imp, 54
 - policy, 52
 - prc, 54
 - procQueue, 54
 - propFile, 52
 - reggie, 53
 - rmic, 53
 - rmid, 53
 - sgl, 54
 - Startvorgänge, 53
 - Target, 52
 - tst, 54
 - Variablen, 52
- Monitor, 21

- p7Exp, 54
- p7ExpQueue, 54
- p7Imp, 54
- Personalisierung, 11
- Pkcs12Helper, 46
 - Implementierung, 46
- Pkcs12Helper.java, 46
- Pkcs7Exporter, 33
 - Implementierung, 33
- Pkcs7Exporter.java, 33
- Pkcs7ExporterQueue, 34
 - Implementierung, 34
- Pkcs7ExporterQueue.java, 34
- Pkcs7ExporterQueueRmi, 34
 - Implementierung, 34
- Pkcs7ExporterQueueRmi.java, 34
- Pkcs7Importer, 33
 - Implementierung, 33
- Pkcs7Importer.java, 33
- PKI, 9

INDEX

- policy, 52
- prc, 54
- Processable, 24
 - Design, 16
- Processable.java, 24
 - Implementierung, 24
- Processables, 41
 - Implementierung, 41
 - Package, 41
- Processables–Package, 41
- Processor, 24
 - Design, 18
 - Implementierung, 24
- Processor.java, 24
- procQueue, 54
- Properties, 39, 41, 55
 - Administrator, 57
 - AdminKeys, 56
 - CA, 56
 - CaKeys, 55
 - global, 55
 - Implementierung, 39, 41
 - Logging, 57
- Properties–File, 55
- Properties.java, 39, 41
- propFile, 52
- Public–Key–Infrastrukturen, 9
- Public–Key–Kryptographie, 9

- Queue, 24
 - Design, 18
 - Implementierung, 24
- Queue.java, 24

- RA, 50
- reggie, 53
- Registration–Authority, 50
- Registrierung, 10
- Revokation, 11
- rmic, 53
- rmid, 53
- RoundRobinThreadScheduler, 46
 - Implementierung, 46

- RoundRobinThreadScheduler.java, 46

- Schlüsselerzeugung, 10
- sgl, 54
- Skalierbarkeit, 14
- Sollzustand, 12
- Start, 49
 - Importer, 50
 - Lookup–Service, 50
 - Processor, 50
 - Singletons, 50
- Startvorgänge, 53
 - Makefile, 53
- StdoutFileLogWriter, 41
 - Implementierung, 41
- StdoutFileLogWriter.java, 41
- StdoutLogWriter, 41
 - Implementierung, 41
- StdoutLogWriter.java, 41
- Stock, 25
 - Design, 20
 - Implementierung, 25
- Stock.java, 25
- Synchronizable, 26
 - Design, 21
 - Implementierung, 26
- Synchronizable.java, 26
- Systemanforderungen, 48
- Systemverhalten, 13

- Target, 52
 - adminKeys, 53
 - all, 54
 - caKeys, 53
 - caStock, 53
 - ccProc, 54
 - cls, 53
 - doc, 53
 - ini, 53
 - javac, 53
 - logger, 53
 - lus, 53
 - Makefile, 52

- p7Exp, 54
- p7ExpQueue, 54
- p7Imp, 54
- prc, 54
- procQueue, 54
- reggie, 53
- rmic, 53
- rmid, 53
- sgl, 54
- tst, 54
- Testsysteme, 58
- ThrowableHandler, 46
 - Implementierung, 46
- ThrowableHandler.java, 46
- tst, 54
- Utilities, 44
 - Implementierung, 44
 - Package, 44
- Utilities–Package, 44
- Variablen, 52
 - autoMode, 52
 - guiMode, 52
 - logLevel, 52
 - logWriter, 52
 - lusHost, 52
 - Makefile, 52
 - policy, 52
 - propFile, 52
- Verzeichnisdienst, 11
- Vorüberlegungen, 14
- Vorgehensweise, 15
- Vorwort, 3
- X509CertProcessable, 43
 - Implementierung, 43
- X509CertProcessable.java, 43
- X509CrlProcessable, 43
 - Implementierung, 43
- X509CrlProcessable.java, 43
- Zeitstempeldienst, 11
- Zertifizierung, 11