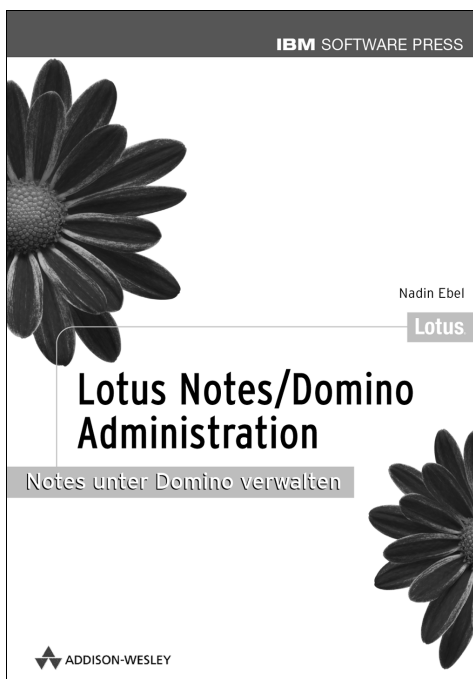


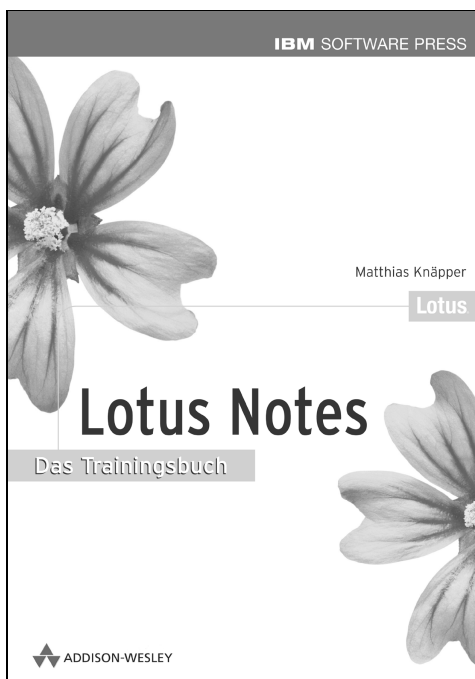
Java-Anwendungen für Notes/Domino entwickeln

IBM Software Press

In Zusammenarbeit mit der IBM Deutschland GmbH verlegt Addison-Wesley diese Reihe professioneller Bücher. Die Reihe deckt die Software WebSphere, DB2, Tivoli und Lotus ab. Die Funktionalitäten, Tools, Anwendungen und Entwicklungsumgebungen der jeweiligen Software werden von Autoren beschrieben, die sich beruflich mit der Software befassen.



Die Autorin, selbst erfahrene Administratorin, erklärt neben den konventionellen Themen der Administration vor allem auch die Internetanbindung, die Verwaltung eines Intranets und die Datensicherheit.



Matthias Knäpper führt seit Jahren Anwender im Bereich Lotus in die Geheimnisse des Programms ein. Seine sehr erfolgreichen Schulungen finden ihren Niederschlag in diesem Buch. Übersichtlich, klar gegliedert und verständlich werden Sie mit der Benutzung von Lotus Notes vertraut gemacht. Das Buch ist zum Selbststudium und zum Einsatz als Lehrwerk bestens geeignet.

Thomas Muhs

Java-Anwendungen für Notes/Domino entwickeln

Notes/Domino als Webserver



ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Die Informationen in diesem Produkt werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

Bei der Zusammenstellung von Texten und Abbildungen wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden.

Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Für Verbesserungsvorschläge und Hinweise auf Fehler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechanischen Wiedergabe und der Speicherung in elektronischen Medien.

Die gewerbliche Nutzung der in diesem Produkt gezeigten Modelle und Arbeiten ist nicht zulässig.

Fast alle Hardware- und Softwarebezeichnungen, die in diesem Buch erwähnt werden, sind gleichzeitig auch eingetragene Warenzeichen oder sollten als solche betrachtet werden.

Umwelthinweis:

Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.

10 9 8 7 6 5 4 3 2 1

06 05 04 03

ISBN 3-8273-1807-6

© 2003 by Addison-Wesley Verlag, ein Imprint der
Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten

Einbandgestaltung:	Barbara Thoben, Köln
Lektorat:	Rolf Pakendorf, rpakendorf@pearson.de
Korrektur:	mediaService, Siegen
Herstellung:	Ulrike Hempel, uhempel@pearson.de
Satz und Layout:	mediaService, Siegen
Druck und Verarbeitung:	Kösel, Kempten (www.KoeselBuch.de)

Printed in Germany

Inhaltsverzeichnis

Einleitung	15
1 Die Programmiersprache Java	17
1.1 Der Unterschied zwischen einer Interpreter- und einer Compilersprache	17
1.2 Die Virtual Machine	17
2 Einführung in die Javaprogrammierung	19
2.1 Die Elemente der Sprache Java	19
2.1.1 Der Aufbau eines Javaprogramms	19
2.1.2 Kommentare	19
2.1.3 Variablen und Datentypen	20
2.1.4 Garbage Collector	22
2.1.5 Elementaroperatoren	22
2.1.6 Verzweigungen	23
2.1.7 Schleifen	24
2.1.8 Packages verwenden	25
2.1.9 Zeichenketten	26
2.1.10 Nicht druckbare Sonderzeichen	27
2.2 Objektorientierte Softwareentwicklung	27
2.2.1 Objekte und Objektinstanzen	27
2.2.2 Verwendung von Klassen	28
2.2.3 Kapselung	30
2.2.4 Sichtbarkeit von Methoden und Eigenschaften	31
2.2.5 Vererbung	32
2.2.6 Überladung von Methoden	32
2.2.7 Abstrakte Klassen und Methoden	33
2.2.8 Statische Methoden und Variablen	33
2.2.9 Das Schlüsselwort final	35
2.2.10 Casting	35
2.2.11 Das Interface	35
2.2.12 Ausnahmebehandlung	36
3 Java Entwicklungsumgebungen	39
3.1 JDK	39
3.2 Visual Age	39
3.2.1 Erstellen eines Notes API Projekts	39
3.2.2 Aufnehmen des Jar Files im Classpath von Age	39
4 Javaprogrammierung unter Notes	41
4.1 Einsatzmöglichkeiten von Java unter Notes	41
4.1.1 Agentprogrammierung	41
4.1.2 Standalone Notes Anwendungen	49
4.1.3 Applet-Anwendungen	53

4.2	Datenbanken	57
4.2.1	Einleitung	57
4.2.2	Lokalisieren einer Notes Datenbank	57
4.2.3	Öffnen einer Datenbank	58
4.2.4	Erstellen, Komprimieren, Löschen und Replizieren einer Datenbank	68
4.2.5	Die wichtigsten Merkmale einer Datenbank	73
4.2.6	Die replikationsrelevanten Eigenschaften einer Datenbank	77
4.2.7	Cutoff-Verhalten	83
4.2.8	Die Eigenschaft Abstract	86
4.2.9	Die Eigenschaften der Datenbankvorlage	87
4.2.10	Das Arbeiten mit Datenbankkategorien	88
4.2.11	Datenbankgrößen	89
4.2.12	Volltextindex	93
4.2.13	Sonstige Datenbankeigenschaften	95
4.3	Dokumente	98
4.3.1	Einleitung	98
4.3.2	Zugriff auf Dokumente in einer Datenbank	98
4.3.3	Die Dokumentensammlung	102
4.3.4	Dokumente suchen	111
4.3.5	Anlegen eines Dokuments	114
4.3.6	Speichern eines Dokuments	114
4.3.7	Löschen eines Dokuments	115
4.3.8	Kopieren eines Dokuments	116
4.3.9	Versenden eines Dokuments	117
4.3.10	Profildokumente	119
4.3.11	Ermitteln aller Profildokumente mit einem bestimmten Profilnamen	120
4.4	Die Zugriffskontrollliste	121
4.4.1	Zugriff auf die Zugriffskontrollliste einer Datenbank	121
4.4.2	Bearbeiten der Einträge der Zugriffskontrollliste	122
4.4.3	Die Zugriffsrollen	132
4.4.4	Konsistente Zugriffskontrollliste	137
4.4.5	Internetzugriffsrechte	138
4.5	Items	139
4.5.1	Einleitung	139
4.5.2	Die Items eines Dokuments	139
4.6	RichText-Items	148
4.6.1	Was ist RichText?	148
4.6.2	Erstellen eines RichText-Items in einem Dokument	149
4.6.3	Zugriff auf ein bestehendes RichText-Item	149
4.6.4	Zugriff auf den Wert eines RichText-Items	149
4.6.5	Auslesen des Texts eines RichText-Items	152

4.6.6	Die Zeichenformatierung in einem RichText-Item	153
4.6.7	Die Absatzformatierung in einem RichText-Item	156
4.7	Datums- und Zeithandling	160
4.7.1	Datums- oder Zeitwerte.....	160
4.7.2	Datums- oder Zeitbereich	164
4.8	Eingebundene Objekte und Dateianhänge	166
4.8.1	Was sind eingebundene Objekte, verknüpfte Objekte oder Dateianhänge?	166
4.8.2	Eingebettete Objekte	167
4.8.3	Verknüpfte Objekte.....	169
4.8.4	Dateianhänge	170
4.9	Internationalisierung	171
4.9.1	Zahlen- und Währungsangaben.....	171
4.9.2	Datumsangaben	172
4.10	Die Protokollierung	173
4.10.1	Das Protokollmedium	173
4.10.2	Meldung protokollieren	176
4.10.3	Fehler protokollieren	178
4.10.4	Anzahl der protokollierten Meldungen und Fehler	179
4.11	Gliederung	179
4.11.1	Erstellen einer Gliederung	179
4.11.2	Zugriff auf eine bestehende Gliederung	180
4.11.3	Zugriff auf die Elemente der Gliederung	181
4.12	Namenshandling.....	182
4.12.1	Der aktuelle Anwender	182
4.12.2	Erzeugen einer neuen Instanz von Name	182
4.12.3	Der abgekürzte Name.....	182
4.12.4	Der Name in kanonischer Form	183
4.12.5	Der allgemeine Name	183
4.12.6	Hierarchische Namen.....	183
4.12.7	Die Sprache	184
4.13	Newsletter	185
4.13.1	Erstellen einer neuen Instanz der Klasse Newsletter	185
4.13.2	Formatierung des Newsletters.....	186
4.13.3	Erzeugen des Übersichtsdokuments	186
4.13.4	Auswahl eines Dokuments im Newsletter	187
5	JDBC	189
5.1	SQL.....	189
5.1.1	Abfrage	189
5.1.2	Einfügen	190
5.1.3	Änderung	190
5.1.4	Datenbankelemente erstellen.....	190

5.2	Aufbau einer Datenbankverbindung zur Datenabfrage mit JDBC	191
5.2.1	Schritt1 – JDBC-Treiberklassen für das Datenbanksystem laden ...	191
5.2.2	Schritt2 – Verbindung zum Datenbanksystem aufbauen.....	191
5.2.3	Schritt3 – Datenabfrage ausführen.....	192
5.2.4	Schritt4 – Datenänderung ausführen	193
5.2.5	Schritt5 – Auslesen des Abfrageergebnisses.....	194
5.2.6	Aufbau einer Datenbankverbindung zur Manipulation von Daten mit JDBC	194
5.3	JDBC-Treiber für Notes	195
6	HTML.....	197
6.1	Aufbau der Seite.....	197
6.2	Die Titelzeile der Seite	198
6.3	Überschriften	198
6.4	Hyperlinks	199
6.5	Formulare	199
7	Servlets.....	201
7.1	Was sind Servlets?	201
7.2	Wie funktioniert ein Servlet?	201
7.3	Zustandslose Servlets.....	202
7.4	Sitzung bzw. Session	202
7.5	Java Beans.....	203
7.6	Die Klasse HttpServletRequest	204
7.6.1	Die aktuelle Sitzung	204
7.6.2	Die Parameter.....	204
7.6.3	Cookies.....	205
7.6.4	Die Anfrageart	205
7.6.5	Die angeforderte Adresse.....	205
7.6.6	SessionID	205
7.7	Die Klasse HttpServletResponse.....	206
7.7.1	Der Ausgabestream	206
7.7.2	Cookies.....	206
7.7.3	Weiterleitung.....	206
7.7.4	http-Statuscode	206
7.8	Die Klasse HttpSession.....	208
7.8.1	Sessionspezifische Daten.....	208
7.8.2	Information über den Client	208
7.8.3	Informationen über den Server	209
7.8.4	Allgemeine Informationen über die Session.....	209
7.8.5	Zerstören einer Session	210
7.9	Die Klasse Cookie	210
7.10	Das erste Servlet.....	211
7.10.1	Schritt1 – Neue Klasse erstellen und vom HttpServlet ableiten.....	211

7.10.2	Schritt2 – Methode für GET oder POST schreiben	211
7.10.3	Schritt3 – Anfordern des Ausgabestreams	212
7.10.4	Schritt4 – Ausgabe des HTML-Codes	212
7.10.5	Schritt5 – Ermitteln der Adresse des Clients	213
7.10.6	Schritt6 – Parameter abfragen	213
7.10.7	Schritt7 – Lesen und Schreiben von Sessiondaten	214
7.10.8	Schritt8 – Verwendung von Cookies	215
8	JSP-Programmierung	217
8.1	Was sind JSPs?	217
8.2	Der Aufbau von JSP-Seiten.....	217
8.3	Was passiert, wenn eine JSP-Seite aufgerufen wird?	217
8.4	Die JSP-Syntax.....	217
8.4.1	Ausdrücke und Direktiven	217
8.4.2	Javacode	218
8.4.3	Kommentare	218
8.4.4	Deklarationen	219
8.5	Die verschiedenen JSP-Direktiven	219
8.5.1	Die Page-Direktive	219
8.5.2	Die Include-Direktive	221
8.5.3	Die Taglib-Direktive	221
8.6	Die verschiedenen JSP-Aktionen	221
8.6.1	Forwarding von Anfragen	221
8.6.2	Java Beans	222
8.7	Die Tag-Bibliothek von Notes	224
8.7.1	Installation der Tag-Bibliothek.....	224
8.8	Übersicht über die Tags der Bibliothek	225
9	Referenz der Notes Javaklassen.....	227
9.1	Klasse ACL.....	227
9.1.1	Konstanten	227
9.1.2	Eigenschaften	227
9.1.3	Methoden	229
9.2	Klasse ACLEntry	233
9.2.1	Konstanten	233
9.2.2	Eigenschaften	233
9.2.3	Methoden	242
9.3	Klasse Agent.....	243
9.3.1	Konstanten	243
9.3.2	Eigenschaften	244
9.3.3	Methoden	251
9.4	Klasse AgentContext	252
9.4.1	Eigenschaften	252
9.4.2	Methoden	257

9.5	Klasse Database	259
9.5.1	Konstanten	259
9.5.2	Eigenschaften	259
9.5.3	Methoden	273
9.6	Klasse DateRange	289
9.6.1	Eigenschaften	289
9.6.2	Methoden	291
9.7	Klasse DateTime	291
9.7.1	Eigenschaften	291
9.7.2	Methoden	294
9.8	Klasse DbDirectory	299
9.8.1	Konstanten	299
9.8.2	Eigenschaften	299
9.8.3	Methoden	300
9.9	Klasse Document	304
9.9.1	Eigenschaften	304
9.9.2	Methoden	318
9.10	Klasse DocumentCollection	333
9.10.1	Eigenschaften	333
9.10.2	Methoden	335
9.11	Klasse EmbeddedObject	341
9.11.1	Konstanten	341
9.11.2	Eigenschaften	341
9.11.3	Methoden	344
9.12	Klasse Form	346
9.12.1	Eigenschaften	346
9.12.2	Methoden	350
9.13	Klasse International	351
9.13.1	Eigenschaften	351
9.13.2	Methoden	360
9.14	Klasse Item	360
9.14.1	Konstanten	360
9.14.2	Eigenschaften	361
9.14.3	Methoden	369
9.15	Klasse Log	374
9.15.1	Konstanten	374
9.15.2	Eigenschaften	374
9.15.3	Methoden	377
9.16	Klasse MIMEEntity	381
9.16.1	Konstanten	381
9.16.2	Eigenschaften	381
9.16.3	Methoden	385

9.17	Klasse MIMEHeader	394
9.17.1	Eigenschaften	394
9.17.2	Methoden	394
9.18	Klasse Name.....	398
9.18.1	Eigenschaften	398
9.18.2	Methoden	408
9.19	Klasse Newsletter	408
9.19.1	Eigenschaften	408
9.19.2	Methoden	410
9.20	Klasse Outline	411
9.20.1	Eigenschaften	411
9.20.2	Methoden	412
9.21	Klasse OutlineEntry.....	419
9.21.1	Konstanten	419
9.21.2	Eigenschaften	419
9.21.3	Methoden	429
9.22	Klasse Registration.....	431
9.22.1	Konstanten	431
9.22.2	Eigenschaften	431
9.22.3	Methoden	437
9.23	Klasse Replication	444
9.23.1	Konstanten	444
9.23.2	Eigenschaften	445
9.23.3	Methoden	448
9.24	Klasse RichTextItem.....	449
9.24.1	Eigenschaften	449
9.24.2	Methoden	450
9.25	Klasse RichTextParagraphStyle.....	455
9.25.1	Konstanten	455
9.25.2	Eigenschaften	455
9.25.3	Methoden	459
9.26	Klasse RichTextStyle	461
9.26.1	Konstanten	461
9.26.2	Eigenschaften	462
9.26.3	Methoden	467
9.27	Klasse RichTextTab	467
9.27.1	Eigenschaften	467
9.27.2	Methoden	468
9.28	Klasse Session.....	468
9.28.1	Eigenschaften	468
9.28.2	Methoden	475
9.29	Klasse Stream	483
9.29.1	Konstanten	483

9.29.2	Eigenschaften	484
9.29.3	Methoden	486
9.30	Klasse View.....	490
9.30.1	Konstanten	490
9.30.2	Eigenschaften	490
9.30.3	Methoden	501
9.31	Klasse ViewColumn	512
9.31.1	Konstanten	512
9.31.2	Eigenschaften	513
9.31.3	Methoden	529
9.32	Klasse ViewEntry.....	529
9.32.1	Konstanten	529
9.32.2	Eigenschaften	529
9.32.3	Methoden	536
9.33	Klasse ViewEntryCollection	536
9.33.1	Eigenschaften	536
9.33.2	Methoden	537
9.34	Klasse ViewNavigator	543
9.34.1	Eigenschaften	543
9.34.2	Methoden	545
10	Referenz JSP	559
10.1	Referenz der JSP Domino Tags.....	559
10.1.1	action	559
10.1.2	attachmentlist.....	559
10.1.3	attachments	559
10.1.4	authors	559
10.1.5	checkbox	560
10.1.6	created	560
10.1.7	db	560
10.1.8	dbfill	561
10.1.9	dbselect.....	561
10.1.10	default.....	561
10.1.11	deleteattachment	562
10.1.12	deletedoc	562
10.1.13	docloop	562
10.1.14	docnavimg	562
10.1.15	document.....	563
10.1.16	editdoc	563
10.1.17	fill	563
10.1.18	form	563
10.1.19	formlink.....	564
10.1.20	formula.....	564

10.1.21	ftsearch	564
10.1.22	ifauthor	565
10.1.23	ifcapability	565
10.1.24	ifcategoryentry	565
10.1.25	ifconflicentry	565
10.1.26	ifdbrole	565
10.1.27	ifdepositor	566
10.1.28	ifdesigner	566
10.1.29	ifdisplaymode	566
10.1.30	ifdocauthor	566
10.1.31	ifdocreader	566
10.1.32	ifdocumententry	566
10.1.33	ifeditmode	567
10.1.34	ifeditor	567
10.1.35	ifmanager	567
10.1.36	ifnoaccess	567
10.1.37	ifreader	567
10.1.38	ifserverevent	567
10.1.39	iftotalentry	568
10.1.40	indent	568
10.1.41	input	568
10.1.42	item	568
10.1.43	lastaccessed	569
10.1.44	lastmodified	569
10.1.45	mailto	569
10.1.46	msg	569
10.1.47	newdoc	570
10.1.48	nextdoc	570
10.1.49	nodocument	570
10.1.50	novalues	570
10.1.51	option	570
10.1.52	page	571
10.1.53	pagebar	571
10.1.54	pagefirst	571
10.1.55	pagelast	571
10.1.56	pagenext	572
10.1.57	pageprev	572
10.1.58	parentlink	572
10.1.59	preserve	572
10.1.60	preset	572
10.1.61	prevdoc	573
10.1.62	profiledocument	573

10.1.63	radio.....	573
10.1.64	response.....	574
10.1.65	richtext.....	574
10.1.66	runagent	574
10.1.67	saveclosedoc.....	574
10.1.68	savedoc	575
10.1.69	savenow	575
10.1.70	select.....	575
10.1.71	selectedloop	575
10.1.72	selectentry	576
10.1.73	session.....	576
10.1.74	setitem	576
10.1.75	textarea	576
10.1.76	unid.....	577
10.1.77	user	577
10.1.78	validatecc	577
10.1.79	validatedate.....	577
10.1.80	validatelength.....	578
10.1.81	validatepattern	578
10.1.82	validaterange.....	578
10.1.83	validaterequired.....	579
10.1.84	validatesummary	579
10.1.85	validatewith	579
10.1.86	view	580
10.1.87	viewitem.....	580
10.1.88	viewloop.....	580
10.2	Referenz der JSP Domutil.....	581
10.2.1	browsertag	581
10.2.2	case.....	581
10.2.3	condition	581
10.2.4	default.....	581
10.2.5	else	581
10.2.6	elseif	582
10.2.7	format	582
10.2.8	if.....	582
10.2.9	nocache.....	582
10.2.10	switch.....	583
10.2.11	switchvalue.....	583
10.2.12	urlconverter	583
11	CGI-Variablen.....	585
	Listingverzeichnis	589
	Stichwortverzeichnis.....	593

Einleitung

Als Notes das Licht der Welt erblickte, waren die Programmiermöglichkeiten noch drastisch eingeschränkt, wurden aber durch die immer größere Ausbreitung von Lotus ständig weiterentwickelt. Einer der wichtigsten Schritte dabei dürfte die Einführung der Programmiersprache Java unter Notes gewesen sein. Denn dadurch wurde dem Notes Entwickler ein sehr mächtiges Werkzeug an die Hand gegeben. Es wurde die Welt der professionellen Softwareentwicklungen mit den Möglichkeiten von Notes verbunden.

Durch Java ist es z. B. möglich, bei der Notes Entwicklung alle Elemente der objekt-orientierten Programmierung zu verwenden. Außerdem stehen alle Möglichkeiten von Java, wie die Netzwerkprogrammierung durch Sockets und CORBA zur Verfügung. Auch ist es erst seit der Einführung von Java möglich, innerhalb von Notes Anwendungen mit Multithread-Architektur zu entwickeln.

Darüber hinaus wird mit der Java-Servlet-Technik eine Möglichkeit geboten, professionelle und leistungsfähige Weboberflächen zu realisieren. In der neusten Notes Version wird dies zusätzlich durch eine Taglib-Bibliothek für Applicationserver unterstützt. Diese Bibliothek stellt alle Elemente zur Verfügung, um auf einfachem Weg JSP-Seiten mit einer Notes Anbindung für einen Applicationserver zu entwickeln.

Besonders interessant für einen Entwickler dürfte auch sein, dass beliebige bestehende Java-Bibliotheken unter Notes verwendet werden können. Dadurch kann z. B. auf die große Zahl der schon existierenden kommerziellen oder freien Javabibliotheken zurückgegriffen werden. Des Weiteren kann ein eigener Pool an Bibliotheken geschaffen werden, welche dann nicht nur in den Notes Anwendungen Verwendung finden, sondern auch in anderen Javaanwendungen genutzt werden können.

Dieses Buch soll zeigen, wie die verschiedenen Techniken kombiniert und unter Notes eingesetzt werden können. Es führt dazu in die verschiedenen Techniken ein und zeigt weiterführende Möglichkeiten auf. Der Entwickler wird somit mit dem Potenzial der Kombination von Java und Notes ausführlich bekannt gemacht.

1

Die Programmiersprache Java

1.1 Der Unterschied zwischen einer Interpreter- und einer Compilersprache

Programmiersprachen lassen sich in zwei große Gruppen einteilen. Die eine Gruppe bilden die Compilersprachen. Bei Compilersprachen wird der Programmtext vor der Programmausführung in ein für den Computer verständliches Format übersetzt. Ein weit verbreitetes Beispiel dafür ist die Programmiersprache C. Wenn ein C-Programm geschrieben wird, wird dieses von einem Compiler in ein maschinenlesbares, ausführbares Programm übersetzt. Dieses erzeugte ausführbare Programm ist dann allerdings plattformspezifisch. Das bedeutet, es kann nur auf dem vorgesehenen Computersystem und unter dem vorgesehenen Betriebssystem verwendet werden. Die großen Vorteile von Compilersprachen sind die Performance und die geringe Größe der ausführbaren Programme.

Die zweite Gruppe ist die Gruppe der Interpretersprachen. Bei Interpretersprachen wird das Programm nicht durch einen Compiler übersetzt, sondern die Befehle werden erst bei der Programmausführung ausgewertet und analysiert. Dieses hat den besonderen Vorteil, dass ein Programm auf mehreren verschiedenen Plattformen und Computersystemen laufen kann. Allerdings wird dieses auf Kosten der Performance und der Größe der ausführbaren Programme erkauft.

Bei einer Javaanwendung werden beide Techniken verwendet. Zuerst wird das Java-Programm in einen so genannten Bytecode umgewandelt. Dieser Bytecode wird dann bei der Programmausführung interpretiert. Der Bytecode hat den besonderen Vorteil, dass er plattform- und systemunabhängig ist und eine bessere Performance bietet als reine Interpretersprachen.

1.2 Die Virtual Machine

Der Bytecode, also das ausführbare Java-Programm, wird in der Java Virtual Machine ausgeführt. Diese Virtual Machine ist ein Programm, welches für verschiedene Plattformen und Computersysteme verfügbar ist. Diese Virtual Machine sorgt dafür, dass eine einheitliche Umgebung für die Ausführung der Java-Programme vorliegt. Sie bildet die Schnittstelle zwischen dem Bytecode und dem Betriebssystem des Computers.

2 Einführung in die Javaprogrammierung

2.1 Die Elemente der Sprache Java

2.1.1 Der Aufbau eines Javaprogramms

Ein Javaprogramm besteht aus einer oder mehreren Klassen. Für diese Klassen werden Methoden und Eigenschaften in der Form von Prozeduren und Variablen definiert. Die Implementation erfolgt mit Javaanweisungen.

```
/** Beispielsklasse
class Beispielklasse {
    private int value;
    public Beispielklasse() {};
    public finalize() {};
    int main ( void ) {
        System.out.println ( „Hello World" );
    };
};
```

Listing 2.1: Beispiel Aufbau eines Javaprogramms

Folgende Regeln sind bei der Kodierung zu beachten:

- ▶ Jede Anweisung wird mit einem Semikolon abgeschlossen.
- ▶ Die Groß- und Kleinschreibung ist relevant.
- ▶ Mehrere Anweisungen können durch geschweifte Klammern gruppiert werden.

2.1.2 Kommentare

In der Programmiersprache Java gibt es zwei Möglichkeiten, Kommentare einzufügen. Eine Möglichkeit, Kommentare zu kennzeichnen, sind zwei Schrägstriche. Alles was den zwei Schrägstichen bis zum Zeilenende folgt, wird als Kommentar interpretiert. Wenn die zwei Schrägstriche am Zeilenanfang stehen, ist die gesamte Zeile eine Kommentarzeile.

```
/** Kommentarzeile
public class Test {
    Test() {}; // Ab hier steht Kommentar
};
```

Listing 2.2: Beispiel Kommentarzeilen

Die zweite Möglichkeit ist die Verwendung der Markierungen `/*` und `*/`. Mit diesen Markierungen ist es möglich, Kommentare über das Zeilenende hinaus zu schreiben. Der Text zwischen den beiden Markierungen gilt als Kommentar.

```
// Kommentarmarkierungen
/*
    Alles was
    zwischen den
    Markierungen
    steht ist Kommentar
*/

public class Test {
    Test() {}; /* Auch das ist Kommentar */
};
```

Listing 2.3: Beispiel Kommentare

2.1.3 Variablen und Datentypen

Wie in den meisten anderen Programmiersprachen werden auch in Java Variablen zur Speicherung von Daten eingesetzt. Jede Variable, die in einem Javaprogramm verwendet werden soll, muss zuvor deklariert werden. Bei der Deklaration wird der Variablen ein Datentyp zugewiesen.

Der Ort einer Variablendeklaration bestimmt den Gültigkeitsbereich einer Variablen. Bei einer Deklaration in einer Klassendefinition ist die Variable in der gesamten Klasse verfügbar. Wenn im Gegensatz dazu die Variable innerhalb eines Anweisungsblocks deklariert wird, so ist sie nur innerhalb dieses Anweisungsblocks verfügbar. Einfach ausgedrückt bedeutet dieses, wenn eine Variable innerhalb von zwei geschweiften Klammern deklariert wird, so ist sie auch nur innerhalb dieser geschweiften Klammern gültig.

Die Deklaration besteht aus Datentyp und dem Namen der Variablen. Bei einer Variablendeklaration können auch mehrere Variablen mit einer Anweisung deklariert werden. Dazu wird nur einmal der Datentyp angegeben und danach die Variablennamen durch Kommas getrennt aufgelistet.

Beispiel:

```
int a, b, c;
double d, e, f;
boolean var1, var2, var3;
```

Die einfachsten Datentypen in Java sind die Elementardatentypen. Diese sind einfache, überwiegend numerische Datentypen.

Elementardatentypen

Unter Java sind die folgenden Elementardatentypen verfügbar:

Name	Wertebereich
byte	vorzeichenbehaftete 8-Bit-Ganzzahl
short	vorzeichenbehaftete 16-Bit-Ganzzahl
int	vorzeichenbehaftete 32-Bit-Ganzzahl
long	vorzeichenbehaftete 64-Bit-Ganzzahl
float	vorzeichenbehaftete 32-Bit-Gleitkommazahl
double	vorzeichenbehaftete 64-Bit-Gleitkommazahl
char	16-Bit-Unicodezeichen
boolean	true oder false

Tabelle 2.1: Elementardatentypen in Java

```
int x;           // Variable x vom Datentyp int wird angelegt
double y,z;      // Variable y und z vom Datentyp double wird angelegt
```

Bei der Deklaration einer Variablen ist es möglich, ihr gleich einen Wert zuzuweisen.

```
int x = 3;
double y = 3.1;
int z = x+1;
```

Konstanten

Konstanten sind Variablen mit einem Wert, welcher während der Programmausführung nicht verändert wird. Diese Konstanten werden in Java mit den Schlüsselwörtern `static final` deklariert. Dadurch führt ein Versuch, den Wert dieser Konstanten zu ändern, zu einem Compilerfehler. Die beiden Schlüsselwörter werden bei der Deklaration der konstanten Variablen vor den Datentyp gesetzt.

```
static final int Pi = 3.148;
static final int max = 99999;
```

Arrays

Wie in den meisten anderen Hochsprachen gibt es auch in Java die Möglichkeit, Arrays zu nutzen. Arrays können unter anderem verglichen werden mit Tabellen, in denen in jeder Zelle ein Wert desselben Datentyps steht.

Jeder gültige Datentyp, elementar oder benutzerdefiniert, kann genutzt werden, um Arrays anzulegen. Dabei werden bei der Variablendeklaration an den Datentyp eine öffnende und eine schließende eckige Klammer angehängt.

```
int[] x;    // Eindimensionaler Array
double[][] y; // Zweidimensionaler Array
```

Um einen Array zu nutzen, reicht es nicht, die Variablen nur zu deklarieren. Der Array kann erst genutzt werden, wenn ihm mitgeteilt wurde, wie groß er ist und der entsprechende Speicherplatz zur Verfügung steht. Dieses erfolgt mit der Anweisung `new`.

Die Anweisung `new` wird immer dann genutzt, wenn eine komplexere Variable angelegt werden muss. Bei der `new`-Anweisung wird die Größe des Array angegeben.

```
int [] x;
int [][] y;
x = new int[3];
y = new int[3][3]
```

Danach kann auf die Elemente des Array zugegriffen werden. Dabei wird hinter dem Variablennamen angegeben, auf welche Position im Array zugegriffen werden soll. Bei der Angabe der Position ist zu beachten, dass bei der Nummerierung bei 0 begonnen wird. Ein Array mit 3 Elementen hat also die Position 0, 1 und 2.

```
int [] x;
int [][] y;
x = new int [3];
y = new int [3][3];
x[0] = 3;
y[1] = 4;
```

2.1.4 Garbage Collector

Jeder Speicherbereich, der von einem Programm genutzt wird, muss nach der Ausführung wieder freigegeben werden. In einigen Hochsprachen muss dieses im Programmtext berücksichtigt werden. Bei Java aber nimmt diese Aufgabe der Automatische Garbage Collector war. Der Garbage Collector ist ein periodisch im Hintergrund laufender Prozess, der nicht mehr benötigten Speicherbereich wieder freigibt. Somit entfällt in Java die Problematik für den Programmierer zu überlegen, welche Variablen nicht mehr benötigt werden.

2.1.5 Elementaroperatoren

Um den Inhalt von Variablen oder konstante Ausdrücke zu verknüpfen, können in Java eine Reihe von Operatoren verwendet werden.

Dieses sind die wichtigsten Operatoren von Java:

Operator	Beschreibung	Operator	Beschreibung
&&	Logisches Und	+=	Addition
	Logisches Oder	-=	Subtraktion
!	Logisches Not	++	Inkrementation
==	Gleich	--	Dekrementation
<=	Kleiner oder Gleich	*	Multiplikation
>=	Größer oder Gleich	/	Divison
!=	Ungleich		

Tabelle 2.2: Elementaroperatoren in Java

```
int x=3;
int y=3+3*3;
boolean z = ( true && false ) || !( false && true );
```

2.1.6 Verzweigungen

Die bedingte Programmausführung lässt sich unter Java auf zwei verschiedene Arten realisieren. Eine Anweisung ist `if then else` und die andere `switch case`.

Die if-then-else-Anweisung

Mit der `if-then-else`-Anweisung können Bedingungen angegeben werden.

```
if ( Bedingung1 ) {  
    // Anweisungsblock1  
} else if ( Bedingung2 ) {  
    // Anweisungsblock2  
...  
} else {  
    // Anweisungsblock3  
};
```

Wenn die `Bedingung1` erfüllt ist, wird der `Anweisungsblock1` ausgeführt und danach das Programm nach der `if`-Anweisung fortgeführt. Sollte die `Bedingung1` nicht erfüllt sein, so wird die `Bedingung2` überprüft. Der `Anweisungsblock2` wird ausgeführt, wenn die `Bedingung2` erfüllt ist. Sollte auch die `Bedingung2` nicht erfüllt sein, so wird der `Anweisungsblock3` ausgeführt, bevor die Programmausführung hinter der `if`-Anweisung fortgesetzt wird.

```
if ( a == 2 ) {  
    System.out.println ( "Wenn a gleich 2" );  
} else if ( b == 3 ) {  
    System.out.println ( "Wenn a ungleich 2 und b gleich 3" );  
} else {  
    System.out.println ( "Wenn a ungleich 2 und b ungleich 3" );  
};
```

Listing 2.4: Beispiel für die if-Anweisung

Die switch-case-Anweisung

Mit der `switch-case`-Anweisung kann eine Fallunterscheidung realisiert werden. Dabei wird anhand eines einfachen Ausdrucks und verschiedenen konstanten Werten unterschieden, mit welchen Anweisungen die Programmausführung fortgesetzt werden soll.

```
switch ( Variable ) {  
    case ( Konstante1 ):  
        // Anweisungsblock für Fall1  
        break;  
    case ( Konstante2 ):  
        // Anweisungsblock für Fall2  
        break;  
    ...  
    default:  
        // Anweisungsblock für Elsefall  
        break;  
};
```

Der Inhalt der angegebenen Variablen dient zur Entscheidung, welcher Anweisungsblock ausgeführt werden soll. Wenn die Variable den Wert Konstante1 beinhaltet, wird der Anweisungsblock1 ausgeführt. Der Anweisungsblock2 wird ausgeführt, wenn die angegebene Variable den Wert Konstante2 enthält. Sollte die Variable keinen der angegebenen konstanten Werte besitzen, wird der else-Anweisungsblock ausgeführt.

```
switch ( farbe ) {
    case ( 1 ):
        System.out.println ( "Farbe 1" );
        break;
    case ( 2 ):
        System.out.println ( "Farbe 2" );
        break;
    default:
        System.out.println ( "Unbekannte Farbe" );
        break;
};
```

Listing 2.5: Beispiel für die switch-Anweisung

2.1.7 Schleifen

In Java gibt es drei verschiedene Möglichkeiten, eine Pogrammschleife zu implementieren.

Die for-Anweisung

Bei der for-Schleife handelt es sich um eine kopfgesteuerte Schleife.

```
for ( Startanweisung; Bedingung; Schrittanweisung ) {
    // Anweisungsblock
}
```

Zuerst wird die Startanweisung ausgeführt, danach wird der Anweisungsblock durchlaufen, solange die Bedingung erfüllt ist. Nach jedem Durchlauf des Anweisungsblocks wird die Schrittanweisung ausgeführt.

```
for ( int stepvar = 1; stepvar<10; stepvar++ ) {
    System.out.println ( "Schritt Nr. " + stepvar );
};
```

Listing 2.6: Beispiel für die for-Anweisung

Die do-Schleife

Die do-Anweisung ist eine fußgesteuerte Pogrammschleife. Das bedeutet, dass zuerst der Anweisungsblock ausgeführt und danach die Bedingung überprüft wird, um zu entscheiden, ob der Anweisungsblock wiederholt werden soll. Solange die Bedingung nicht erfüllt ist, wird die Schleife wiederholt.

```
do {
    // Anweisungsblock
} while ( Bedingung )
```


Beispiel:

```
int a = 1;
do {
    System.out.println ( "Schritt Nr. " + a );
    a++;
} while ( a < 10 );
```

Listing 2.7: Beispiel für die do-Anweisung

Die while-Schleife

Bei der while-Schleife handelt es sich um eine kopfgesteuerte Schleife. Es wird die Bedingung überprüft und der Anweisungsblock so lange wiederholt, wie die Bedingung erfüllt ist.

```
while ( Bedingung ) {
    // Anweisungsblock
};
```

Beispiel:

```
int a = 1;
while ( a < 10 ) {
    System.out.println ( "Schritt Nr. " + a );
    a++;
};
```

Listing 2.8: Beispiel für die while-Anweisung

2.1.8 Packages verwenden

Auch in Java wird das Rad nicht immer wieder neu erfunden. Es gibt ebenso wie in anderen Programmiersprachen die Möglichkeit, Bibliotheken mit immer wieder benötigten Programmsegmenten anzulegen. Diese Bibliotheken heißen in Java Packages. Um diese Packages zu verwenden, muss das entsprechende Package in das eigene Programm eingebunden werden. Dieses Einbinden wird mit der import-Anweisung veranlasst.

Bei der Namensvergabe für Packages wird ein spezielles Konzept verfolgt. Der Name besteht aus mehreren Wörtern, welche durch Punkte getrennt werden. Ziel dieser Struktur ist es, eine hierarchische Struktur von Packages aufzubauen. Das erste Element im Packagenamen ist häufig identisch mit dem Domänennamen der Firma in umgekehrter Reihenfolge:

```
ibm.com => com.ibm
notes.net => net.notes
```

Nach der Bezeichnung der Firma folgen dann weitere beliebige Namen, welche möglichst das Package beschreiben oder identifizieren sollten:

```
com.ibm.logging
com.ibm.db2.jdbc
import com.ibm.logging.*;
import com.ibm.db2.jdbc.*;
import com.ibm.logging.Logger;
```

Ein Stern am Ende der `import`-Anweisung zeigt an, dass alle Klassen dieses Packages importiert werden sollen. Steht statt des Sterns am Ende ein Klassenname, wird nur diese Klasse importiert.

Bevor sich jetzt jemand aber auf das Entwickeln von riesigen Javabibliotheken stürzt, sollte er einen Blick auf die bei Java schon vorhandenen Bibliotheken werfen. Es gibt eine riesige Anzahl von Standardpackages, in denen schon viele Aufgabenstellungen realisiert wurden. Die Namen dieser Standardpackages fangen häufig mit „Java“ an.

Package	Inhalt
<code>java.util</code>	Container (Listen, Vektor etc.)
<code>java.io</code>	Ein- und Ausgabe-Routinen
<code>java.net</code>	Netzwerksteuerung

Tabelle 2.3: Beispiel von Standard-Packages

Wenn ein Package mit der `import`-Anweisung eingebunden werden soll, so wird die entsprechende Datei mit dem Bytecode des Packages im so genannten Classpath gesucht. Der Classpath ist der Suchpfad für Packages. Dabei ist er eine Umgebungsvariable wie auch die bekannte `PATH`.

2.1.9 Zeichenketten

Um Zeichenketten in Java zu nutzen, steht im Standardpackage *java.lang* die Klasse *String* zur Verfügung. Diese Klasse kann wie ein Elementardatentyp für die Deklaration von Variablen verwendet werden. Das Package muss in das aktuelle Programm mit der `import`-Anweisung eingebunden werden.

```
String kette = "Hallo";
```

Wie im Beispiel zu sehen ist, werden konstante Zeichenketten in Java in doppelten Hochkommas eingeschlossen. Auch für den Datentyp `String` kann ein Großteil der Standardoperatoren verwendet werden.

```
import java.lang.*;

// ...

String kettel = "Hallo";
String kette2 = "Welt";
String kette3 = kettel + " " + kette2;
if ( kettel == kette2 ) {
    kette3 = kettel;
} else {
    kette3 = kette2;
}
```

Listing 2.9: Beispiel einer Zeichenkette

2.1.10 Nicht druckbare Sonderzeichen

Nicht druckbare Sonderzeichen werden in Java mit Escape-Sequenzen dargestellt.

Zeichen	Bedeutung
\n	Zeilenwechsel
\t	Tabulator
\r	Wagenrücklauf
\'	'
\"	"
\\	\

Tabelle 2.4: Nicht druckbare Sonderzeichen

```
String kette = "\"Hallo Welt\"";    // Inhalt von kette ist 'Hallo Welt'
```

2.2 Objektorientierte Softwareentwicklung

2.2.1 Objekte und Objektinstanzen

Java ist eine objektorientierte Programmiersprache. Bei der objektorientierten Programmierung wird versucht, die an der Problemstellung beteiligten Objekte zu definieren. Ein Objekt ist ein realer oder imaginärer Gegenstand. Für diesen Gegenstand bzw. das Objekt werden die Eigenschaften und das Verhalten, auch Methoden genannt, definiert. Ein Objekt wird in Java mit Hilfe von Klassen definiert.

```
class Klassenname {  
    Variablentyp1 Variablenname1;  
    Variablentyp2 Variablenname2;  
    ...  
    VariablentypA Methodenname ( Parameterliste ) {  
        // MethodenanweisungA  
    };  
    VariablentypB Methodenname ( Parameterliste ) {  
        // MethodenanweisungB  
    };  
};
```

Mit der Hilfe der Klassendefinition werden Objektinstanzen erzeugt, welche je einen einzelnen identifizierbaren Gegenstand dieser Art darstellen. Dabei hängt die Abgrenzung der Klassen und Objektinstanzen immer von der Art und dem Umfang der betrachteten Problemstellung ab.

Beispiel:

Klasse bzw. Objekt	Objektinstanz
Auto Eigenschaften: Marke, Farbe, Türen Methoden: fahren, starten, stoppen <pre>public class Auto { public String marke; public String farbe; public int tueren; public void fahren(); public void starten(); public void stoppen(); };</pre>	Auto1: VW Rot 5-Türer Auto2: Audi Blau 4-Türer Auto3: BMW Grün 3-Türer
Farbe Eigenschaft: Name, RGB-Wert Methoden: anzeigen <pre>public class Farbe { public String name; public int[] rgbwert; public void anzeigen(); };</pre>	Farbe1: Rot 255,0,0 Farbe2: Grün 0,255,0 Farbe3: Blau 0,0,255
Kunde Eigenschaften: Vorname, Nachname Methoden: Ausgabe <pre>public class Kunde { public String vorname; public String nachname; public void ausgabe(); };</pre>	Kunde1: Hans Meier Kunde2: Fitz Schulz Kunde3: Frank Wagner

Tabelle 2.5: Beispiele für Klassen bzw. Objekte und Instanzen

2.2.2 Verwendung von Klassen

Eine Klasse kann in den meisten Fällen in Java wie ein Datentyp verwendet werden. Das bedeutet, es können Variablen von diesem Typ genutzt werden.

```
Auto autovar;
Farbe zeichenfarbe;
```

Erzeugen einer Instanz einer Klasse

Anders als bei den Elementardatentypen müssen, wenn eine Variable mit Datentypen einer Klasse genutzt werden soll, einige zusätzliche Maßnahmen ergriffen werden. Eine Objektinstanz wird dabei z. B. in einen definierten Initialzustand versetzt.

Das Verhalten bei der Initialisierung einer Instanz wird durch den Konstruktor der Klasse beschrieben. Der Konstruktor ist eine spezielle Methode der Klasse. Der Methodenname des Konstruktors entspricht dem Klassennamen. Außerdem besitzt die Konstruktormethode keinen Rückgabewert.

```
public class Auto {  
  
    public Auto () {  
        marke = "Unbekannt";  
        farbe = "Unbekannt";  
        tueren = 0;  
    };  
  
    public String marke;  
    public String farbe;  
    public int tueren;  
    public void fahren(){};  
    public void starten(){};  
    public void stoppen(){};  
  
};  
// Verwendung der Klasse:  
Auto autovar;  
Auto autovar();  
Auto autovar = new Auto();
```

Listing 2.10: Beispiel für eine Klasse

Wenn die Konstruktormethode keine Parameter besitzt, spricht man von dem Defaultkonstruktor. Aber es ist durchaus möglich und häufig auch angebracht, dass ein Konstruktor Parameter erwartet. Auch ist es möglich, verschiedene Konstruktormethoden mit unterschiedlichen Parametern zu definieren.

Welche Konstruktorvariante verwendet wird, hängt von den angegebenen Parametern ab.

```
public class Auto {  
  
    public Auto () {  
        marke = "Unbekannt";  
        farbe = "Unbekannt";  
        tueren = 0;  
    };  
  
    public Auto ( String pmarke, String pfarbe ) {  
        marke = pmarke;  
        farbe = pfarbe;  
        tueren = 0;  
    };  
  
    public Auto ( String pmarke, String pfarbe, int ptuere ) {  
        marke = pmarke;  
        farbe = pfarbe;  
        tueren = ptuere;  
    };  
  
};
```

```

public String marke;
public String farbe;
public int tueren;
public void fahren();
public void starten();
public void stoppen();

};
// Verwendung der Klasse
Auto autovar ( "VW", "Rot" );
Auto autovar ( "Audi", "Blau", 3 );

```

Listing 2.11: Beispiel Konstruktoraufruf

Entfernen einer Instanz

Wie beim Erzeugen einer Instanz, so müssen auch beim Entfernen einer Instanz unter Umständen besondere Anweisungen ausgeführt werden. Zu diesem Zweck kann eine Destruktormethode definiert werden. Diese Methode wird immer dann ausgeführt, wenn die Instanz vom System aus dem Speicher entfernt wird. Der Zeitpunkt des Aufrufs des Destruktors kann nicht direkt beeinflusst werden. Den Garbage Collector der Virtual Machine steuert den Aufruf.

2.2.3 Kapselung

Ein wichtiges Merkmal der objektorientierten Programmierung ist die Kapselung. Das Prinzip der Kapselung ist, dass derjenige, der eine Klasse verwendet, nichts über die interne Realisierung der Eigenschaften und Methoden wissen muss. Dabei unterscheidet man die so genannte Definition und Implementation der Klasse. Als Definition werden die Namen der Eigenschaften bzw. die Methoden verstanden sowie die Parameter, die beim Parameterruf übergeben werden müssen. Die Implementation ist der Programmcode in den Methoden und Eigenschaften.

```

public class Color {

    private boolean rot, gruen, blau;

    public setColor(String pcolor ) {
        if ( pcolor == "rot" ) {
            rot = true; gruen=false; blau=false;
        } else if ( pcolor == "gruen" ) {
            rot = false; gruen=true; blau=false;
        } else if ( pcolor == "blau" ) {
            rot = false; gruen=false; blau=true;
        } else {
            rot = false; gruen=false; blau=false;
        }
    };

};

String getColor() {
    if ( rot ) {
        return "rot";
    }
}

```

```
        } else if ( gruen ) {  
            return "gruen";  
        } else if ( blau ) {  
            return "blau";  
        } else  
            return "unbekannt";  
    };  
};
```

Listing 2.12: Beispiel Kapselung

2.2.4 Sichtbarkeit von Methoden und Eigenschaften

Um die Kapselung zu gewährleisten, bietet Java die Möglichkeit, den Zugriff auf die Eigenschaften und Methoden der Klassen zu steuern. Dazu werden die Schlüsselwörter `private`, `protected` und `public` verwendet. Diese Schlüsselwörter werden bei der Klassendeklaration verwendet.

Schlüsselwort `public`

Auf die mit dem Schlüsselwort `public` markierten Methoden und Variablen kann über die Objektinstanz beliebig zugegriffen werden. Die so deklarierten Variablen können beliebig ausgelesen oder gesetzt und die Methoden aufgerufen werden.

Schlüsselwort `private`

Auf die mit dem Schlüsselwort `private` deklarierten Methoden oder Variablen kann nicht über die Objektinstanz zugegriffen werden. Dies ist nur innerhalb von Methoden der entsprechenden Klasse möglich, d. h. nicht von „Außen“. Auch die von der Klasse abgeleiteten Klassen können nicht auf diese Methoden oder Variablen zugreifen.

Schlüsselwort `protected`

Wenn eine Methode oder Variable als `protected` deklariert wurde, so ist wie bei `private` kein Zugriff von Außen möglich. Der Unterschied zu dem Schlüsselwort `private` ist allerdings, dass die abgeleiteten Klassen auf diese Elemente zugreifen können.

```
class Basisklasse {  
  
    // Nur innerhalb der Klasse sichtbar  
    private int a,b;  
  
    // Sichtbar in abgeleitete Klassen  
    protected int getB() { return b; };  
  
    // Außerhalb Sichtbar  
    public int getA() { return a; };  
};
```

Listing 2.13: Beispiel Sichtbarkeit

2.2.5 Vererbung

Eine weitere Technik der objektorientierten Entwicklung ist die Vererbung von Methoden und Variablen von einer Klasse auf eine andere. Ähnlich wie bei der biologischen Vererbung kann in Java eine Klasse eine Elternklasse besitzen. Eine Klasse kann dadurch bestimmte Methoden oder Variablen seiner Elternklasse übernehmen.

```
Fortbewegungsmittel
|---Fahrzeug
|---Schiff
|---Flugzeug

class Fortbewegung {
    ...
};

class Fahrzeug extends Fortbewegung {
    ...
};

class Schiff extends Fortbewegung {
    ...
};

class Flugzeug extends Fortbewegung {
    ...
};
```

Listing 2.14: Beispiel Vererbung

2.2.6 Überladung von Methoden

Eine für Einsteiger manchmal gewöhnungsbedürftige Technik ist das Überladen von Methoden. Bei der Überladung wird eine Methode einer Klasse in ihren abgeleiteten Klassen überschrieben. Somit gibt es eine Methode in der abgeleiteten Klasse unter demselben Namen wie in der übergeordneten Klasse, allerdings mit einer anderen Implementation, also einem anderen Programmcode im Inneren. Dies sind dann so genannte virtuelle Methoden und müssen mit dem Schlüsselwort `virtual` gekennzeichnet werden.

```
public class Grafik {
    public virtual void zeichnen {};
};

public class Kreis extends Grafik {

    private int x,y,r;

    virtual public void zeichnen {
        // Zeichne Kreis mit Mittelpunkt x,y und Radius r
    };

};
```



```
public class Linie extends Grafik {  
  
    private int x1, y1, x2, y2;  
  
    virtual public void zeichne {  
        // Zeichne Linie mit den Punkten x1,y1 und x2,y2  
    };  
  
};  
  
public class Testprogramm {  
  
    public void main ( void ) {  
        Grafik g1 = new Kreis( 10,10,100 );  
        Grafik g2 = new Linie ( 10,10,100,100);  
        g1.zeichne();  
        g2.zeichne();  
    };  
  
};
```

Listing 2.15: Beispiel Überladung

2.2.7 Abstrakte Klassen und Methoden

Es ist in Java zwar möglich, Methoden in einer Klasse zu deklarieren, jedoch nicht zu implementieren. Diese Methoden werden dann als abstrakt bezeichnet. Es wird damit in Java abgebildet, dass eine Klasse eine Methode besitzt, aber nicht, wie sie implementiert wird. Diese Technik macht z. B. dann Sinn, wenn eine Elternklasse zwar eine Methode anbieten soll, aber erst in ihren abgeleiteten Klassen feststeht, was bei dem Aufruf der Methoden ausgeführt werden soll. Eine solche Methode wird mit dem Schlüsselwort `abstract` gekennzeichnet.

```
public abstract class Fahrzeug {  
  
    private int a,b;  
    public abstract void ausgabe ();  
  
};
```

Listing 2.16: Beispiel Abstrakte Klassen und Methoden

Wenn eine Klasse abstrakte Methoden beinhaltet, muss sie als abstrakt gekennzeichnet werden. Eine abstrakte Klasse kann niemals dazu genutzt werden, eine Instanz der Klasse anzulegen. Solche Klassen dienen nur dazu, Methoden und Variablen zu deklarieren, die von den abgeleiteten Klassen gemeinsam genutzt werden sollen.

2.2.8 Statische Methoden und Variablen

Wir haben bisher Methoden und Variablen kennen gelernt, auf die über eine Objektinstanz zugegriffen wurde. Das bedeutet, dass jedes Mal, wenn ein Zugriff erfolgen soll, zuerst eine Instanz erzeugt werden muss.

Statische Variablen

Bei den bisher gezeigten Variablen der Klasse wurde in jeder Objektinstanz Speicherplatz für den Wert der Variablen reserviert. In jeder Instanz kann die Variable einen anderen Wert besitzen. Bei statischen Variablen ist dies nicht so. Für eine statische Variable wird pro Klasse nur einmal Speicherplatz reserviert und alle Instanzen greifen auf denselben Wert zu.

```
public class Fahrzeug {  
  
    private int a,b;  
    private static count;  
  
    public Fahrzeug() {  
        count++;  
    };  
  
    public void finalize() {  
        super.finalize();  
        count--;  
    };  
  
};
```

Listing 2.17: Beispiel Statische Variablen

Statische Methoden

Bei den statischen Methoden geht es nicht um den Speicherplatz, der reserviert wird. Statische Methoden werden dann eingesetzt, wenn Methoden zur Verfügung gestellt werden sollen, die sich nicht auf eine bestimmte Instanz der Klasse beziehen. Eine häufige Anwendung von statischen Methoden sind Methoden, die statische Variablen setzen oder auslesen.

```
public class Fahrzeug {  
  
    private int a,b;  
    private static count;  
  
    public Fahrzeug() {  
        count++;  
    };  
  
    public void finalize() {  
        super.finalize();  
        count--;  
    };  
  
    static int getCount() {  
        return count;  
    };  
  
};
```

Listing 2.18: Beispiel Statische Methoden

2.2.9 Das Schlüsselwort final

Mit dem Schlüsselwort `final` bei einer Methode kann verhindert werden, dass diese Methode in einer abgeleiteten Klasse überschrieben wird. Wenn dies trotzdem versucht wird, meldet der Javacompiler einen Fehler.

```
public class Fahrzeug {  
    public int a;  
  
    public final void setA(int pa) {  
        a = pa;  
    };  
  
    public final int getA() {  
        return a;  
    };  
};  
  
class Auto extends Fahrzeug {  
    public final void setA(int pa) {  
        a = pa;  
    }; // !! Compilerfehler  
  
    public final int getA() {  
        return a;  
    }; // !! Compilerfehler  
};
```

Listing 2.19: Beispiel Schlüsselwort final

2.2.10 Casting

Unter Casting wird verstanden, wenn der Datentyp eines Ausdrucks geändert werden soll. Teilweise funktioniert das Casting unter Java automatisch bei Wertzuweisungen.

```
int x = 3;  
double y = x;
```

Teilweise kann es aber auch vorkommen, dass ein Casting implizit angegeben werden muss. So etwas kann vorkommen, wenn mit mehreren voneinander abgeleiteten Klassen gearbeitet wird. Das Casten eines Ausdrucks wird durch den neuen Datentyp in Klammern gekennzeichnet.

```
int x =(int)3.14;
```

2.2.11 Das Interface

Im Unterschied zu vielen anderen objektorientierten Sprachen unterstützt Java keine Mehrfachvererbung. Von Mehrfachvererbung wird gesprochen, wenn eine Klasse von mehreren anderen Klassen abgeleitet wird. Das Fehlen dieser recht umstrittenen Technik wird in Java mit der Verwendung von Schnittstellen (Interface) ausgeglichen.

Eine Schnittstelle kann mit einer abstrakten Klasse, die nur einen Deklarationsteil mit Methoden und keinen Variablen besitzt, verglichen werden. Es ist zwar nur möglich, eine Klasse von einer anderen Klasse abzuleiten, jedoch kann eine Klasse eine beliebige Anzahl von Schnittstellen implementieren. Wenn eine Klasse eine Schnittstelle implementiert, so bedeutet dies, dass die Klasse alle in der Schnittstelle deklarierten Methoden implementiert hat.

Um zu kennzeichnen, dass eine Klasse eine bestimmte Schnittstelle implementiert, wird das Schlüsselwort `implements` in der Klassendeklaration verwendet.

```
public interface Fahrzeug {  
  
    public void fahren();  
  
};  
  
class Auto implements Fahrzeug {  
  
    public Auto() {  
    };  
  
    public void fahren() {  
        // Anweisungen  
    };  
  
};
```

Listing 2.20: Beispiel Schnittstelle

2.2.12 Ausnahmebehandlung

Kein Programm ist fehlerfrei. Daher sollte jedes Programm damit rechnen, dass Fehler auftreten können. Um Fehler bei der Programmausführung kontrolliert behandeln zu können, wird in Java das Exceptionhandling eingesetzt.

Bei dem Exceptionhandling werden Fehler durch das Werfen von Exceptions angezeigt. Diese geworfenen Exceptions werden dann an entsprechender Stelle im Programm abgefangen und ausgewertet. Als Exception können beliebige Klassen, die von der Klasse *Exception* abgeleitet sind, geworfen werden.

Zum Fangen einer Exception werden die `try`- und `catch`-Anweisungen verwendet. Dazu wird vor dem Anweisungsblock, in dem eine Exception geworfen werden könnte, der Befehl `try` geschrieben. Dann folgt in geschweiften Klammern der Anweisungsblock. Nach dem Anweisungsblock stehen dann eine oder mehrere `catch`-Anweisungen. Die `catch`-Anweisung wird von dem Schlüsselwort `catch` eingeleitet, dann folgt in runden Klammern der Klassen- sowie ein beliebiger Variablenname. Danach kommt der Anweisungsblock, der beschreibt, was beim Auftreten der Exception passieren soll.

```
try {  
    // Anweisungsblock  
} catch ( IOException e ) {  
    // Anweisungsblock der bei einer  
    // IOException ausgeführt wird  
}  
} catch ( Exception e ) {  
    // Anweisungsblock der bei einer  
    // Exception ausgeführt wird  
}  
try {  
    int x = 3;  
    if ( x == 0 )  
        throw new Exception();  
} catch ( Exception e ) {  
    System.out.println ( "Ein Fehler ist aufgetreten" );  
};
```

Listing 2.21: Beispiel Ausnahme

Das Finale mit finally

Es gibt in vielen Situationen Anweisungen, die immer ausgeführt werden sollen, auch wenn innerhalb des Anweisungsblocks ein Fehler auftritt. Diese können z. B. Aufräumarbeiten wie das Schließen von Dateien sein. Um die Ausführung dieser Anweisungen garantieren zu können, gibt es den Zusatz `finally`.

Die `finally`-Klausel wird hinter die `catch`-Anweisungen angehängt. Alle Anweisungen in dem Anweisungsblock hinter `finally` werden immer ausgeführt.

```
try {  
    // Anweisungsblock  
} catch ( IOException e ) {  
    // Anweisungsblock der bei einer  
    // IOException ausgeführt wird  
}  
} catch ( Exception e ) {  
    // Anweisungsblock der bei einer  
    // Exception ausgeführt wird  
}
```

```
} finally {  
    // Anweisungsblock der immer  
    // ausgeführt wird  
}  
  
try {  
    // Datei oeffnen  
    // Anweisungen  
} catch ( Exception e ) {  
    // Fehlerbehandlung  
}  
finally {  
    // Datei schliessen  
}
```

Listing 2.22: Beispiel finally

Werfen von Exceptions

Bis jetzt wurde nur gezeigt, wie man Ausnahmen fängt. Aber da man nur das fangen kann, was auch geworfen wurde, soll jetzt gezeigt werden, wie Ausnahmen geworfen werden. Dies kann z. B. notwendig sein, wenn bestimmte Zustände eintreten. Das Werfen erfolgt dann mit der Anweisung `throw`.

```
try {  
    if ( param1 == null ) {  
        throw new Exception  
            ( "param1 is null" );  
    };  
} catch ( Exception e ) {  
    e.printStackTrace();  
};
```

Listing 2.23: Beispiel für das Werfen einer Exception

In diesem Beispiel wird eine Exception geworfen, wenn in der Variablen *param1* der Wert null steht.

Unbehandelte Exceptions

Sollte eine Exception, die geworfen wird, nicht durch die `try-catch`-Anweisung abgefangen werden, so wird die momentan bearbeitete Methode sofort abgebrochen und die Exception an die aufrufende Methode weitergegeben.

Die Exception wird so immer eine Ebene weiter nach oben geworfen. Wenn auch die letzte Methode in der Anwendung diese Exception nicht aufhält, so wird sie bis an die JVM durchgereicht und führt dort zu einem Programmabbruch mit entsprechender Meldung.

3 Java Entwicklungs- umgebungen

Die Anzahl der möglichen Entwicklungsumgebungen für Java ist recht groß. Dabei reicht die Palette von kommandozeilenorientiert bis zu grafischen Oberflächen mit Workgroup- und Versionierungsunterstützung.

3.1 JDK

Die einfachste und preiswerteste Entwicklungsumgebung wird von der Firma Sun herausgegeben. Das offizielle JDK (Java Development Kit) wird kostenlos zum Herunterladen auf der offiziellen Java Webseite (java.sun.com) angeboten.

Diese Entwicklungsumgebung ist kommandozeilenorientiert, besitzt aber alle notwendigen Anwendungen, um ein Javaprogramm zu entwickeln. Um eine Javaanwendung mit Zugriff auf das Notes System zu entwickeln, müssen die Notes Jar-Dateien im Klassenpfad aufgenommen werden.

3.2 Visual Age

Die am weitesten verbreitete Entwicklungsumgebung ist Visual Age for Java. Diese Entwicklungsumgebung wird von IBM vertrieben. Sie bietet viele Hilfsmittel für die professionelle Anwendungsentwicklung, besonders für größere Entwicklerteams.

Um Notes Anwendungen in Visual Age zu entwickeln, müssen die entsprechenden Jar-Dateien in der Entwicklungsumgebung bekannt gemacht werden. Dafür gibt es zwei verschiedene Möglichkeiten.

3.2.1 Erstellen eines Notes API Projekts

In Visual Age werden die verschiedenen Packages zu Projekten gruppiert. Um die API von Notes zu verwenden, kann mit Packages aus der *notes.jar*-Datei ein entsprechendes Projekt angelegt werden. Dieses Projekt muss dann nur bei den Projekten eingebunden werden, in denen eine Notes Anwendung erstellt werden soll.

3.2.2 Aufnehmen des Jar Files im Classpath von Age

Eine andere Alternative ist es, die Datei *notes.jar* im Klassenpfad der Visual-Age-Umgebung einzubinden. Dazu muss nur die entsprechende Einstellung in den Optionen von Visual Age vorgenommen werden.

4 Javaprogrammierung unter Notes

4.1 Einsatzmöglichkeiten von Java unter Notes

Es gibt verschiedene Möglichkeiten, die Sprache Java bei der Anwendungsentwicklung unter Notes einzusetzen. Dabei unterscheiden sich die für den Zugriff auf Datenbanken oder Dokumenten eingesetzten Klassen nicht. Der einzige Unterschied ist der Weg, der genutzt wird, um eine Verbindung zum Notes Server im Programm aufzubauen.

4.1.1 Agentprogrammierung

Die einfachste Variante ist die Verwendung der Sprache Java in der Agentprogrammierung. Ein Agent kann in Notes entweder mit einfachen Aktionen, mit der Formelsprache, mit LotusScript oder mit Java realisiert werden.

Wenn die Wahl auf die Sprache Java fällt, kann der Entwickler entscheiden ob er das Javaprogramm innerhalb des Javaclients entwickeln will, oder ob er fertige Javaklassen in den Notesclient importieren will.

Programmcode innerhalb des Notes Clients

Um einen Agent mit der Programmiersprache Java innerhalb des Clients zu entwickeln, muss im Lotus Domino Designer unter Gemeinsamer Code die Rubrik AGENT ausgewählt werden. Dadurch wird in der rechten Bildschirmhälfte eine Liste der bereits vorhandenen Agents angezeigt. Außerdem befinden sich im oberen Bereich der rechten Bildschirmhälfte die Schaltflächen *Neuer Agent*, *Aktivieren*, *Deaktivieren* und *Signieren*.

An dieser Stelle muss die Schaltfläche *Neuer Agent* gewählt werden. Durch die Betätigung dieser Schaltfläche wird ein neuer Agent erstellt und das Eigenschaftsfenster für den neuen Agent wird angezeigt.

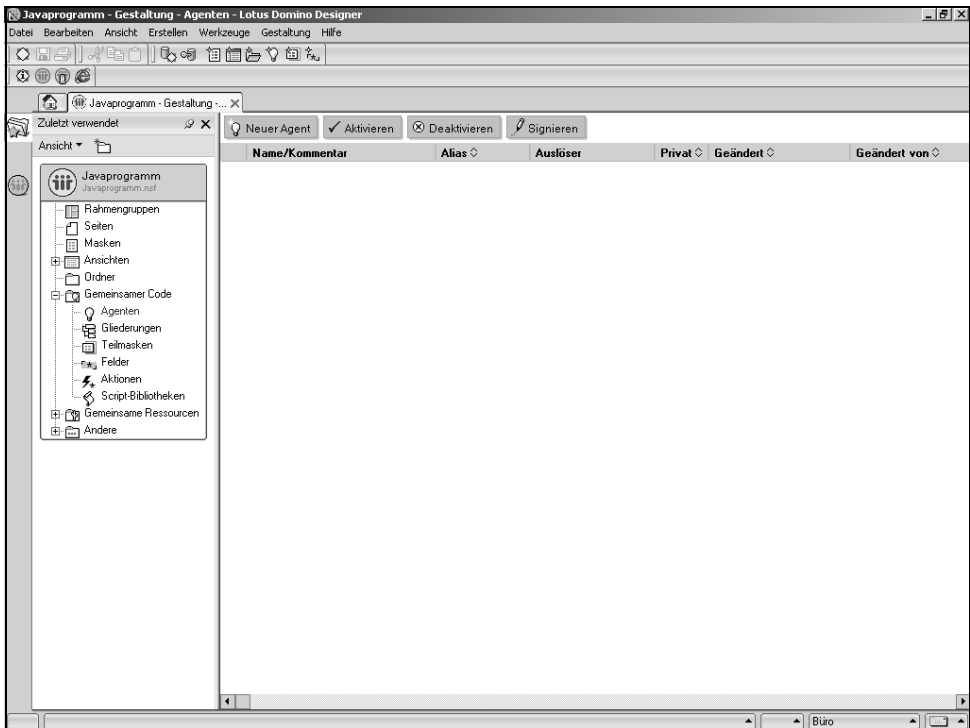


Abbildung 4.1: Agentübersicht der Datenbank



Abbildung 4.2: Agenteeigenschaften

In diesem Eigenschaftsfenster kann nun der Name des Agents eingegeben werden. Unter dem Eingabefeld für den Namen und dem Kommentar befindet sich die Auswahlmöglichkeit für einen gemeinsamen bzw. öffentlichen oder privaten Agent. Nachdem die gewünschten Eingaben in diesem Fenster erfolgt sind, kann es geschlossen werden.

Im rechten oberen Bereich der Bildschirmseite kann nun die Auswahl der Sprache für den Agent getroffen werden.

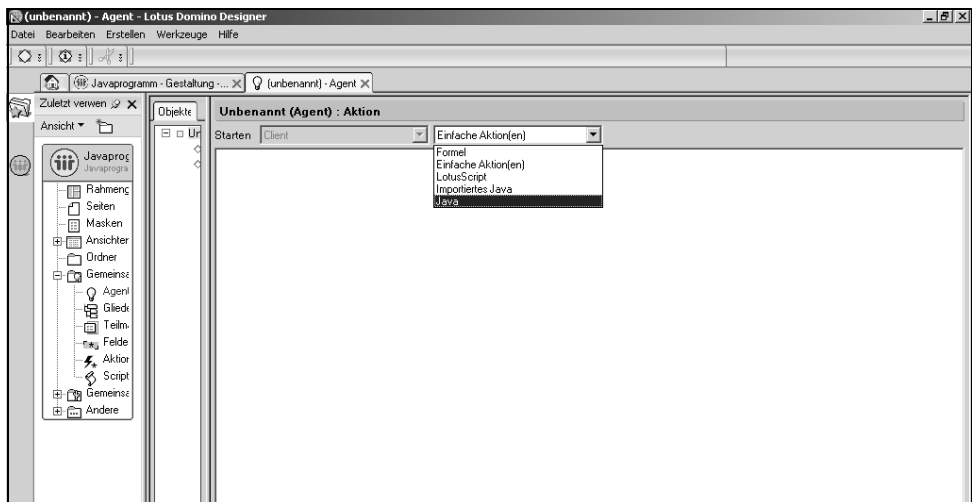


Abbildung 4.3: Agentsprachwahl

Zur Auswahl stehen hier *Formel*, *Einfache Aktionen*, *LotusScript*, *Importiertes Java* und *Java*. Um einen Java-Agent innerhalb des Clients zu entwickeln, muss hier die Option *Java* gewählt werden. Nach dieser Wahl wird ein Javagrundgerüst für den Agent erzeugt und angezeigt.

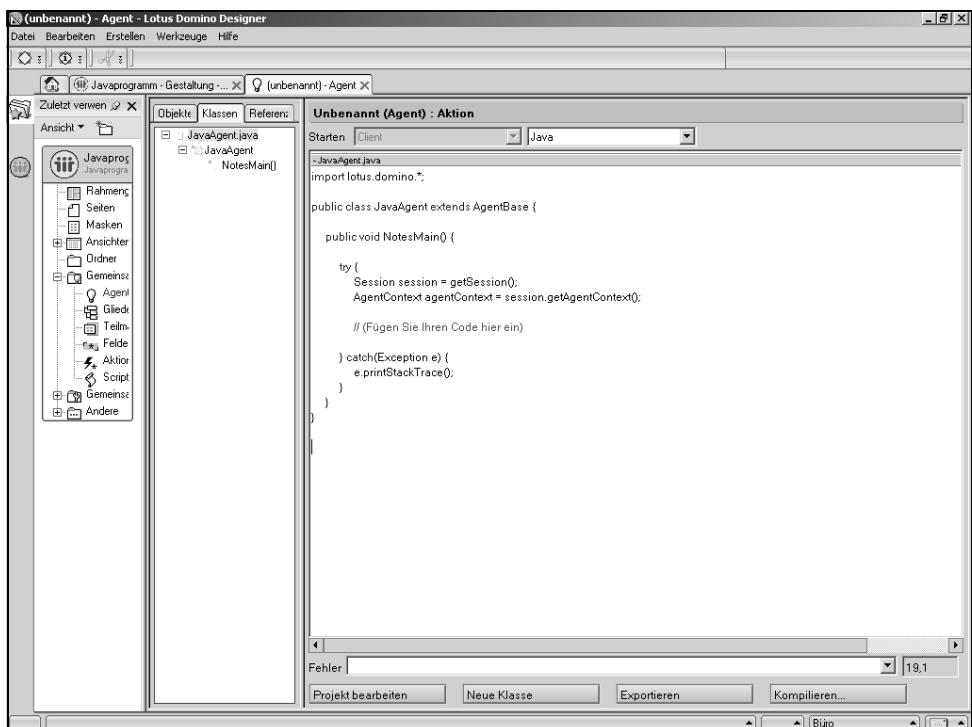


Abbildung 4.4: Neuer Java-Agent

Jetzt kann an der im Code markierten Stelle mit der Implementation des Programms begonnen werden. Standardmäßig wird eine Klasse mit dem Namen *JavaAgent* angelegt. Wenn weitere Klassen verwendet werden sollen, wird dieses durch die Betätigung der Schaltfläche *Neue Klasse* im unteren Bildschirmbereich dem Notes Client mitgeteilt. Durch die Betätigung dieser Schaltfläche wird auch für diese neue Klasse ein Grundgerüst erstellt.

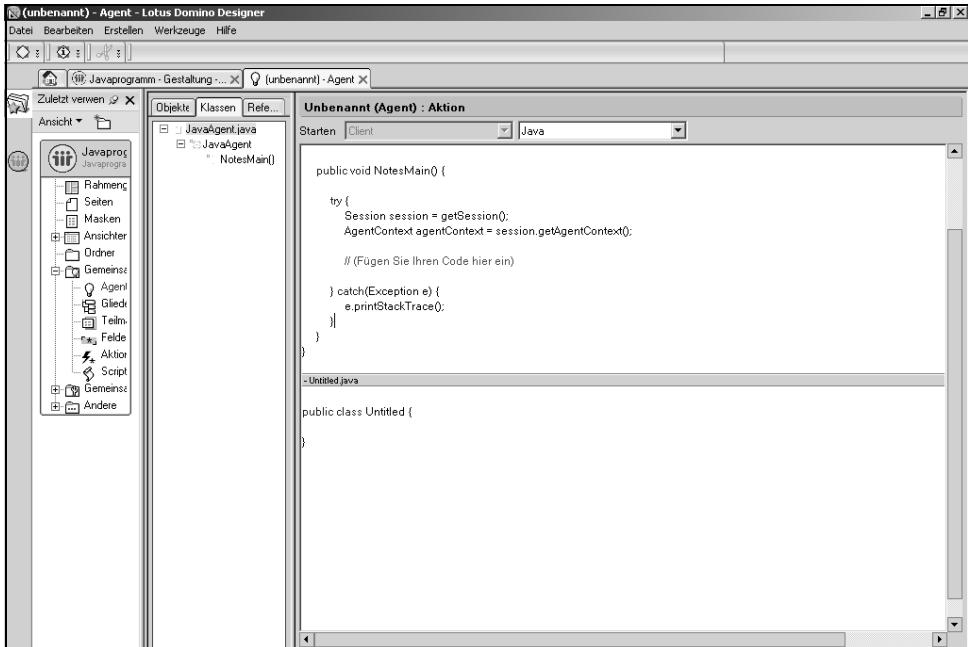


Abbildung 4.5: Neue Javaklasse

Um Klassen in einem Agent zu manipulieren, wählen Sie *Projekt bearbeiten* aus.

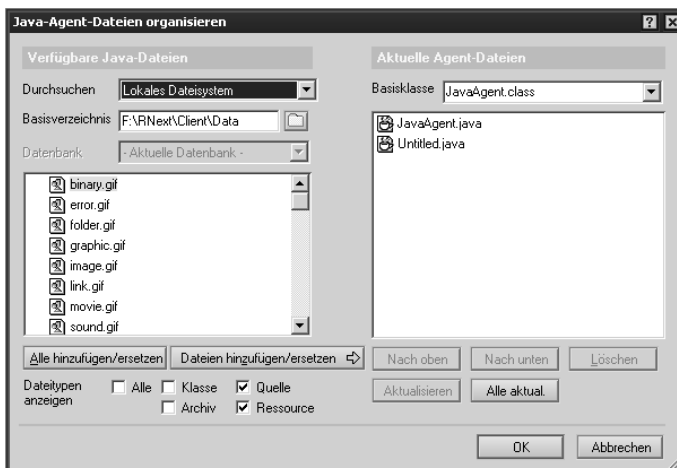


Abbildung 4.6: Externe Javadateien

Über dieses Fenster können Javaklassen aus externen Dateien in den Agent übernommen werden. Außerdem können bestehende Klassen gelöscht werden.

Importierte Javaklassen nutzen

Etwas anders sieht der Weg aus, wenn die Javaklassen für einen Agent aus externen Jar- oder Class-Dateien importiert werden sollen. Auch hier wird die Rubrik AGENTS unter GEMEINSAMER CODE ausgewählt. Nachdem dann der neue Agent angelegt wurde, muss als Sprache für den Agent nicht *Java*, sondern *importiertes Java* ausgewählt werden.

Danach kann durch die Betätigung der Schaltfläche am unteren Rand des Bildschirms die Jar- oder Class-Datei ausgewählt werden, die in den Agent eingefügt werden soll. Des Weiteren muss im unteren Bildschirmbereich der Name der Basisklasse angegeben werden. Die Basisklasse ist die Klasse aus dem importierten Javacode, welche als Hauptklasse des Agents ausgeführt wird.

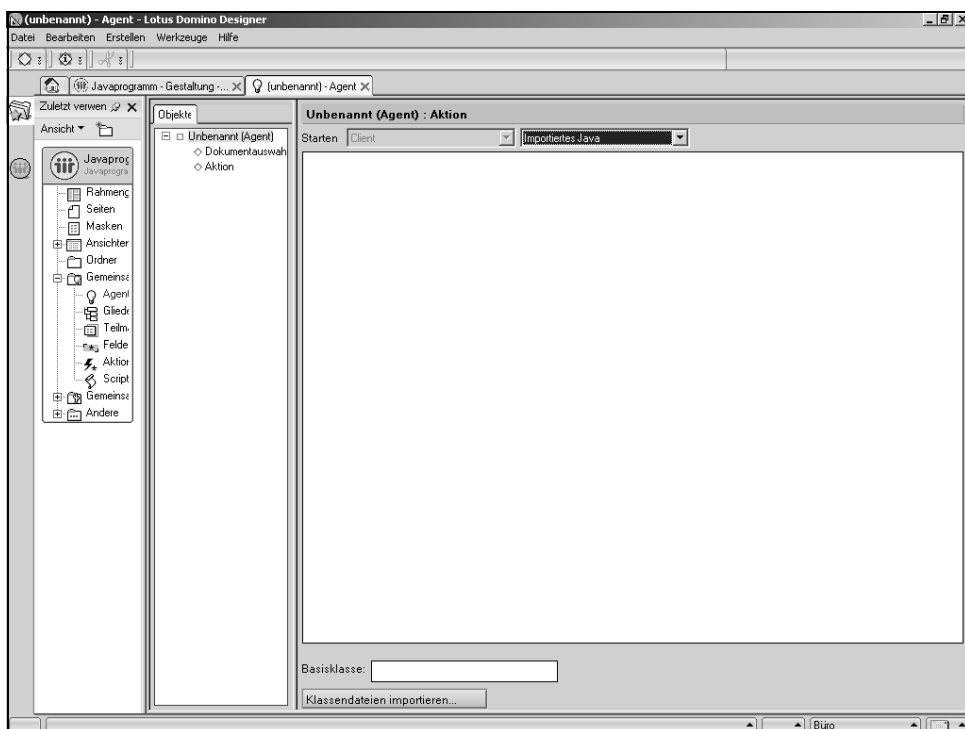


Abbildung 4.7: Importierter Java-Agent

Der Javacode in einem Agent

In jedem Agent gibt es eine Basisklasse, die die Methode beinhaltet, die beim Starten des Agents ausgeführt wird. Diese Basisklasse muss von der Klasse AgentBase abgeleitet sein und eine Methode NotesMain besitzen. Der Javacode dieser Methode wird beim Starten des Agents ausgeführt.

```
import lotus.domino.*;

public class JavaAgent extends AgentBase {

    public void NotesMain () {
        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

            // Auszuführende Programmzeilen

        } catch ( Exception e ) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.1: Beispiel Java-Agent

Die Eigenschaften und Methoden der Basisklasse des Agents

Die Basisklasse eines Agents ist immer abgeleitet von der Klasse *AgentBase*. Über diese Klasse erhält man Zugriff auf die Klasse *Agent*, diese stellt einige interessante Eigenschaften und Methoden zur Verfügung.

Die Klasse *Agent* bietet z. B. eine Eigenschaft, die verrät, ob der Agent aktiviert oder deaktiviert ist. Diese Eigenschaft heißt *Enabled* und wird mit *isEnabled* abgefragt und mit *setEnabled* gesetzt. Des Weiteren erfährt man durch die Eigenschaft *isPublic*, ob der Agent ein öffentlicher oder privater Agent ist. Außerdem verraten die Eigenschaften *NotesAgent* und *WebAgent* etwas über die Art und den Zweck des Agents.

```
class Agent {

    boolean isEnabled ();
    void setEnabled ( boolean flag );

    boolean isPublic ();
    boolean isNotesAgent ();
    boolean isWebAgent ();

}
```

Ebenso erfährt man den Namen des Agents über die Eigenschaft *Name*. Der Name des Besitzers bzw. des Autors des Agenten verraten die Eigenschaften *CommonOwner* und *Owner*.

```
class Agent {

    String getName ();
    String getCommonOwner ();
    String getOwner ();

}
```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Agent ag = ac.getCurrentAgent();
            Database db = ac.getCurrentDatabase();

            System.out.println ( "Name:" + ag.getName() );
            System.out.println ( "CommonOwner:" +
                ag.getCommonOwner() );
            System.out.println ( "Owner:" + ag.getOwner() );

        } catch(Exception e) {

            e.printStackTrace();

        };
    };
};

```

Listing 4.2: Beispiel für die Programmierung der Agent-Eigenschaften

Des Weiteren gibt es Eigenschaften wie Comment, LastRun, Owner, Parent, Query, Server-Name, Target und Trigger.

Methoden

Zwei Methoden der Klasse *AgentBase* verdienen besondere Aufmerksamkeit. Die erste Methode ist *run*. Mit dieser Methode kann ein Agent aus Java heraus gestartet werden. Der Methode kann optional als Parameter die Notes ID eines Dokuments mitgegeben werden. Dieses Dokument ist das so genannte Parameterdokument des Agents. Über dieses Dokument können Daten bzw. Parameter an den Agent übergeben werden. Die ID dieses Dokuments kann bei der Agentausführung über die Eigenschaft *ParameterDocID* ausgelesen werden.

```

class AgentBase {

    void run;
    void run ( String noteID;
    String getParameterDocID;

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

```

```

        Agent ag = ac.getCurrentAgent();
        Database db = ac.getCurrentDatabase();
        Document doc =
            db.getDocumentByID(ag.getParameterDocID());

        // Zugriff auf Parmeterdokument

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};
};

```

Listing 4.3: Beispiel Agent-Parameterdokument

Es gibt aber noch eine weitere Methode zur Ausführung des Agents. Mit der Methode `runOnServer` kann ebenso der Agent gestartet werden. Allerdings findet bei dieser Methode die Ausführung nicht auf dem aktuellen Computer statt, sondern auf dem Computer, auf dem sich die Datenbank befindet.

```

class Agent {

    void runOnServer;
    void runOnServer (String noteID );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Agent ag = ac.getAgent( "Testagent");

            ag.runOnServer();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.4: Beispiel runOnServer

Datenspeicherung zwischen den Aufrufen

Außer der Datenübergabe beim Starten des Agents ist es häufig notwendig, dass Daten zwischen zwei Ausführungen eines Agents gespeichert werden können. Dafür stellt das Notes System für jeden Agent ein spezielles Dokument zur Verfügung. Dieses Dokument wird bei der Erstellung oder der Änderung eines Agents automatisch angelegt. Sollte bereits ein Dokument existieren, wird dieses gelöscht.

Um auf dieses Dokument zuzugreifen, wird eine Referenz benötigt. Diese Referenz lässt sich aus der Eigenschaft `SavedData` ermitteln.

```
class Agent {  
    Document getSavedData;  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Agent ag = ac.getCurrentAgent();  
  
            Document doc = ag.getSavedData();  
            // Bearbeiten der Daten im Dokument  
  
            doc.save();  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    };  
};
```

Listing 4.5: Beispiel Agent-Datenspeicherung

4.1.2 Standalone Notes Anwendungen

Mit Java ist es zum Unterschied zu LotusScript möglich, Javaanwendungen ganz außerhalb des Notes Clients zu realisieren. Dabei unterscheiden sich diese Programme in ihrem Grundaufbau nicht von anderen Javaprogrammen. Auch bei Notes Anwendungen wird die Methode `main` als Einsprungspunkt genutzt. Anders als bei dem Java Agent müssen dann allerdings einige besondere Programmschritte unternommen werden, um die Notes Klassen aufrufen zu können. Die Notes Sitzung, die bei einem Agent vom Client oder Server automatisch erzeugt wird, muss im Standalone-Programm durch besondere Programmzeilen aufgebaut werden.

Bevor diese Programmzeilen allerdings in das Javaprogramm eingefügt werden können, muss die Art des Zugriffs bestimmt werden. Es gibt zwei Möglichkeiten, wie ein Zugriff von einem Javaprogramm aus auf Notes stattfinden kann.

Lokale Anwendungen

Die einfachste Art für ein Standalone-Programm, auf Notes zuzugreifen, ist eine Anwendung mit lokalem Zugriff. Das bedeutet, auf dem Computer, der das Javaprogramm ausführt, ist ein Server oder Client installiert. Dadurch können die Javaklassen über die JNI (Java Native Interface)-Schnittstelle direkt auf die DOI (Domino Object Interface)-Klassen zugreifen. Diese DOI-Klassen übernehmen dann die Kommunikation mit dem Notes Kernel.

Notes Javaklassen	Java Native Interface	DOI-Klassen	NotesKernel

Aufbau des Javaprogramms für den lokalen Zugriff

Um eine Javaanwendung mit lokalem Zugriff auf die Notes Elemente zu implementieren, gibt es zwei Ansatzmöglichkeiten.

Bei dem ersten Ansatz wird die Klasse *NotesThread* aus dem Package *lotus.domino* verwendet. Die Klasse der Anwendung muss von dieser Klasse abgeleitet werden. *NotesThread* selbst ist abgeleitet von der Javastandardklasse *Thread*.

In der Klasse *NotesThread* existiert die Methode *runNotes*, welche in der abgeleiteten Klasse überlagert und mit dem ausführbaren Code bestückt werden muss. In der Methode *main* selbst wird eine Instanz der abgeleiteten Anwendungsklasse *Thread* instanziiert und mit dem Aufruf *start* gestartet.

```
import lotus.domino.*;

public class Anwendungsklasse extends NotesThread {

    public static void main ( String argv[] ) {

        Anwendungsklasse anwendung = new Anwendungsklasse();
        anwendung.start();

    };

    public void runNotes() {

        try {
            Session se = NotesFactory.createSession();

            // Auszuführende Programmzeilen

        } catch ( Exception ex ) {
            e.printStackTrace();
        };

    };

};
```

Listing 4.6: Beispiel Standalone-Anwendung NotesThread

Bei dem anderen Ansatz für eine Anwendung mit lokalem Notes Zugriff wird die Anwendungsklasse nicht von der Klasse *NotesThread* abgeleitet. Dann allerdings muss vor dem ersten Zugriff auf ein Notes Element die statische Methode *sinitThread* der Klasse *NotesThread* gefolgt von der statischen Methode *createSession* der Klasse *NotesFactory* verwendet werden, um eine entsprechende Umgebung zu schaffen.

```
import lotus.domino.*;

public class Anwendungsklasse {

    public static void main ( String argv[] ) {
        try {
            NotesThread.sinitThread();
            Session se = NotesFactory.createSession();

            // Auszuführende Programmzeilen

        } catch ( Exception ex ) {
            e.printStackTrace();
        } finally {
            NotesThread.stermThread();
        };
    };
};
```

Listing 4.7: Beispiel Standalone-Anwendung

Wichtig bei dieser Variante ist der Aufruf der Methode *stermThread* im *finally*, da eine nicht ordnungsgemäß aufgeräumte Umgebung zu großen Problemen in der JVM (Java Virtual Machine) führen kann.

Entfernte Anwendungen

Die andere Zugriffsart für Anwendungen erfolgt über CORBA. CORBA ist die Abkürzung für Common Object Request Broker Architecture.

Was ist CORBA?

Als Computernetzwerke das Licht der Welt erblickten, wurde versucht, Anwendungen zu entwickeln, die auf mehreren Computern gleichzeitig Berechnungen ausführten. Um dies zu realisieren, wurden Protokolle benötigt, die die Kommunikation zwischen den verschiedenen Computern regeln. Einer der ersten Ansätze war damals die Socket-Kommunikation. Bei der Socket-Kommunikation werden Daten über eine Art Kanal ausgetauscht. Basierend auf der Socket-Kommunikation wurde bald das RPC (Remote Procedure Call)-Protokoll ins Leben gerufen. Dieses Protokoll erlaubt im Gegensatz zur Socket-Kommunikation nicht nur das Übertragen von Daten, sondern auch den Aufruf von Methoden über das Netz. Aber als die objektorientierte Programmierung immer mehr Einfluss gewann, musste auch für die Netzwerkkommunikation ein objektorientiertes Protokoll her. Das war der Zeitpunkt, zu dem CORBA entstanden ist.

Mit CORBA ist es möglich, Objekte über das Netzwerk zu erstellen und zu bearbeiten. Das bedeutet, dass ein Client ein Objekt auf einem Server erzeugen, es aber wie ein lokales Objekt behandeln kann. Somit ist es z. B. für den Client möglich, Methoden des Objekts aufzurufen, die dann auf dem Server ausgeführt werden.

Lokales Javaprogramm	Client JVM	ORB CORBA Client	Netzwerk	Domino ORB DIIOP Task	CORBA-Schnittstelle	DOI-Klassen	Notes Kernel

CORBA für den Zugriff auf Notes

Der Vorteil bei der Verwendung des CORBA-Protokolls ist, dass auf dem Computer, an dem die Java-Anwendung ausgeführt wird, keine Notes Software installiert werden muss.

Dieses CORBA-Protokoll wird bei Notes vom Server zur Verfügung gestellt. Das Bearbeiten der CORBA-Anfragen übernimmt dabei der Server-Task DIIOP. Dieser Server-Task muss dazu auf dem Server gestartet und konfiguriert sein. Um den Task zu starten, kann dieser in der notes.ini in der Zeile „ServerTask=“ eingetragen werden. Außerdem können verschiedene Einstellungen im Serverdokument des Servers vorgenommen werden.

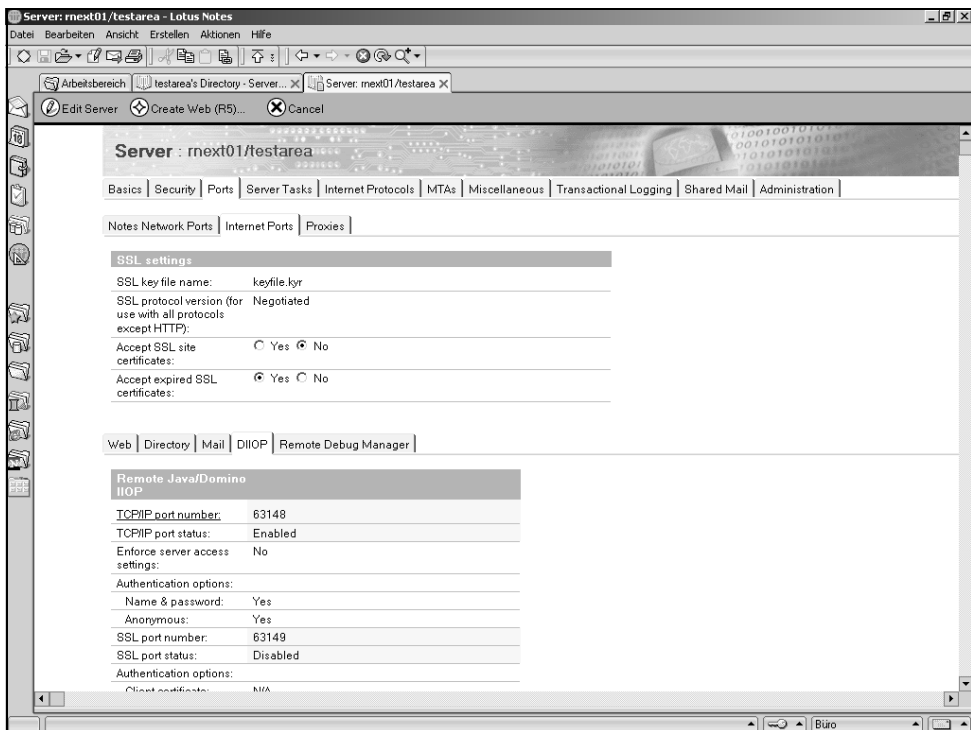


Abbildung 4.8: Das Serverdokument

Aufbau des Javaprogramms für den Zugriff über CORBA

Ein Programm, das über das CORBA-Protokoll auf den Server zugreift, verwendet ähnlich dem lokalen Programm die Klasse *NotesFactory* mit der Methode *createSession*, um eine Sitzung aufzubauen. Allerdings wird dabei kein Aufruf über die Klasse *NotesThread* benötigt.

```
import lotus.domino.*;

public class Anwendungsklasse

    public static void main ( S tring argv[]) {

        try {
            Session se = NotesFactory.createSession(
                servername, username, password);

            // Auszuführende Programmzeilen

        } catch ( Exception ex ) {
            e.printStackTrace();
        };

    };

};
```

Listing 4.8: Beispiel Standalone-Anwendung CORBA

4.1.3 Applet-Anwendungen

Was sind Applets?

Applets entstanden, als die Webseiten laufen lernten. Als die ersten HTML-Seiten das Internet bevölkerten, war der Punkt schnell erreicht, an dem es nicht mehr ausreichte, unbewegliche Seiten anzubieten. Es kam die Forderung auf, dass mit den Seiten Anwendungen verteilt werden sollten, welche nicht auf dem jeweiligen Server laufen, sondern in dem Browser bei der Person, die die Webseiten aufruft.

Solche Programme sind die Applets. Applets sind also z. B. in Webseiten integriert. Für die Ausführung dieser Applets wurde eine Spielwiese in den Internetbrowsern geschaffen. Diese Spielwiese ist die JVM, welche in den Internetbrowsern integriert wurde. Applets können also als Java-Anwendungen verstanden werden, die in der JVM auf einem beliebigen Computer mit Internetbrowser ausgeführt werden.

Ein Applet ohne Notes Zugriff basiert auf der Klasse *Applet*. Um ein Applet zu entwerfen, wird eine neue Klasse deklariert, die von der Klasse *Applet* abgeleitet wird. In der neuen Klasse werden verschiedene Methoden aus der Klasse *Applet* überlagert. Die Methoden, die im Regelfall überlagert werden, sind *init*, *start*, *stop* und *paint*.

```
class Applet {
    void init();
    void start();
    void stop();
    void paint( Graphics g );
};

import java.applet.* ;
import java.awt.* ;

public class Appletbeispiel extends Applet {

    public void init () {
        System.out.println ( "Applet.init()" );
    } ;

    public void start () {
        System.out.println ( "Applet.start()" );
    } ;

    public void stop () {
        System.out.println ( "Applet.stop()" );
    } ;

    public void paint ( Graphics g ) {
    } ;

};
```

Listing 4.9: Beispiel Standard-Applet

Methodenname	Beschreibung
init	Wird bei der Initialisierung des Applets ausgeführt
start	Wird beim dem Starten des Applets ausgeführt
stop	Wird beim Stoppen des Applets ausgeführt
paint	Wird bei der Anzeige des Applets ausgeführt

Tabelle 4.1: Vordefinierte Methoden der Klasse Applet

Applets mit Notes Zugriff

Notes stellt für die Entwicklung von Applets eine Basisklasse *AppletBase* zur Verfügung. Diese Basisklasse nimmt dem Programmierer Einiges an Arbeit ab. Dazu muss eine neue Klasse definiert werden, welche von dieser Basisklasse *AppletBase* abgeleitet wird. In der Klasse *AppletBase* sind verschiedene Methoden definiert, welche in bestimmten Situationen automatisch aufgerufen werden. Der Programmierer kann diese Methoden in seiner abgeleiteten Klasse beliebig überlagern.

```

class AppletBase {

    void notesAppletInit() {
    };

    void notesAppletDestroy() {
    };

    void notesAppletStart() {
    };

    void notesAppletStop() {
    };

}

```

Methodenname	Beschreibung
notesAppletInit	Ausführung bei der Initialisierung des Applets
notesAppletDestroy	Ausführung bei der Entfernung des Applets
notesAppletStart	Ausführung, wenn Applet gestartet wird
notesAppletStop	Ausführung, wenn Applet gestoppt wird

Tabelle 4.2: Vordefinierte Methoden der Klasse AppletBase

Importieren von Applets in Notes Datenbanken

Applets werden in eine Notes Datenbank durch die Wahl des Menüpunkts Java-Applet im Menü Erstellen importiert.

In diesem Fenster können dann die entsprechenden Dateien auf der linken Seite ausgewählt und in die Notes Datenbank importiert werden. Danach kann dieses Applet in den Notes Masken verwendet werden.

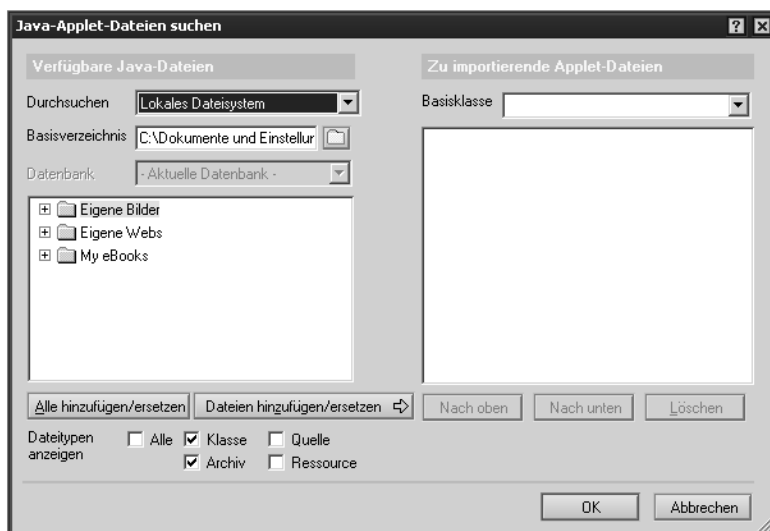


Abbildung 4.9: Dialogfenster zum Importieren von Applets

Einfügen von Applets in Notes Masken

Um ein Applet in einer Maske einzufügen, wird im Designer bei geöffneter Maske der Menüpunkt JAVA-APPLET im Menü ERSTELLEN gewählt. Dann sollte folgendes Dialogfenster erscheinen:



Abbildung 4.10: Dialogfenster zum Einfügen von Applets

In diesem Dialogfenster kann ausgewählt werden, ob sich das Servlet im Notes System befindet oder ob es auf einem externen Webserver liegt. Bei einem Applet, das sich auf dem Notes Server befindet, wird die erste Option ausgewählt. Danach kann der Speicherort der Javodatei gewählt werden. Wenn auf ein Applet auf einem anderen Server verwiesen wird, kann an dieser Stelle die URL angegeben werden.

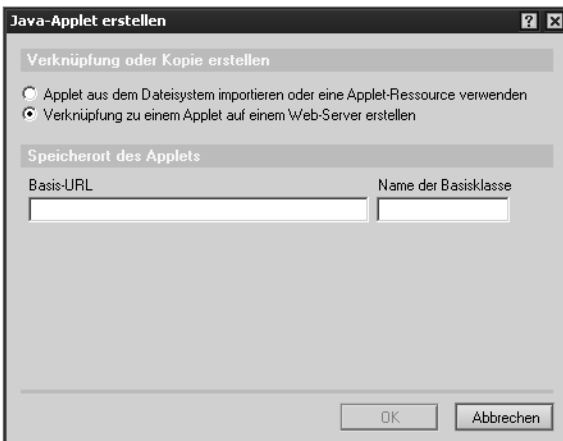


Abbildung 4.11: Dialogfenster Applet URL-Verknüpfung

4.2 Datenbanken

4.2.1 Einleitung

Die Notes Datenbanken werden in der Javaumgebung mithilfe der Klasse *Database* angesprochen. Über diese Klasse lassen sich dann die benötigten Referenzen für alle weiteren Elemente der Datenbank ermitteln.

4.2.2 Lokalisieren einer Notes Datenbank

Um mit einer vorhandenen Notes Datenbank arbeiten zu können, muss man sie erst finden. Um auf eine Liste aller Datenbanken zuzugreifen, welche sich in einem bestimmten Verzeichnis auf der lokalen Arbeitsstation oder auf einem entfernten Notes Server befinden, wird die Klasse *DbDirectory* bereitgestellt. Um eine Instanz dieser Klasse zu erhalten, muss die Funktion *getDbDirectory* der Klasse *Session* aufgerufen werden. Diese Funktion erwartet als Parameter optional den Namen des Servers oder die Konstante *null*. Nach dem Aufruf liefert die Funktion als Rückgabewert eine neu erzeugte Instanz der Klasse *DbDirectory*. Wenn kein Servername übergeben wird, wird angenommen, dass auf den aktuellen Server der Sessionumgebung zugegriffen werden soll. Bei einem Aufruf über IIOP darf kein Servername angegeben werden.

```
class Session {  
  
    DbDirectory getDbDirectory(String server);  
  
}
```

Wenn nach dem Aufruf der Funktion die benötigte Instanz dieser Klasse vorliegt, kann die Liste der Datenbanken auf dem angegebenen Server oder Arbeitsstation durchwandert werden. Die erste Datenbank erhält man durch den Aufruf der Funktion *getFirstDatabase*. Entweder erhält man den Wert *null*, falls keine Datenbank vorhanden ist, oder eine Referenz auf die erste Datenbank in Form einer Instanz der Klasse *Database*. Der Funktion *getFirstDatabase* muss beim Aufruf angegeben werden, welche Datenbanktypen durchsucht werden sollen. Dies erfolgt mit der Hilfe eines numerischen Parameters. Zur Auswahl stehen die Konstanten *DATABASE*, *TEMPLATE*, *REPLICA_CANDIDATE* und *TEMPLATE_CANDIDATE*, welche in der Klasse *DbDirectory* definiert sind.

Konstante	Bedeutung	Wert
DATABASE	Alle Datenbanken (.nsf, .nsg oder .nsh)	
TEMPLATE	Alle Vorlagendatenbanken (.ntf)	
REPLICA_CANDIDATE	Alle nicht für Replikation gesperrte Datenbanken	
TEMPLATE_CANDIDATE	Alle Datenbanken und Vorlagendatenbanken	

Tabelle 4.3: Die Konstanten der Klasse *DbDirectory* zur Definition der Datenbanktypen

Um die weiteren Datenbanken zu ermitteln, muss nach der Funktion *getFirstDatabase* die parameterlose Funktion *getNextDatabase* aufgerufen werden. Entweder erhält man

als Rückgabewert dadurch die Referenz auf die nächste gefundene Datenbank, oder – falls keine weiteren Datenbanken vorhanden sind –, den Wert null zurück.

```
class DbDirectory {

    Database getFirstDatabase (int type);

    Database getNextDatabase ();

}

public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            DbDirectory dbdir = se.getDbDirectory(null);
            System.out.println("Lokale Datenbanken\n");
            Database db = dbdir.getFirstDatabase
                (DbDirectory.DATABASE);
            while (db != null) {
                System.out.println(db.getTitle());
                db = dbdir.getNextDatabase();
            }

        } catch (Exception e) {

            e.printStackTrace();

        }

    };

};
```

Listing 4.10: Beispiel *DbDirectory.getFirstDatabase* und *DbDirectory.getNextDatabase*

4.2.3 Öffnen einer Datenbank

Wenn man weiß, wo sich die Datenbank, mit der man arbeiten will, befindet, muss man diese für den weiteren Zugriff öffnen. Es gibt einige verschiedene Vorgehensweisen, um eine Notes Datenbank zu öffnen. Sie unterscheiden sich je nachdem, in welcher Situation und welche Art von Datenbank geöffnet werden soll.

Öffnen einer Datenbank, die als Referenz der Klasse Database vorliegt

In vielen Fällen hat man eine Instanz der Klasse *Database* und will diese öffnen. Das ist z. B. der Fall, wenn man verschiedene Datenbanken mithilfe der Methoden *getFirstDatabase* und *getNextDatabase* der Klasse *DbDirectory* durchläuft. In diesem Fall erhält man nämlich eine Referenz auf eine Datenbank, die bis jetzt noch nicht geöffnet wurde. Um diese zu öffnen, kann man die Methode *open* der Klasse *Database* einsetzen.

```
class Database {  
  
    boolean open();  
  
}  
  
public class JavaAgent extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            DbDirectory dbdir = se.getDbDirectory(null);  
            System.out.println("Lokale Datenbanken\n");  
            Database db = dbdir.getFirstDatabase  
                (DbDirectory.DATABASE);  
            while (db != null) {  
                System.out.println(db.getTitle());  
                db.open();  
                System.out.println ( db.getCreated().getLocalTime());  
                db = dir.getNextDatabase();  
            }  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        };  
  
    };  
  
};  
  
};
```

Listing 4.11: Beispiel Database.open

Öffnen einer beliebigen Datenbank mit Namen bzw. Servernamen

Um eine beliebige Datenbank auf einem Server oder einer Arbeitsstation zu öffnen, muss man entweder deren Namen, ggf. auch den Server oder die Replikations-ID kennen.

Um eine Datenbank zu öffnen, deren Namen man kennt, gibt es die Methode `getDatabase` der Klasse `Session`. Diese Methode erwartet als Parameter den Servernamen sowie den Namen der Datenbank. Außerdem kann optional mit einem booleschen Parameter bestimmt werden, ob die Datenbank, falls sie nicht geöffnet werden kann, automatisch angelegt werden soll. Wenn der aktuelle Server der Session bei dem Zugriff verwendet werden soll, muss anstelle des Servernamens die Konstante `null` übergeben werden. Nachdem die Methode erfolgreich aufgerufen wurde, liefert sie eine Referenz auf die geöffnete Datenbank zurück.

Sollte es sich um eine IIOP-Anwendung handeln, so darf als Parameter kein Servername und somit nur der Wert `null` angegeben werden.

```

class Session {

    Database getDatabase(String server, String db);

    Database getDatabase(
        String server, String db, boolean createonfail );

}

// Öffnen der lokalen names.nsf
public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = se.getDatabase
                ("", "names.nsf" );
            if ( db != null ) {
                System.out.println(db.getTitle());
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.12: Beispiel *Session.getDatabase*

Eine weitere Möglichkeit zum Öffnen einer Datenbank bietet die Methode `openDatabase` der Klasse *DbDirectory*. Bei dieser Methode werden nur der Datenbankname und ein optionaler boolescher Parameter übergeben. Auch hier bestimmt der boolesche Parameter, ob die Datenbank im Bedarfsfall angelegt werden soll. Nach dem erfolgreichen Aufruf liefert die Methode eine Referenz auf die geöffnete Datenbank zurück.

```

class DbDirectory {

    Database openDatabase(String dbfile);
    Database openDatabase(String dbfile, boolean failover);

}

// Öffnen der lokalen names.nsf
public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();

```

```

        AgentContext ac = se.getAgentContext();
        DbDirectory dbdir = se.getDbDirectory("white");
        Database db = dbdir.openDatabase
            ("log.nsf");
        if ( db != null ) {
            System.out.println(db.getTitle());
        };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};
};

```

Listing 4.13: *Beispiel DbDirectory.openDatabase*

Öffnen einer beliebigen Datenbank anhand ihrer Replikations-ID

Die Replikations-ID einer Datenbank kann genauso zum Öffnen einer Datenbank genutzt werden wie ihr Datenbank- oder Servername. Um eine Datenbank mit ihrer Replikations-ID zu öffnen, steht die Methode `openDatabaseByReplicaID` der Klasse *DbDirectory* zur Verfügung. Bei dieser Methode muss als Parameter nur die Replikations-ID als String übergeben werden. Nach dem Aufruf erhält man eine Referenz auf die geöffnete Datenbank zurück. Syntax:

```

class DbDirectory {
    Database openDatabaseByReplicaID(String rid);
}

public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            DbDirectory dbdir = se.getDbDirectory("white");
            Database db = dbdir.openDatabaseByReplicaID
                ("8525687E:005800B8");
            if ( db != null ) {
                System.out.println(db.getTitle());
            }

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.14: *Beispiel DbDirectory.openDatabaseByReplicaID*

Öffnen einer beliebigen Datenbank, wenn diese vor einem bestimmten Zeitpunkt geändert worden ist

Die Klasse *DbDirectory* stellt die Methode *openDatabaseIfModified* zur Verfügung, welche dazu dient, das Öffnen einer Datenbank abhängig zu machen von dem letzten Änderungsdatum der Datenbank. Die Methode erwartet zwei Parameter. Als Erstes wird der Datenbankname erwartet und als Zweites ein Datum. Die Datenbank wird nur geöffnet, wenn nach dem angegebenen Datum eine Änderung stattgefunden hat. Sollte dies nicht der Fall sein, wird der Rückgabewert null zurückgeliefert, ansonsten erhält man eine Referenz auf die geöffnete Datenbank.

```
class DbDirectory {

    Database openDatabaseIfModified( String dbfile, DateTime date );

}

public class JavaAgent extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            DbDirectory dbdir = se.getDbDirectory(null);
            Database db = dbdir.openDatabaseIfModified (
                "log.nsf",
                se.createDateTime("Today")
            );
            if ( db != null ) {
                System.out.println(db.getTitle());
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.15: Beispiel *DbDirectory.openDatabaseIfModified*

Öffnen der aktuellen Datenbank in einem Agent

Bei der Agentprogrammierung will man häufig die aktuelle Datenbank, in der sich der Agent befindet, öffnen. Hierfür gibt es die Eigenschaft *CurrentDatabase* der Klasse *AgentContext*. Diese Eigenschaft kann mit der Funktion *getCurrentDatabase* ausgelesen werden. Diese Funktion liefert nach dem Aufruf eine Referenz auf die geöffnete aktuelle Datenbank zurück.

```

class AgentContext {
    Database getCurrentDatabase();
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();
            Database db = agentContext.getCurrentDatabase();

            // Programmcode
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}

```

Listing 4.16: *Beispiel AgentContext.getCurrentDatabase*

Öffnen der persönlichen oder öffentlichen Adressdatenbank

Um eine Adressdatenbank der aktuellen Session zu ermitteln, gibt es in der Klasse *Session* die Eigenschaft *AddressBooks*. Diese Eigenschaft wird mit der Methode *getAddressBooks* ausgelesen. Nach dem Auslesen liefert diese Methode einen Vektor der für die aktuelle Session gültigen Adressbücher zurück. Die Adressbücher werden in diesem Vektor in Form von Referenzen auf die Database-Instanzen der Adressbücher übergeben. Um in diesen Datenbanken zu arbeiten, muss allerdings nach dem Aufruf von *getAddressBooks* das jeweilige Adressbuch noch mit der Methode *open* der Klasse *Database* geöffnet werden.

```

class Session {
    java.util.Vector getAddressBooks();
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session session = getSession();
            Vector vec = session.getAddressBooks();
            Enumeration enum = vec.elements();
            while ( enum.hasMoreElements() ) {
                Database db = (Database)enum.nextElement();
            }
        }
    }
}

```

```

        // Programmcode

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

};

};

```

Listing 4.17: Beispiel *Session.getAddressBooks*

In Notes gibt es zwei Kategorien von Adressbüchern. Um zu ermitteln, ob es sich um ein *privates* oder öffentliches Adressbuch handelt, gibt es die Funktionen *isPrivateAddressBook* oder *isPublicAddressBook* der Klasse *Database*. Diese Funktionen liefern als Rückgabewert einen booleschen Ausdruck zurück.

```

class Database {

    boolean isPublicAddressBook();
    boolean isPrivateAddressBook();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();
            Database db = agentContext.currentDatabase();
            if ( db.isPublicAddressBook() ) {
                System.out.println ( "Öffentliches Adressbuch" );
            } else if ( db.isPrivateAddressBook() ) {
                System.out.println ( "Privates Adressbuch" );
            } else
                System.out.println ( "Kein Adressbuch" );
        }
        } catch(Exception e) {
            e.printStackTrace();
        }
    };

}

```

Listing 4.18: Beispiel *Database.isPublicAddressBook* und *Database.isPrivateAddressBook*

Öffnen der aktuellen Maildatenbank

Jeder Anwender hat im Regelfall unter Notes eine Maildatenbank. Um diese Maildatenbank zu öffnen, gibt es die Methode `openMailDatabase` der Klasse *DbDirectory*. Durch den Aufruf dieser Methode lässt sich die Maildatenbank des aktuellen Users öffnen. Dieser Aufruf darf allerdings nur entweder auf einer Arbeitsstation oder innerhalb eines Serveragents auf dem Mailserver des Agent-Besitzers erfolgen. Die Methode erwartet keine Parameter und liefert als Rückgabewert eine Referenz auf die geöffnete Datenbank zurück.

```
class DbDirectory {  
    Database openMailDatabase();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
  
        try {  
            Session session = getSession();  
            DBDirectory dbdir = session.getDbDirectory(null);  
            Database db = dbdir.openMailDatabase();  
  
            // Programmcode  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
};
```

Listing 4.19: Beispiel *DbDirectory.openMailDatabase*

Öffnen der Webnavigator-Datenbank

Für den Abruf von Webseiten existiert unter Notes eine spezielle anwenderspezifische Webdatenbank. Diese Datenbank lässt sich durch die Methode `getURLDatabase` der Klasse *Session* in einem Javaprogramm öffnen. Bei dieser Methode wird kein Parameter erwartet und eine Referenz auf die geöffnete Webdatenbank zurück geliefert.

```
class Session {  
    Database getURLDatabase();  
}
```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session session = getSession();
            DBDirectory dbdir = session.getDbDirectory(null);
            Database db = dbdir.getURLDatabase();

            // Programmcode

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.20: Beispiel *Session.getURLDatabase*

Öffnen einer Datenbank in einer Remote-Anwendung

Speziell für eine Remote-Anwendung stellt Notes in der Klasse *Session* eine Methode zur Verfügung, um eine Datenbank sowie Ansichten, Masken, Dokumente oder Agents zu öffnen, die durch eine URL spezifiziert wird. Diese Methode hat den Namen *resolve*, sie erwartet als Parameter die URL des gesuchten Objekts, z. B. einer Datenbank, und liefert als Rückgabewert eine Referenz auf eine Instanz der Klasse *Base*. Diese Referenz muss nach dem Aufruf entsprechend des erwarteten Ergebnisses gecastet werden.

```

class Session {

    Base resolve(String url);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Database db = (Database)se.resolve
                ("notes://hostname/names.nsf?OpenDatabas");
            View view = (View)se.resolve
                ("notes://hostname/names.nsf/drafts?OpenView");
            Form form = (Form)se.resolve
                ("notes://hostname/names.nsf/memo?OpenForm");
            Document document =(Document)se.resolve
                ("notes://hostname/names.nsf/memo?OpenDocument");

```

```

        Agent agent = (Agent)se.resolve
            ("notes://hostname/names.nsf/testagent?OpenAgent");

    } catch(Exception e) {

        e.printStackTrace();

    };

};

};

```

Listing 4.21: Beispiel *Session.resolve*

Der Zugriff auf die Datenbank über die Eigenschaft *ParentDatabase*

In den meisten Klassen, die genutzt werden, um auf Bestandteile einer bestimmten Datenbank zuzugreifen, existiert die Eigenschaft *ParentDatabase*. Diese kann jederzeit mit der Funktion *getParentDatabase* ausgelesen werden und liefert eine Referenz auf die entsprechende Datenbank. Gegebenenfalls muss die Datenbank dann an diesen Punkt noch mit der Methode *open* – wie oben beschrieben – geöffnet werden.

Überprüfen, ob eine Datenbank schon geöffnet wurde

Nachdem wir eben mehrere Wege gesehen haben, eine Datenbank zu öffnen, folgt hier eine Möglichkeit zu überprüfen, ob wir unser Ziel, eine geöffnete Datenbank, erreicht haben. Um zu überprüfen, ob eine Datenbank geöffnet ist, muss die Funktion *isOpen* der Klasse *Database* aufgerufen werden. Diese liefert den booleschen Wert *true*, falls die Datenbank geöffnet ist, ansonsten den booleschen Wert *false* zurück.

```

class Database {

    boolean isOpen();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session session = getSession();
            DBDirectory dbdir = session.getDbDirectory(null);
            Database db = dbdir.openMailDatabase();

            if ( db.isOpen() ) {
                // Programmcode
            } else {
                System.out.println ( "Probleme beim Öffnen der DB" );
            }

        } catch(Exception e) {

```

```

        e.printStackTrace();

    };

};

};

```

Listing 4.22: Beispiel Database.isOpen

4.2.4 Erstellen, Komprimieren, Löschen und Replizieren einer Datenbank

Nachdem im letzten Abschnitt gezeigt wurde, wie man auf eine bereits existierende Notes Datenbank zugreift, soll in diesem Abschnitt gezeigt werden, wie man Datenbanken erstellen, komprimieren, löschen oder replizieren kann.

Erstellen einer leeren Datenbank

Als erste Möglichkeit soll hier gezeigt werden, wie Sie eine leere Datenbank erstellen können. Dies erfolgt mit der Methode `createDatabase` der Klasse *DbDirectory*. Eine mit dieser Methode erstellte Datenbank beinhaltet weder eine Ansicht noch irgendeine Maske. Dadurch ist es für keinen Anwender möglich, diese Datenbank in einem Notes Client zu öffnen. Der Zugriff auf solche Datenbanken ist nur über ein Notes Programm in Java, NotesScript oder Formelsprache möglich. Es wird von der Methode als ersten Parameter der Name der Datenbank erwartet. Mit dem zweiten Parameter kann optional über einen booleschen Wert gesteuert werden, ob die Datenbank durch den Methodenaufruf gleichzeitig geöffnet werden soll.

```

class DbDirectory {

    Database createDatabase(String dbfile);

    Database createDatabase(String dbfile, boolean open);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session session = getSession();
            DBDirectory dbdir = session.getDbDirectory(null);
            Database db = dbdir.openMailDatabase();

            if ( db.isOpen() ) {
                // Programmcode
            } else {
                System.out.println ( "Probleme beim Öffnen der DB" );
            }
        }
    }
}

```

```

        } catch(Exception e) {
            e.printStackTrace();
        };

    };

};

```

Listing 4.23: Beispiel DbDirectory.createDatabase

Erstellen einer Datenbank basierend auf einer Vorlagendatenbank

Im vorherigen Beispiel wurde eine Datenbank ohne ein einziges Gestaltungselement oder Dokument erstellt. Um im Gegensatz dazu eine Datenbank basierend auf einer Datenbankschablone zu generieren, gibt es die Methode `createFromTemplate` der Klasse `Database`. Dabei muss als erster Parameter der Name des Servers angegeben werden, auf dem die Datenbank erstellt werden soll. Dem folgen muss als zweiter Parameter der Dateiname der neuen Datenbank. Mit dem letzten booleschen Parameter kann gesteuert werden, ob spätere Änderungen an der Datenschemablone in der neu erstellten Datenbank übernommen werden sollen. Wenn hier der Wert *true* angegeben wird, werden die Änderungen später übernommen.

```

class Database {

    Database createFromTemplate (
        String server, String dbfile, boolean inherit );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Database tp = se.getDatabase(null,"discsw50.ntf");
            Database db = tp.createFromTemplate(null,"neuedb", true );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.24: Beispiel Database.createFromTemplate

Erstellen einer Kopie einer Datenbank

Außer den Möglichkeiten, eine leere Datenbank oder eine Datenbank basierend auf einer Datenbankschablone zu erstellen, bietet die Methode `createCopy` der Klasse *Database* die Möglichkeit, eine neue Kopie einer existierenden Datenbank zu erstellen. Bei dieser Methode wird der Servername als erster und der Dateiname der zu erstellenden Datenbank als zweiter Parameter erwartet. Des Weiteren gibt es die Option, eine maximale Größe in Gigabyte für die neue Datenbank festzulegen. Dies erfolgt mit dem dritten optionalen Parameter.

```
class Database {

    Database createCopy(String server, String dbfile);

    Database createCopy(
        String server, String dbfile, int maxsize );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Database sc = se.getDatabase(null,"names.nsf");
            Database dt = sc.createCopy (null,"names2.nsf" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.25: Beispiel Database.createCopy

Erstellen einer Datenbankreplik sowie das Replizieren einer Datenbank

Eine der herausragenden Merkmale von Notes ist die Replikationstechnik. Um das Anlegen einer Datenbankreplik unter Java zu ermöglichen, wird die Methode `createReplica` der Klasse *Database* angeboten. Nach der Angabe der zwei Parameter, des Servernamens und des Dateinamens, wird eine neue Replik der Datenbank an der angegebenen Stelle erstellt.

```
class Database {

    Database createReplica( String server, String dbfile );

    Database createReplica(
        String server, String dbfile, int maxsize );

}
```

Allerdings enthält die neue Replik der Datenbank noch keine Dokumente oder Gestaltungselemente. Die Replikation der Dokumente oder Gestaltungselemente muss erst entweder durch den Aufruf der entsprechenden Methode `replicate` der Klasse *Database* oder durch den Anwender im Frontend erfolgen. Die Methode `replicate` der Klasse *Database* erwartet dabei den Namen des Servers, mit dem repliziert werden soll. Vor der Replizierung kann die Datenbank in der Oberfläche des Notes Client nicht geöffnet werden.

```
class Database {  
  
    boolean replicate(String server);  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            Database db = se.openDatabase(null,"names.nsf");  
            Database dt = sc. createreplica(null,"names2.nsf" );  
            db.replicate ( null );  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        };  
  
    };  
};
```

Listing 4.26: *Beispiel Database.replicate*

Komprimieren einer Datenbank

Die Pflanzen und viele Notes Datenbanken haben eines gemeinsam: sie wachsen in der Regel unaufhaltsam. Um eine Notes Datenbank zu stutzen bzw. zu komprimieren, wird die Methode `compact` der Klasse *Database* verwendet. Diese parameterlose Methode sorgt dafür, dass eine lokal liegende Datenbank komprimiert wird. Sie liefert als Rückgabewert die Differenz des Speicherbedarfs vor und nach der Komprimierung als Integerwert zurück. Sollte die Datenbank sich auf dem lokalen Server oder der Arbeitsstation befinden, wird eine Notes Exception geworfen.

```
class Database {  
  
    int compact();  
  
}
```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Database db = se.openDatabase(null,"log.nsf");
            db.compact();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.27: Beispiel Database.compact

Löschen einer Datenbank

Damit auch die Möglichkeit gegeben ist, eine einmal erstellte Datenbank wieder los zu werden, wird von der Klasse *Database* die parameterlose Methode *remove* angeboten. Durch den Aufruf dieser Methode wird die entsprechende Notes Datenbank unwiderruflich gelöscht.

```

class Database {

    int remove();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Database db = se.openDatabase ( null, "temp.nsf" );
            db.remove();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.28: Beispiel Database.remove

4.2.5 Die wichtigsten Merkmale einer Datenbank

Das sicher interessanteste Merkmal einer Datenbank ist der Ort, an dem sich die Datenbank befindet. Des Weiteren interessant ist die Frage, wann die Datenbank angelegt bzw. zuletzt geändert worden ist.

Der Servername in der Datenbank

Jede Datenbank, das heißt genauer jede Replik einer Datenbank, liegt auf einem Server. Der Name dieses Servers lässt sich in der Eigenschaft `server` in der Klasse *Database* finden. Ausgelesen wird diese nicht änderbare Eigenschaft mit der Funktion `getServer`. Sollte die Datenbank auf einer Arbeitsstation liegen, so wird dies durch einen Leerstring als Servername angezeigt.

```
class Database {  
    String getServer();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            DbDirectory dd = se.getDbDirectory(null);  
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );  
            while ( db != null ) {  
                db.open();  
                System.out.println ( db.getFilePath()+" - " +  
                    db.getServer());  
                db.close();  
                db = dd.getNextDatabase();  
            };  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.29: *Beispiel Database.getServer*

Dateipfad und Dateiname der Datenbank

Der Standort jeder Notes Datenbank bzw. Replik wird außer durch den Servernamen auch durch den Dateipfad und den Dateinamen bestimmt. Um diese beiden schreibgeschützten Informationen aus den Eigenschaften `filename` und `filepath` auslesen zu

können, stehen die Funktionen `getFileName` und `getFilePath` der Klasse *Database* zur Verfügung.

```
class Database {
    String getFileName();
    String getFilePath();
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getFilePath()+" - " +
                    db.getFileName());
                db.close();
                db = dd.getNextDatabase();

            };
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.30: Beispiel *Database.getFileName* und *Database.getFilePath*

Datenbanktitel der Datenbank

Jede Datenbank hat unter Notes einen freien und von Server, Dateinamen oder Dateipfad unabhängigen Datenbanktitel. Die Eigenschaft `title` befindet sich in der Klasse *Database* und kann durch `getTitle` gelesen und durch `setTitle` geschrieben werden.

Dabei sei an dieser Stelle darauf hingewiesen, dass der Titel einer Datenbank, in der sich das gerade laufende Programm befindet, nicht geändert werden kann.

```
class Database {
    String getTitle();
    void setTitle( String title );
}
```

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getTitle() );
                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};
```

Listing 4.31: Beispiel Database.Title

Das Erstelldatum der Datenbank

Das Datum der Geburt, also der Erstellung der Datenbank, liefert die Eigenschaft `Created` über die Funktion `getCreated` der Klasse *Database*. Als Rückgabewert erhält man das Datum in Form einer Instanz der Notes Klasse *DateTime*.

```
class Database {

    DateTime getCreated();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getCreated() );
                db.close();
                db = dd.getNextDatabase();

            };

        };

    };
};
```

```

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};

```

Listing 4.32: Beispiel Database.Created

Das Änderungsdatum der Datenbank

Die letzte Änderung verrät die Eigenschaft `LastModified` der Klasse *Database*. Durch die Funktion `getLastModified` erhält man diese Information in Form eines Werts mit dem Notes Datentyp `DateTime`.

```

class Database {

    DateTime getLastModified();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getLastModified() );
                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.33: Beispiel Database.LastModified

4.2.6 Die replikationsrelevanten Eigenschaften einer Datenbank

Wie eine Replik einer Datenbank angelegt und wie eine Replizierung einer Datenbank ausgelöst wird, wurde in einem vorherigen Abschnitt gezeigt. Hier werden jetzt weitere für die Replikation notwendige Eigenschaften vorgestellt.

Die Replikations-ID

Um die Repliken einer Notes Datenbank eindeutig zu identifizieren, wird die so genannte Replikations-ID verwendet. Diese Replikations-ID ist in jeder Notes Datenbank gespeichert und findet sich in der Eigenschaft `ReplicaID` der Klasse *Database* wieder. Um sie auszulesen, kann die Funktion `getReplicaID` aufgerufen werden. Diese liefert als Rückgabewert die Replikations-ID als Hexadezimalzahl in einem String.

```
class Database {  
    String getReplicaID();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            DbDirectory dd = se.getDbDirectory(null);  
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );  
  
            while ( db != null ) {  
                db.close();  
                System.out.println ( db.getReplicaID () );  
                db.close();  
                db = dd.getNextDatabase();  
            };  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.34: *Beispiel Database.ReplicaID*

Die Referenz auf den Replikaktionsdatenblock

Alle für die Replikation wichtigen Eigenschaften einer Datenbank finden sich in der *Notes* Klasse mit *Replication* wieder. Jede Instanz der Klasse *Database* beinhaltet genau eine Instanz dieser Klasse. Durch den Aufruf *getReplicationInfo* der *Database*-Klasse wird eine Referenz auf diese Instanz zurückgeliefert.

```
class Database {
    Replication getReplicationInfo();
}
```

An dieser Stelle ist besonders zu beachten, dass nach etwaigen Änderungen von Eigenschaften in dieser Instanz der Klasse *Replication* diese mit der Methode *save* gespeichert bzw. mit der Methode *reset* zurückgesetzt werden müssen.

```
class Replication {
    int save();
    int reset();
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replicate rep = db.getReplicationInfo();

                // Zugriffe auf Replikationsdaten

                rep.save();

                db.close();
                db = dd.getNextDatabase();
            };
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.35: Beispiel *Replication.save*

Die Weitergabe der Löschung von Dokumenten

Bei der Replikation zweier Repliken einer Datenbank kann bestimmt werden, ob Löschungen von Dokumenten in der einen Replik an die andere Replik übertragen werden sollen. Diese Regel wird durch zwei Eigenschaften in der Klasse *Replication* in Java abgebildet.

Ausgehende Löschungen von Dokumenten

Mit der Eigenschaft *IgnoreDeletes* in der Klasse *Replication* wird festgelegt, ob Löschungen von Dokumenten in der Replik bei der Replizierung an eine andere Replik übermittelt werden sollen. Wenn die Eigenschaft den Wert *true* besitzt, werden die Löschungen nicht weitergegeben, bei dem Wert *false* erfolgt die Weitergabe. Das Auslesen dieser Eigenschaft erfolgt mit der Funktion *isIgnoreDeletes* und wird gesetzt mit der Funktion *setIgnoreDeletes*.

```
class Replication {  
  
    boolean isIgnoreDeletes();  
  
    void setIgnoreDeletes(boolean bValue);  
  
}
```

Reinkommende Löschungen von Dokumenten

Ob Löschungen, die in einer anderen Replik erfolgt sind, in die eigene Replik übernommen werden sollen, bestimmt die Eigenschaft *IgnoreDeletes* in der Klasse *Replication*. Sollte die Eigenschaft den Wert *true* besitzen, werden die Löschungen nicht angenommen, bei dem Wert *false* werden die Löschungen in die eigene Replik eingepflegt. Das Lesen dieser Eigenschaft erfolgt mit der Funktion *isIgnoreDestDeletes* und das Setzen mit *setIgnoreDestDeletes*.

```
class Replication {  
  
    boolean isIgnoreDestDeletes();  
    void setIgnoreDestDeletes(boolean bValue);  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            DbDirectory dd = se.getDbDirectory(null);  
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );  
            while ( db != null ) {  
  
                db.open();  
                Replication rep = db.getReplicationInfo();
```



```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                System.out.println ( "Datenbank:" + db.getTitle()+
                    " | " + getPriority() );

                db.close();
                db = dd.getNextDatabase();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.37: Beispiel Replication.Priority

Ein- und Ausschalten der Replikation

Selbstverständlich gibt es unter Java auch, falls man seine Ruhe haben will, die Möglichkeit, die Replikation einer Datenbank komplett auszuschalten. Dies wird durch die Eigenschaft `disabled` vom Datentyp `boolean` der Klasse *Replication* ermöglicht. Ausgelesen wird der Wert der Eigenschaft mit der Funktion `isDisabled` und gesetzt mit der Funktion `setDisabled`.

```

class Replication {

    boolean isDisabled();
    void setDisabled(boolean bValue);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );

```

```

while ( db != null ) {

    db.open();
    Replication rep = db.getReplicationInfo();

    System.out.print ( "Datenbank:" + db.getTitle() );
    if ( rep.isDisabled() ) {
        System.out.print ( " | Disable" );
    } else {
        System.out.print ( " | Enable" );
    };

    db.close();
    db = dd.getNextDatabase();

};

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.38: Beispiel Replication.disabled

Löschen der Replikationshistorie

Für jede Replik wird vom Notes System eine Replikationshistorie geführt. Um diese zu löschen, bietet die Klasse *Replication* die Methode `clearHistory`.

```

class Replication {

    int clearHistory ();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                rep.clearHistory();

                db.close();
                db = dd.getNextDatabase();
            }
        }
    }
}

```

```

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

    };

};

```

Listing 4.39: Beispiel Replikation.clearHistory

4.2.7 Cutoff-Verhalten

In vielen Anwendungsfällen unter Notes kommt es vor, dass Datenbanken eingesetzt werden, um aktuelle Informationen austauschen. Da dann häufig gefordert wird, dass Dokumente nach einer bestimmten Zeit aus der Datenbank automatisch entfernt werden sollen, gibt es hierfür unter Notes einen bestimmten Mechanismus.

Das Cutoff-Intervall

Für das automatische Löschen alter Dokumente muss in der Datenbank angegeben werden, nach welchem Zeitabschnitt die Dokumente entfernt werden sollen. Dieser Zeitabschnitt wird in Java durch die Eigenschaft `cutoffInterval` der Klasse *Replication* repräsentiert. Diese Eigenschaft beinhaltet die Anzahl an Tagen, nach denen ein Dokument gelöscht werden soll. Geschrieben wird diese Eigenschaft mit der Funktion `setCutoffInterval` und gelesen mit `getCutoffInterval`.

```

class Replication {

    long getCutoffInterval();

    void setCutoffInterval(long interval);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                rep.clearHistory();

                db.close();
                db = dd.getNextDatabase();
            }
        }
    }
}

```

```

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

    };
};

```

Listing 4.40: Beispiel Replication.CutoffInterval

Das Cutoff-Datum

Eine einfache Möglichkeit, das aktuelle Cutoff-Datum zu ermitteln, ist die Funktion `getCutoffDate` der Klasse *Replication*. Diese Funktion berechnet auf Grund des aktuellen Datums und des Wertes in der Eigenschaft `cutoffInterval` das gerade gültige Cutoff-Datum und liefert dieses in Form einer Referenz der Klasse *Notes DateTime* zurück.

```

class Replication {

    DateTime getCutoffDate();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                System.out.println ( db.getTitle() + " | " +
                    rep.getCutoffDate() );

                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.41: Beispiel Replication.CutoffDatum

An- und Abschalten des Cutoff-Mechanismus

Da sicher nicht bei jeder Datenbank alte Dokumente automatisch verschwinden sollen, gibt es selbstverständlich auch eine Eigenschaft, mit der man den Cutoff-Mechanismus einer Datenbank komplett an- bzw. ausschalten kann. Gesteuert wird dies in Java über die Eigenschaft `CutoffDelete` sowie mit den Funktionen `setCutoffDelete` und `isCutoffDelete`. Wenn die Eigenschaft den booleschen Wert *true* besitzt, werden veraltete Dokumente automatisch gelöscht, bei dem Wert *false* nicht.

```
class Replication {

    boolean isCutoffDelete();
    void setCutoffDelete( boolean flag );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                System.out.print ( db.getTitle() + " | " );
                if ( rep.isCutoffDelete() ) {
                    System.out.println ( "Cutoff Enable" );
                } else {
                    System.out.println ( "Cutoff Disenable" );
                };

                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.42: Beispiel Replication.CutoffDelete

4.2.8 Die Eigenschaft Abstract

Eine Eigenschaft, die steuert, ob lange Dokumente aufgeteilt oder Attachments aus den Dokumenten entfernt werden sollen, ist die Eigenschaft `abstract`. Zugegriffen wird auf diese boolesche Eigenschaft mit den Funktionen `getAbstract` und `setAbstract` der Klasse *Replication*.

```
class Replication {

    boolean isAbstract();
    void setAbstract(boolean bValue);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                Replication rep = db.getReplicationInfo();

                System.out.print ( db.getTitle() + " | " );
                if ( rep.isAbstract() ) {
                    System.out.println ( "Abstract True" );
                } else {
                    System.out.println ( "Abstract False" );
                };

                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.43: Beispiel Replication.Abstract

4.2.9 Die Eigenschaften der Datenbankvorlage

Jede Datenbank kann unter Notes eine Datenbankschablone darstellen. Sie kann somit als Vorlage für andere Datenbanken dienen. Dadurch werden Änderungen an ihren Gestaltungselementen ggf. an die anderen Datenbanken übertragen. Es kann also interessant sein, zu wissen, wie der Name einer Datenbankschablone ist oder wie die Schablone heißt, auf der die aktuelle Datenbank basiert.

Der Name einer Datenbankschablone

Jede Datenbankschablone besitzt einen speziellen Schablonennamen. Dieser Name ist unabhängig von dem Datenbanktitel oder Dateinamen. Er wird in der Eigenschaft `TemplateName` der Klasse *Database* abgelegt und kann durch die Funktion `getTemplateName` ausgelesen werden.

```
class Database {  
    String getTemplateName();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            DbDirectory dd = se.getDbDirectory(null);  
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );  
            while ( db != null ) {  
                db.open();  
                System.out.println ( db.getTitle() + " | " +  
                    db.getTemplateName() );  
                db.close();  
                db = dd.getNextDatabase();  
            };  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.44: *Beispiel Database.TemplateName*

Der Name der Datenbankschablone, auf der die Datenbank basiert

Der Name der Datenbankschablone, auf der die aktuelle Datenbank basiert, befindet sich in der Eigenschaft `DesignTemplateName` der Klasse *Database*. Das Auslesen dieser Eigenschaft erfolgt durch die Funktion `getDesignTemplateName`. Syntax:

```
class Database {

    String getDesignTemplateName();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getTitle() + " | " +
                    db.getDesignTemplateName() );

                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.45: *Beispiel Database.DesignTemplateName*

4.2.10 Das Arbeiten mit Datenbankkategorien

Wenn sich das Notes System erwartungsgemäß großer Beliebtheit erfreut, wird die Anzahl der Notes Datenbanken stetig steigen. Irgendwann dürfte es daher schwer werden, den Überblick bei den Datenbanken zu behalten. Deshalb gibt es unter Notes die Möglichkeit, jede Datenbank einer oder mehreren Kategorien zuzuordnen. Diese Kategorien werden dann z. B. genutzt, wenn der Datenbankkatalog erstellt wird. Der Datenbankkatalog ist eine Notes Datenbank mit einem Verzeichnis aller auf dem jeweiligen Server befindlichen Notes Datenbanken.

Der Zugriff auf die Kategorien einer Datenbank erfolgt unter Java mit der Eigenschaft *Categories* und den Funktionen *getCategories* oder *setCategories* der Klasse *Database*.

```
Class Database {  
    String getCategories();  
    void setCategories(String categories);  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            DbDirectory dd = se.getDbDirectory(null);  
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );  
            while ( db != null ) {  
                db.open();  
                System.out.println ( db.getTitle() + " | " +  
                    db.getCategories() );  
                db.close();  
                db = dd.getNextDatabase();  
            };  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.46: *Beispiel Database.Categories*

4.2.11 Datenbankgrößen

Der Speicherbedarf einer Datenbank, ein immer leidiges Thema, wird in Java durch verschiedene Eigenschaften der Klasse *Database* überwacht bzw. gesteuert.

Der aktuelle Speicherplatzbedarf einer Datenbank

Um den aktuellen Speicherplatzbedarf einer Datenbank zu überwachen, gibt es die Eigenschaft *Size* der Klasse *Database*. In dieser Eigenschaft findet man den aktuellen Bedarf der Datenbank in Byte. Ausgelesen werden kann diese Zahl mit der Funktion *getSize*.

```

class Database {

    double getSize();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getTitle() + " | " +
                    db.getSize() );

                db.close();
                db = dd.getNextDatabase();

            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.47: Beispiel Database.size

Die maximale Datenbankgröße Quota

Da – wenn man nicht haushaltet – der Speicherplatzbedarf erfahrungsmäßig immer ins Unermessliche wächst, bietet Notes eine Größenbeschränkung für Notes Datenbanken an. Ab der Notes Version 5 wird dies unter Java durch die Eigenschaft *SizeQuota* und den Funktionen *getSizeQuota* und *setSizeQuota* der Klasse *Database* gemacht. Die Eigenschaft beinhaltet die Größenbeschränkung der Datenbank in Kbyte.

```

class Database {

    int getSizeQuota();
    void setSizeQuota(int quota);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

```

```

try {
    Session se = getSession();
    DbDirectory dd = se.getDbDirectory(null);
    Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
    while ( db != null ) {

        db.open();
        System.out.println ( db.getTitle() + " | " +
            db. getSizeQuota() );

        db.close();
        db = dd.getNextDatabase();

    };

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.48: Beispiel Database.sizeQuota

Die maximale Datenbankgröße

Bis zur Version 5 wurde über die Eigenschaft `maxSize` die maximale Größe des Speicherbedarfs einer Notes Datenbank angegeben. Das Auslesen dieser Eigenschaft erfolgt mit der Funktion `getMaxSize` der Klasse *Database*. Dabei wird die Begrenzung in Kbyte zurückgegeben.

```

class Database {

    double getMaxSize();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getTitle() + " | " +
                    db.getMaxSize() );

            };

        };

    };

};

```

```

        db.close();
        db = dd.getNextDatabase();

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

    };
};

```

Listing 4.49: Beispiel Database.MaxSize

Der ausgenutzte Speicherplatz

Durch verschiedene Schreiboperationen auf einer Datenbank kommt es in der Datenbankdatei auf der Festplatte schnell zur Fragmentierung. Das bedeutet, es wird durch die Datenbankdatei viel mehr Speicherplatz beansprucht, als tatsächlich für reale Daten genutzt wird. Bei der Administration eines Notes Servers ist das Verhältnis zwischen dem benötigten und dem verbrauchten Speicherplatz sehr interessant. Dieser Wert kann z. B. genutzt werden, wenn entschieden werden muss, ob eine Datenbank komprimiert werden soll. Damit auf diesen Wert unter Java zugegriffen werden kann, liefert die Klasse *Database* die Funktion *getPercentUsed*. Nach dem Aufruf dieser Funktion wird das Verhältnis zwischen benötigtem und gebrauchtem Speicherbedarf als ein Wert vom Datentyp *Double* zurückgeliefert.

```

class Database {

    double getPercentUsed();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( db.getTitle() + " | " +
                    db.getPercentUsed() );

                db.close();
                db = dd.getNextDatabase();
            }
        }
    }
}

```

```

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

    };
};

```

Listing 4.50: *Beispiel Database.PercentUsed*

4.2.12 Volltextindex

Um Dokumente in einer Notes Datenbank zu suchen, können in einer Datenbank alle Dokumente volltextindiziert werden. Das bedeutet, es wird eine Liste aller in den Dokumenten der Datenbank enthaltenen Wörter angelegt und ihre Fundstelle notiert. Wenn dann nach einem bestimmten Stichwort gesucht wird, muss nur dieser Index durchlaufen werden, um das Dokument zu finden. Da dieses Volltextindizieren einen nicht unerheblichen Aufwand bedeutet, erfolgt die Indizierung nicht direkt beim Einfügen oder Ändern eines Dokuments, sondern in regelmäßigen Zeitabschnitten.

Der Stand der Volltextindizierung

Ob eine Datenbank schon mal volltextindiziert wurde, kann unter Java mit der Eigenschaft `FTIndexed` ermittelt werden. Der boolesche Inhalt dieser Eigenschaft wird mit der Funktion `isFTIndex` ausgelesen. Der Wert *true* zeigt dabei an, dass die Datenbank einen Volltextindex besitzt.

```

class Database {

    boolean isFTIndexed();

}

```

Um erkennen zu können, auf welchem Stand sich der Volltextindex einer Datenbank befindet, kann die Eigenschaft `LastFTIndexes` der Klasse *Database* mit der Funktion `getLastFTIndexed` ausgelesen werden. Die Funktion liefert das Datum der letzten Volltextindizierung in einem Notes Data-Objekt zurück. Wenn die Datenbank noch nicht volltextindiziert wurde, liefert die Funktion die Konstante *null* zurück.

```

class Database {

    DateTime getLastFTIndexed();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

```



```
System.out.print ( db.getTitle() + " | " );  
if ( db.isMultiDbSearch() ) {  
    System.out.println ( "True" );  
} else {  
    System.out.println ( "False" );  
};  
  
db.close();  
dd = dd.getNextDatabase();  
  
};  
  
} catch( Exception e) {  
  
    e.printStackTrace();  
  
};  
  
};
```

Listing 4.52: Beispiel Database.MultiDbSearch

4.2.13 Sonstige Datenbankeigenschaften

Status für das verzögerte Schreiben

Um die Performance bei Schreiboperationen auf entfernte Datenbanken zu steigern, bietet Notes die Technik des verzögerten Schreibens. Wenn das verzögerte Schreiben für eine Datenbank aktiviert ist, werden Änderungen an Dokumenten zuerst auf dem lokalen Computer zwischengespeichert. Die Änderung wird dann zu einem späteren Zeitpunkt an den Server übertragen. Dieses Verhalten steigert die Performance, birgt allerdings gewisse Risiken eines Crashes des lokalen Computers und damit eines Datenverlusts. Das Aktivieren dieses Verhaltens einer Datenbank wird mit der Eigenschaft `DelayUpdates` der Klasse `Database` und der Funktion `isDelayUpdates` zum Lesen sowie der Funktion `setDelayUpdate` zum Schreiben dieser Eigenschaft gesteuert. Wenn die Eigenschaft den booleschen Wert `true` beinhaltet, werden Änderungen zuerst lokal zwischengespeichert. Bei dem Wert `false` für diese Eigenschaft erfolgt kein Zwischenspeichern; die Änderungen werden sofort zum Server übertragen.

```
class Database {
    boolean isDelayUpdates();
    void setDelayUpdates(boolean delay);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
```



```

        db.open();
        System.out.print ( db.getTitle() + " | " );
        if ( db.getFolderReferencesEnabled () ) {
            System.out.println ( "True" );
        } else {
            System.out.println ( "False" );
        };

        db.close();
        db = dd.getNextDatabase();

    };

} catch(Exception e) {

    e.printStackTrace();

};

};

};

```

Listing 4.54: Beispiel Database.FolderReferencesEnabled

Der Datenbankmanager

Für jede Notes Datenbank sollte es einen oder mehrere Manager geben. Die Manager betreuen die Datenbank. Um auf einfachem Wege die Liste aller Administratoren einer Datenbank zu ermitteln, wird die Funktion `getManagers` zum Auslesen der Eigenschaft `managers` in der Klasse *Database* angeboten. Diese Funktion liefert als Rückgabewert eine Liste der Manager der aktuellen Datenbank in einem Vektor.

```

class Database {

    java.util.Vector getManagers();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dd = se.getDbDirectory(null);
            Database db = dd.getFirstDatabase(DbDirectory.DATABASE );
            while ( db != null ) {

                db.open();
                System.out.println ( "Manager von " + db.getTitle() );

                Vector vec = db.getManagers();
                Enumeration enum = vec.elements();
                while ( enum.hasMoreElements() ) {
                    System.out.println ( "    " + enum.nextElement() );
                }
            }
        }
    }
}

```

```

        };

        db = dd.getNextDatabase();

    };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};
};

```

Listing 4.55: Beispiel Database.Managers

Die übergeordnete Notes Session

Jede Instanz der Klasse *Database* wird durch eine Instanz der Klasse *Session* erstellt. In der Eigenschaft *parent* der Klasse *Database* wird die Referenz auf die Session zwischengespeichert und kann mit der Funktion *getParent* ausgelesen werden.

```

class Database {

    Session getParent();

}

```

4.3 Dokumente

4.3.1 Einleitung

Der Zugriff auf die Dokumente in einer Datenbank erfolgt unter Java in Notes mit der Klasse *Document*. Der folgende Abschnitt soll zeigen, wie man Dokumente unter Notes öffnen, anlegen, ändern und verschieben oder anderweitig umwandeln kann.

4.3.2 Zugriff auf Dokumente in einer Datenbank

Finden eines Dokuments anhand der Dokument-ID

Bevor mit den Dokumenten einer Datenbank gearbeitet werden kann, müssen diese gefunden werden. Die einfachste Aufgabe dürfte die Suche nach einem Dokument anhand seiner ID bzw. UNID sein.

Um ein Dokument anhand seiner ID zu ermitteln, wird von der Klasse *Database* die Methode *getDocumentByID* angeboten. Dieser Methode muss die entsprechende ID als Parameter übergeben werden. Nach dem Aufruf erhält man als Rückgabewert das entsprechende Dokument in Form einer Instanz der Klasse *Document*. Für den Fall, dass das Dokument nicht gefunden werden konnte, bekommt man als Rückgabewert den konstanten Wert *null*.

```
class Database {  
    Document getDocumentByID(String noteid);  
}
```

Die ID eines Dokuments wird durch die Klasse *Document* mit der Methode *getNodeID* ausgegeben.

```
class Document {  
    String getNodeID();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            String id = "123123123123";  
  
            Document doc = db.getDocumentByID ( id );  
  
            if ( doc != null ) {  
                System.out.println ( "Dokument gefunden" );  
            } else {  
                System.out.println ( "Dokument nicht gefunden" );  
            };  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.56: Beispiel Database.getDocumentByID

```
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col = db.getAllDocuments();
```



```

        Database db = ac.getCurrentDatabase();
        String id = "123123123123";

        Document doc = db.getDocumentByUNID ( id );

        if ( doc != null ) {
            System.out.println ( "Dokument gefunden" );
        } else {
            System.out.println ( "Dokument nicht gefunden" );
        };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};

```

Listing 4.58: Beispiel Database.getDocumentByUNID

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getFirstDocument();
            while ( doc != null ) {
                System.out.println ( doc.UniversalID() );
                doc = col.getNextDocument();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.59: Beispiel Document.getUniversalID

Alle Dokumente einer Datenbank

Fast genauso einfach wie ein Dokument ermittelt werden kann, können alle Dokumente einer Datenbank angesprochen werden. Hierfür muss nur die Methode `getAllDocuments` der Klasse *Database* aufgerufen werden. Diese Methode liefert alle Doku-

mente einer Datenbank als Dokumentensammlung zurück. Der Rückgabewert ist eine Instanz der Klasse *DocumentCollection*. Die Funktionalität dieser Klasse wird im folgenden Abschnitt erklärt.

```
class Database {  
    DocumentCollection getAllDocuments;  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col = db.getAllDocuments();  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.60: *Beispiel Database.AllDocuments*

4.3.3 Die Dokumentensammlung

Wenn mehrere Dokumente bearbeitet werden sollen, wird meist die Klasse *DocumentCollection* verwendet. Diese Klasse bietet verschiedene Methoden und Eigenschaften, auf eine Sammlung von Dokumenten zuzugreifen oder diese zu verändern.

Durchlaufen aller Dokumente einer Dokumentensammlung

Vorwärts durchlaufen

Nachdem eine Dokumentensammlung zurückgegeben wurde, steht man vor der Aufgabe, die einzelnen Dokumente anzusprechen. Eine Möglichkeit, dies zu tun, ist das Nacheinander-Durchwandern aller Dokumente in der Sammlung.

Um den Anfang zu finden, stellt die Klasse *DocumentCollection* die parameterlose Methode *getFirstDocument* zur Verfügung, mit der das erste Dokument ermittelt werden kann. Als Rückgabewert wird die Referenz auf das erste Dokument als Instanz der Klasse *Document* geliefert. Sollte kein Dokument vorhanden sein, wird der Wert null geliefert.

```
class DocumentCollection {  
    Document getFirstDocument();  
}
```

Nachdem das erste Dokument gefunden ist, können die folgenden Dokumente mit der Methode `getNextDocument` durchwandert werden. Dabei kann der Methode als optionalen Parameter angegeben werden, von welchem Dokument der Nachfolger ermittelt werden soll. Wenn kein Parameter angegeben wird, wird das Nachfolgedokument des zuletzt ermittelten Dokuments der Sammlung zurückgeliefert. Der hier mögliche Parameter sollte nur in Ausnahmefällen verwendet werden, da darunter einerseits die Performance leidet, andererseits wird dieser Parameter bei Zugriffen über IIOP von Notes nicht akzeptiert. Falls ein weiteres Dokument vorhanden ist, erhält man als Rückgabewert eine Instanz der Klasse *Document*, ansonsten den Wert null.

```
class DocumentCollection {  
    Document getNextDocument();  
    Document getNextDocument(Document doc);  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col = db.getAllDocuments();  
  
            Document doc = col.getFirstDocument();  
            while ( doc != null ) {  
                System.out.println ( doc.UniversalID() );  
                doc = col.getNextDocument();  
            };  
        } catch (Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.61: *Beispiel DocumentCollection.nextDocument*

Rückwärts durchlaufen

Mit den gezeigten Methoden durchläuft man die Dokumentensammlung vorwärts. Um diese z. B. auch rückwärts durchlaufen zu können, gibt es unter Notes die Methoden `getLastDocument` und `getPrevDocument`. Die Methode `getLastDocument` liefert dabei das letzte Dokument in der Sammlung in Form einer Instanz der Klasse *Document*.

```
class DocumentCollection {
    Document getLastDocument();
}
```

Um beim Rückwärtsdurchlaufen wie entsprechend bei `getNextDocument` das vorherige Dokument zu bekommen, ist die Methode `getPrevDocument` verfügbar. In der Parameterstruktur und der Art des Rückgabewerts sind beide Methoden identisch. Auch hier kann optional ein Dokument angegeben werden, dessen Vorgänger ermittelt werden soll. Die Einschränkung bzgl. IIOP gilt hier genauso wie bei der `getNextDocument`-Methode. Auch die Performance-Auswirkungen bezüglich des Parameters sollten hier bedacht werden.

```
class DocumentCollection {
    Document getPrevDocument();
    Document getPrevDocument(Document doc);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getLastDocument();
            while ( doc != null ) {
                System.out.println ( doc.UniversalID() );
                doc = col.getPrevDocument();
            };

        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.62: Beispiel *DocumentCollection.getPrevDocument*

Zugriff auf einzelne Dokumente der Sammlung

Da nicht immer alle Dokumente einer Sammlung durchwandert werden sollen, stellt Notes eine Methode mit dem Namen `getNthDocument` zur Verfügung. Mit dieser wird die Möglichkeit geboten, auf ein bestimmtes Dokument in der Sammlung zuzugreifen. Dabei wird als Parameter die Position des gesuchten Dokuments in der Sammlung erwartet. Als Rückgabewert bekommt man eine Referenz auf das Dokument in Form einer Instanz der Klasse *Document*.

```
class DocumentCollection {  
    Document getNthDocument(int n);  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col = db.getAllDocuments();  
  
            Document doc = col.getNthDocument(1);  
        } catch (Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.63: *Beispiel DocumentCollection.getNthDocument*

Ein Dokument aus der Sammlung entfernen

Wenn man ein Dokument in einer Dokumentensammlung gefunden hat, kann es sein, dass man dieses Dokument aus der Sammlung entfernen möchte. Dazu steht die Methode `deleteDocument` der Klasse *DocumentCollection* zu Verfügung. Durch den Aufruf dieser Methode mit der Referenz des Dokuments als Parameter wird das Dokument aus der Sammlung entfernt. Allerdings wird das Dokument nicht physikalisch gelöscht.

```
class DocumentCollection {  
    void deleteDocument(Document doc);  
}
```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getNthDocument(1);
            col.deleteDocument(doc) ;

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.64: *Beispiel DocumentCollection.deleteDocument*

Ein Dokument zur Sammlung hinzufügen

Es wurde erklärt, wie ein Dokument aus der Sammlung aussortiert werden kann. Natürlich funktioniert dieser Vorgang auch andersherum. Damit ein Dokument in einer Sammlung eingefügt werden kann, wird als Parameter nur die entsprechende Instanz der Klasse *Document* benötigt. Nach dem Aufruf der Methode mit der Referenz auf das Dokument als Parameter veranlasst man das Aufnehmen dieses Dokuments in die Sammlung.

```

class DocumentCollection {

    void addDocument(Document doc);
    void addDocument(Document doc, boolean checkDups);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();
            Document doc = col.getNthDocument(1);
            col.deleteDocument(doc) ;
            col.addDocument ( doc );

        }

    };
};

```

```

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.65: *Beispiel DocumentCollection.addDocument*

Löschen aller Dokumente der Sammlung

Neben dem Löschen einzelner Dokumente oder Datenbanken bietet Notes die Möglichkeit, alle Dokumente einer Auswahl zu entfernen. Hierzu muss die Methode `removeAll` der Klasse *DocumentCollection* aufgerufen werden. Als Parameter wird ein boolescher Ausdruck erwartet, der spezifiziert, ob ein Dokument auch dann gelöscht werden soll, wenn es gerade von einem anderen Anwender bearbeitet wird. Bei dem Wert *true* für diesen Parameter erfolgt das Löschen auch, wenn das Dokument gerade bearbeitet wird. Sollte der Wert *false* angegeben werden, so wird ein Dokument auch gelöscht, wenn es gerade zum Editieren geöffnet worden ist.

```

class DocumentCollection {

    void removeAll(boolean force);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getNthDocument(1);
            col.removeAll(true);

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.66: *Beispiel DocumentCollection.removeAll*

Alle Dokumente einer Auswahl zu einem Ordner hinzufügen

Um alle Dokumente einer Auswahl einem bestimmten Ordner hinzuzufügen, wird die Methode `putAllInFolder` in der Klasse *DocumentCollection* angeboten. Diese Methode erwartet als ersten Parameter den Namen des Ordners und als zweiten optionalen Parameter einen booleschen Wert, der angibt, ob der angegebene Ordner im Bedarfsfall angelegt werden soll.

```
class DocumentCollection {

    void putAllInFolder(String folderName);

    void putAllInFolder(String folderName,
        boolean createonfail);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getNthDocument(1);
            col.putAllInFolder("Uebersicht",true);

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.67: *Beispiel DocumentCollection.putAllInFolder*

Alle Dokumente einer Auswahl aus einem Ordner entfernen

Natürlich gibt es, wenn man alle Dokumente einer Auswahl zu einem Ordner hinzufügen kann, auch die entsprechende komplementäre Methode in der Klasse *DocumentCollection*, mit der man alle Dokumente einer Auswahl aus einem Ordner wieder entfernen kann. Diese Methode erwartet als Parameter den Namen des entsprechenden Ordners.

```

class DocumentCollection {

    void removeAllFromFolder(String foldername);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getNthDocument(1);
            col.removeAllFromFolder("Uebersicht");

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.68: *Beispiel DocumentCollection.removeAllFromFolder*

Sammeländerungen in der Dokumentensammlung

Um in einem Feld in allen Dokumenten einer Dokumentenauswahl denselben Wert zu setzen, steht die Methode `stampAll` der Klasse *DocumentCollection* zur Verfügung. Mit dieser Methode kann das Setzen der Felder erfolgen, ohne dass alle Dokumente in einer Schleife durchlaufen werden müssen und jeweils eine Referenz der Klasse *Document* erzeugt werden muss. Die Methode erwartet als Erstes den Feldnamen und als zweiten Parameter den Wert, der gesetzt werden soll.

```

class DocumentCollection {

    void stampAll(String itemname, Object value);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

```

```

        DocumentCollection col = db.getAllDocuments();

        col.stampAll ( "Status", "Erledigt" );

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};

```

Listing 4.69: Beispiel DocumentCollection.stampAll

Änderungsmarkierung der Dokumente einer Sammlung

In Notes ist es möglich, für alle Dokumente herauszufinden, ob sie seit der letzten Änderung gelesen worden sind. Um diese Markierung für alle Dokumente einer Dokumentenauswahl auf „ungelesen“ zu setzen, gibt es die parameterlose Methode `updateAll`. Nach dem Aufruf dieser Methode werden alle entsprechenden Dokumente als ungelesen markiert.

```

class DocumentCollection {

    void updateAll();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            col.updateAll ();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.70: Beispiel DocumentCollection.updateAll

4.3.4 Dokumente suchen

Damit sich das Sprichwort „Wer sucht, der findet“ auch erfüllen kann, bietet Notes verschiedene Möglichkeiten, nach Dokumenten zu suchen.

Suche über eine Suchformel

Eine einfache Suche lässt sich mit der Methode `search` in der Klasse *Database* realisieren. Diese Methode erwartet das Suchkriterium in Form einer Notes Formel. Des Weiteren können optional ein Datum und ein numerischer Wert angegeben werden. Es werden dann nur Dokumente zurückgeliefert, die nach dem angegebenen Datum erstellt oder geändert wurden. Die Anzahl der zurückgegebenen Dokumente wird durch den numerischen Parameter beschränkt.

```
class Database {  
  
    DocumentCollection search(String formula);  
  
    DocumentCollection search(String formula, DateTime dt);  
  
    DocumentCollection search(String formula,  
        DateTime dt, int max);  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col =  
                db.search ( "nachname=\"meier\"");  
  
            Document doc = col.getFirstDocument();  
            while ( doc != null ) {  
  
                // Bearbeiten des Dokuments  
  
                doc = col.getNextDocument();  
            }  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        }  
  
    };  
  
};
```

Listing 4.71: Beispiel Database.search

Volltextsuche in einer Datenbank oder Dokumentenauswahl

Die Suche nach einem bestimmten Wort in einem Dokument ist ein sehr häufiger Wunsch des Anwenders. Für diese Suche steht die Methode `FTSearch` in der Klasse *Database* oder in der Klasse *DocumentCollection* zur Verfügung.

```
class Database {

    DocumentCollection FTSearch(String query);

    DocumentCollection FTSearch(String query, int max);

    DocumentCollection FTSearch(String query, int max,
        int sortopt, int otheropt );

}

class DocumentCollection {

    public void FTSearch(String query);
    public void FTSearch(String query, int maxdocs);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col =
                db.FTSearch ( "meier");

            Document doc = col.getFirstDocument();
            while ( doc != null ) {

                // Bearbeiten des Dokuments

                doc = col.getNextDocument();

            }

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.72: Beispiel Database.FTSearch und DocumentCollection.FTSearch

Suche nach unbehandelten Dokumenten

Innerhalb eines Agents gibt es eine Besonderheit bei der Durchsuchung von Dokumenten in der Datenbank. Notes bietet eine besondere Technik an, mit der gleichzeitig überprüft werden kann, ob ein Dokument seit der letzten Bearbeitung durch einen Agent angelegt oder verändert wurde. Bei einer Suche kann nun bestimmt werden, dass nur die Dokumente angezeigt werden sollen, die seit ihrer Erstellung oder letzten Änderung nicht durch den Agent bearbeitet wurden. Dazu werden die Methoden `unprocessedFTSearch` oder `unprocessedSearch` eingesetzt.

```
class DocumentCollection {

    public DocumentCollection unprocessedFTSearch(
        String query, int maxdocs)
    public DocumentCollection unprocessedFTSearch(
        String query, int maxdocs, int sortopt, int otheropt)

}

class Database {

    public DocumentCollection unprocessedSearch(
        String formula, DateTime limit, int maxdocs )

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col =
                db.unprocessedFTSearch ( "meier");

            Document doc = col.getFirstDocument();
            while ( doc != null ) {

                // Bearbeiten des Dokuments

                doc = col.getNextDocument();

            }

        };

    };
};
```

Listing 4.73: *Beispiel Database.unprocessedSearch*

4.3.5 Anlegen eines Dokuments

Um ein Dokument anzulegen, wird eine gültige Referenz auf die Datenbank benötigt, in der das Dokument angelegt werden soll. Durch den Aufruf der Methode `createDocument` in der Klasse `Database` kann ein beliebiges Notes Dokument angelegt werden. Nach dem Aufruf der Methode erhält man als Rückgabewert eine Referenz auf das neue Dokument. Um dieses Dokument dauerhaft zu speichern, muss anschließend die Methode `save` der Klasse `Document` angewendet werden.

```
class Database {  
    public Document createDocument()  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
  
            Document doc = db.createDocument ();  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.74: *Beispiel Database.createDocument*

4.3.6 Speichern eines Dokuments

Die bereits erwähnte Methode `save` der Klasse `Document` dient zum Speichern von Dokumenten. Wenn Änderungen an einem Dokument durchgeführt wurden, werden sie erst durch den Aufruf dieser Funktion dauerhaft gespeichert.

Der Methode `save` kann beim Aufruf wahlweise einen bis drei oder keinen Parameter übergeben werden. Als erster Parameter kann ein boolescher Ausdruck übergeben werden, der bestimmt, ob gleichzeitige Änderungen anderer Anwender beim Speichern überschrieben werden sollen. Wenn der Parameter den Wert `true` hat, wird die Version des Dokuments gespeichert, auch wenn inzwischen eine Änderung an demselben Dokument von anderer Stelle stattgefunden hat. Sollte dieser Parameter den Wert `false` besitzen, wird keine Änderung überschrieben und die Speicherung des Dokuments kann schiefgehen. Sollte der erste Parameter den Wert `false` besitzen, so kann durch den optionalen zweiten Parameter bestimmt werden,

ob die neue Version des Dokuments als Antwortdokument gespeichert werden soll. Bei dem Wert *true* wird die aktuelle Version als Antwortdokument gespeichert, bei dem Wert *false* wird versucht, das Originaldokument zu überschreiben. Als letzter optionaler Parameter kann ein boolescher Wert zu Bestimmung der Lesemarkierung übergeben werden. Dieser Parameter veranlasst – wenn der Wert *true* übergeben wird –, dass das neue Dokument als bereits gelesen markiert wird. Bei der Übergabe des Wertes *false* wird das Dokument als noch nicht gelesen markiert. Nach dem Aufruf der Methode bekommt der Anwender als Rückgabewert einen booleschen Wert, der angibt, ob das Speichern erfolgreich war.

```
class Document {  
  
    public boolean save()  
  
    public boolean save(boolean force)  
  
    public boolean save(boolean force, boolean makesresponse)  
  
    public boolean save(boolean force, boolean makesresponse, boolean markread)  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
  
            Document doc = db.createDocument ();  
            doc.save();  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        };  
  
    };  
};
```

Listing 4.75: *Beispiel Document.save*

4.3.7 Löschen eines Dokuments

Nachdem ein Dokument angelegt wurde, soll es nun wieder gelöscht werden. Dafür stellt die Klasse *Document* die Methode *remove* zur Verfügung. Die Parameter dieser Methode bestimmen, ob ein Dokument auch dann entfernt werden soll, wenn es gerade von einem anderen Anwender bearbeitet wird. Wenn der Parameter den Wert *true* besitzt, wird auch ein gerade in Bearbeitung befindliches Dokument entfernt. Ein Entfernen von geöffneten Dokumenten ist beim Wert *false* nicht möglich.

Die Erfolgsmeldung der Löschoperation erhält man durch den booleschen Rückgabewert der Methode.

```
class Document {
    public boolean remove(boolean force)
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DatabaseCollection col = db.getAllDocuments();

            Document doc = col.getFirstDocument ();
            doc.remove(true);
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.76: *Beispiel Document.remove*

4.3.8 Kopieren eines Dokuments

Ein Dokument lässt sich ganz einfach durch die Methode `copyToDatabase` der Klasse *Document* kopieren. Die Methode erwartet lediglich als Parameter den Datenbanknamen der Zieldatenbank. Sollte statt eines Namens der Wert *null* angegeben werden, so wird die Kopie in die aktuelle Datenbank geschrieben. Die Methode liefert die Referenz auf das neue Dokument zurück.

```
class Document {
    public Document copyToDatabase(Database db)
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
```

```

AgentContext ac = se.getAgentContext();
Database dbdest = ac.getCurrentDatabase();
Database dbsource = se.getDatabase ( null, "names.nsf" );
DatabaseCollection col = dbsource.getAllDocuments();

Document doc = col.getFirstDocument ( );
while ( doc != null ) {

    dbsource.copyToDatabase ( dbdest );
    doc = col.getNextDocument();

}

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.77: *Beispiel Document.copyToDatabase*

4.3.9 Versenden eines Dokuments

Natürlich darf eine Funktion zum Versenden eines Dokuments in einem System wie Notes nicht fehlen. Hierzu wird die Methode `send` der Klasse *Document* zur Verfügung gestellt. Diese Methode kann mit sechs verschiedenen Parameteranordnungen aufgerufen werden. Zuerst einmal kann die Methode ohne Parameter aufgerufen werden. Des Weiteren können ein oder mehrere Empfänger als Parameter übergeben werden. Dabei wird ein Empfänger in einem Parameter vom Typ `String` und eine Empfängerliste in einem Parameter vom Typ Vektor von `Strings` erwartet. Der andere optionale Parameter ist der boolesche Wert, der angibt, ob die Maske beim Versenden mit dem Dokument versandt wird. Wenn dieser Parameter den Wert *true* hat, wird die Maske versendet, bei dem Wert *false* erfolgt kein Versand.

```

class Document {

    public void send() ;
    public void send(String recipient) ;
    public void send(java.util.Vector recipients);
    public void send(boolean attachform);
    public void send(boolean attachform, String recipient);
    public void send(boolean attachform,
        java.util.Vector recipients);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

```



```

        }

        } catch(Exception e) {
            e.printStackTrace();
        };

    };
};

```

Listing 4.79: *Beispiel Document.SaveMessageOnSend*

4.3.10 Profildokumente

Ein typischer Anwender ist ein Gewohnheitsmensch. Er will in einer Anwendung am liebsten seine eigene Arbeitsumgebung einstellen können. Das bedeutet, es müssen z. B. benutzerspezifische Daten für eine Datenbankanwendung in einer Notes Datenbank gespeichert werden. Dafür gibt es in Notes so genannte Profildokumente.

Ein Profildokument kann für jeden Anwender in einer Datenbank angelegt werden. Dabei besitzt ein Profildokument immer einen Namen und einen Schlüssel. Der Name bezeichnet den Typ des Dokuments und der Schlüssel dient z. B. zur Identifikation des Anwenders. Profildokumente können nicht nur benutzerspezifisch, sondern auch abhängig von beliebigen anderen Schlüsseln sein.

Erstellen eines Profildokuments

Mit der Methode `getProfileDocument` der Klasse *Database* kann ein Profildokument angefordert werden. Diese Methode erwartet den Namen des Profildokuments und den Schlüssel, z. B. den Namen des Anwenders. Nach dem Aufruf liefert die Methode eine Instanz der Klasse *Document* als Referenz auf das gewünschte Profildokument zurück. Sollte mit dem angegebenen Namen und dem Schlüssel kein Profildokument vorliegen, wird dieses bei Aufruf der Methode automatisch angelegt.

```

class Database {

    public Document getProfileDocument(
        String profil, String key );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.getProfileDocument (
                "Farbeinstellung", se.getUserName() );

```

```

        // Daten lesen und aendern im Profildokument

        doc.save();

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};
};

```

Listing 4.80: *Beispiel Database.getProfileDocument*

4.3.11 Ermitteln aller Profildokumente mit einem bestimmten Profilnamen

In einigen Fällen kann es notwendig sein, dass alle Profildokumente mit einem bestimmten Profilnamen ermittelt werden müssen. Dafür gibt es in der Klasse *Database* die Methode *getProfileDocCollection*, die eine Dokumentensammlung mit allen entsprechenden Profildokumenten zurückgibt.

```

class Database {

    public DocumentCollection getProfileDocCollection(
        String profil );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            DocumentCollection col = db.getProfileDocCollection (
                "Farbeinstellung" );

            Document doc = col.getFirstDocument();
            while ( doc != null ) {

                // Bearbeiten des Profildokuments

                doc = col.getNextDocument();

            };

        } catch(Exception e) {

```



```
        e.printStackTrace();
    };
};
};
```

Listing 4.81: Beispiel Database.getProfileDocCollection

4.4 Die Zugriffskontrollliste

Die Zugriffsregeln für eine Datenbank werden in der Zugriffskontrollliste (Access Control List/ACL) verwaltet. Um diese Zugriffskontrollliste zu bearbeiten, wird in Java die Klasse *ACL* eingesetzt.

4.4.1 Zugriff auf die Zugriffskontrollliste einer Datenbank

Um die Zugriffskontrollliste einer Datenbank zu bearbeiten, wird eine entsprechende Referenz der Klasse *ACL* für die jeweilige Datenbank benötigt. An diese Referenz gelangt man durch die Eigenschaft *ACL* der Klasse *Database*. Durch den Aufruf von *getACL* erhält man eine Referenz.

```
class Database {
    public ACL getACL();
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();

        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.82: Beispiel Database.getACL

4.4.2 Bearbeiten der Einträge der Zugriffskontrollliste

Die Zugriffskontrollliste besteht aus Einträgen für verschiedene Notes Anwender. Für jeden Eintrag ist definiert, was erlaubt oder verboten ist.

Für den Zugriff auf die Einträge stehen in der Klasse *ACL* drei Methoden zur Verfügung: die Methoden *getEntry*, *getFirstEntry* und *getNextEntry*.

Die Methoden *getFirstEntry* und *getNextEntry*

Die Methoden *getFirstEntry* und *getNextEntry* dienen zum Auslesen aller Einträge der Liste. Dazu wird zuerst die parameterlose Methode *getFirstEntry* genutzt, um den ersten Eintrag zu der Liste zu ermitteln. Als Rückgabewert liefert die Methode eine Referenz auf den Eintrag.

```
class ACL {

    public ACLEntry getFirstEntry();

}
```

Um den folgenden Eintrag aus der Liste zu erhalten, kann die Methode *getNextEntry* eingesetzt werden. Dabei gibt es zwei mögliche Aufrufformen für die Methode. Diese unterscheiden sich in ihren Parametern. Die erste Aufrufform erwartet keine Parameter. Sie liefert den Eintrag zurück, der dem letzten gelieferten Eintrag beim letzten Aufruf der Methode *getFirstEntry* oder *getNextEntry* folgt.

```
class ACL {

    public ACLEntry getNextEntry();

}
```

Bei der zweiten Aufrufform erwartet die Methode als Parameter eine Referenz auf einen Eintrag in der Zugriffskontrollliste. Nach dem Aufruf wird eine Referenz auf den Eintrag zurückgegeben, der dem als Parameter angegebenen Eintrag folgt.

```
class ACL {

    ACLEntry getNextEntry(ACLEntry entry);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();

            ACLEntry entry = acl.getFirstEntry();
```

```

        while ( entry != null ) {
            System.out.println ( entry.getName() );
            entry = acl.getNextEntry();
        }
    } catch(Exception e) {
        e.printStackTrace();
    };
};
};

```

Listing 4.83: Beispiel *ACL.getFirstEntry* und *ACL.getNextEntry*

Einzelne Einträge aus der Zugriffskontrollliste lesen

Die oben gezeigten Methoden sind geeignet, um alle Einträge der Zugriffskontrollliste zu durchwandern. Wenn aber ein einzelner bestimmter Eintrag gesucht wird, bietet sich die Methode *getEntry* der Klasse *ACL* an. Bei dem Aufruf dieser Methode kann ein Name angegeben werden. Es wird dann von der Methode eine Referenz auf den Eintrag zurückgeliefert, welcher für die angegebene Person, den Server oder die Gruppe steht.

```

class ACL {
    ACLEntry getEntry(String ename);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();

            ACLEntry entry = acl.getEntry ( "Mausman/world" );
            System.out.println ( entry.getLevel() );

        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
};

```

Listing 4.84: Beispiel *ACL.getEntry*

Einträge anlegen

Einträge in der Zugriffskontrollliste werden durch die Methode `createACLEntry` angelegt. Dabei muss der Name der Person, des Servers oder der Gruppe angegeben werden, für die/den der Eintrag Gültigkeit haben soll. Als zweiten Parameter muss das gewünschte Zugriffslevel angegeben werden. Nach dem Aufruf befindet sich der gewünschte Eintrag in der Zugriffskontrollliste. Nach dem Anlegen eines Eintrags muss die Methode `save` der Klasse `ACL` verwendet werden, um diese Änderung permanent zu speichern.

```
class ACL {

    ACLEntry createACLEntry(String name, int level)

}
```

Folgende Zugriffslevel können gesetzt werden:

Zugriffslevel	Beschreibung
ACL.LEVEL_NOACCESS	Kein Zugriff
ACL.LEVEL_DEPOSITOR	Einlieferer
ACL.LEVEL_READER	Leser
ACL.LEVEL_AUTHOR	Autor
ACL.LEVEL_EDITOR	Editor
ACL.LEVEL_DESIGNER	Entwickler
ACL.LEVEL_MANAGER	Manager

Tabelle 4.5: Die Zugriffslevel

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            acl.createACLEntry ( "Mausman/world", ACL.LEVEL_MANAGER );
            acl.save();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.85: Beispiel `ACL.createACLEntry`

Einträge löschen

Einträge können natürlich auch über die Javaklassen aus der Zugriffskontrollliste entfernt werden. Das Entfernen des Eintrags kann entweder mit der Methode `removeACLEntry` der Klasse `ACL` oder mit der Methode `remove` der Klasse `ACLEntry` erfolgen.

Die Methode `removeACLEntry` der Klasse `ACL`

Die Methode `removeACLEntry` erwartet den Namen der Person, des Servers oder der Gruppe, deren Eintrag aus der Zugriffskontrollliste entfernt werden soll.

Nach dem Entfernen eines Eintrags muss die Methode `save` der Klasse `ACL` verwendet werden, um diese Änderung permanent zu speichern.

```
class ACL {  
    removeACLEntry(String name)  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            ACL acl = db.getACL();  
            acl.removeACLEntry ( "Mausman/world");  
            acl.save();  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    };  
};
```

Listing 4.86: *Beispiel `ACL.removeACLEntry`*

Die Methode `remove` der Klasse `ACLEntry`

Eine weitere Möglichkeit, Einträge zu löschen, ist die Methode `remove` der Klasse `ACLEntry`. Diese Methode eignet sich dann, wenn eine Referenz auf den zu löschenden Eintrag vorliegt.

Nach dem Entfernen eines Eintrags muss die Methode `save` der Klasse `ACL` verwendet werden, um diese Änderung permanent zu speichern.

```

class ACLEntry {

    remove ()

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getEntry ( "Mausman/world" );
            entry.remove ( );
            acl.save();
        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.87: Beispiel ACLEntry.remove

Einträge ändern

Bisher wurde gezeigt, wie man Einträge anlegen, löschen oder suchen kann. Natürlich können die Einträge in der Zugriffskontrollliste auch geändert werden.

Um Änderungen an einem Eintrag vorzunehmen, müssen nur die Eigenschaften in der ermittelten Referenz geändert werden.

Nach der Änderung eines Eintrags muss die Methode `save` der Klasse `ACL` verwendet werden, um diese Änderung permanent zu speichern.

Ein Eintrag in der Zugriffskontrollliste bzw. die Klasse `ACLEntry` hat unter anderem folgende Eigenschaften:

Der Typ des Eintrags

Ein Eintrag in der Zugriffskontrollliste kann vom Typ Unbestimmt, Person, Personengruppe, Server, Servergruppe oder gemischte Gruppe sein. Dieser Typ kann über die Eigenschaft `UserType` in der Klasse `ACLEntry` ausgelesen oder gesetzt werden.

```

class ACLEntry {

    int getUserType();
    void setUserType(int tp);

}

```

Folgende Konstanten stehen dabei als Zugriffstypen zur Verfügung:

Konstante	Bedeutung
ACLEntry.TYPE_MIXED_GROUP	Gemischte Gruppe
ACLEntry.TYPE_PERSON	Person
ACLEntry.TYPE_PERSON_GROUP	Personengruppe
ACLEntry.TYPE_SERVER	Server
ACLEntry.TYPE_SERVER_GROUP	Servergruppe
ACLEntry.TYPE_UNSPECIFIED	Unbestimmt

Tabelle 4.6: Konstante für Zugriffslevel

Außer der Eigenschaft `UserType` gibt es noch drei weitere Eigenschaften, die den Typ des Eintrags bestimmen.

Mit den booleschen Eigenschaften `Person`, `Server` und `Group` kann ebenfalls der Typ des Eintrags ausgelesen und gesetzt werden.

```
class ACLEntry {
    boolean isPerson();
    void setPerson(boolean flag);
    boolean isServer();
    void setServer(boolean flag);
    boolean isGroup();
    void setGroup(boolean flag);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();

            ACLEntry entry = acl.getFirstEntry();
            while ( entry != nul ) {
                System.out.print ( acl.getName() );
                if ( entry.isPerson() ) {
                    System.out.print ( " | Person: ja | " );
                } else {
                    System.out.print ( " | Person: nein | " );
                }
            };
            if ( entry.isServer() ) {
                System.out.print ( " | Server: ja | " );
            } else {
                System.out.print ( " | Server: nein | " );
            };
        }
    }
}
```



```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getFirstEntry ();
            while ( entry != null ) {
                if ( acl.isAdminServer() )
                    System.out.println ( entry.getName() );
                entry = acl.getNextEntry();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.89: Beispiel ACLEntry.AdminServer

Das Zugriffslevel

Das bereits beim Anlegen eines Eintrags erwähnte Zugriffslevel kann über die Eigenschaft `Level` nachträglich für einen Eintrag geändert werden.

```

class ACLEntry {

    int getLevel();
    void setLevel(int level);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getFirstEntry ();
            while ( entry != null ) {
                System.out.println ( entry.getName() + " : " +
                    entry.getLevel());
                entry = acl.getNextEntry();
            };

        };

    };
};

```

```

    } catch(Exception e) {
        e.printStackTrace();
    };
};

```

Listing 4.90: Beispiel ACLEntry.Level

Der Name eines Eintrags

Der Name eines Eintrags, also der Name der Person, des Servers oder der Gruppe kann über die Eigenschaft Name verändert oder ausgelesen werden.

```

class ACLEntry {
    String getName();
    void setName(String name);
    void setName(Name n);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getFirstEntry ();
            while ( entry != null ) {
                if ( entry.getLevel() == ACL.LEVEL_MANAGER )
                    System.out.println ( entry.getName() );
                entry = acl.getNextEntry();
            };
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};

```

Listing 4.91: Beispiel ACLEntry.Name

Spezielle Rechte-Einstellungen

Die Aktionen, die durch einen Eintrag in der Zugriffskontrollliste erlaubt oder verboten werden, können durch verschiedene Eigenschaften spezifiziert werden.

Folgende Eigenschaften können dabei verwendet werden:

Eigenschaft	Beschreibung
CanCreateDocuments	Erlaubt das Erstellen von Dokumenten
CanCreateLSOrJavaAgent	Erlaubt das Schreiben von gemeinsamen Agents
CanCreatePersonalAgent	Erlaubt das Schreiben von persönlichen Agents
CanCreatePersonalFolder	Erlaubt das Erstellen von persönlichen Ordnern
CanCreateSharedFolder	Erlaubt das Erstellen von gemeinsamen Ordnern
CanDeleteDocuments	Erlaubt das Löschen von Dokumenten
PublicReader	Erlaubt das Lesen von öffentlichen Dokumenten
PublicWriter	Erlaubt das Schreiben von öffentlichen Dokumenten

Tabelle 4.7: Eigenschaften zur Steuerung der Berechtigungen

Teilweise werden diese Berechtigungen schon durch das Zugriffslevel erteilt. In einem solchen Fall liefert die Abfrage der Eigenschaft den Wert *true* zurück.

```
class ACLEntry {

    boolean isCanCreateDocuments();
    void setCanCreateDocuments(boolean flag);

    boolean isCanCreateLSOrJavaAgent();
    void setCanCreateLSOrJavaAgent(boolean flag);

    boolean isCanCreatePersonalAgent();
    void setCanCreatePersonalAgent(boolean flag);

    boolean isCanCreatePersonalFolder();
    void setCanCreatePersonalFolder(boolean flag);

    boolean isCanCreateSharedFolder();
    void setCanCreateSharedFolder(boolean flag);

    boolean isCanDeleteDocuments();
    void setCanDeleteDocuments(boolean flag);

    boolean isPublicReader();
    void setPublicReader(boolean flag);

    boolean isPublicWriter();
    void setPublicWriter(boolean flag);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {
```

```

Session se = getSession();
AgentContext ac = se.getAgentContext();
Database db = ac.getCurrentDatabase();
ACL acl = db.getACL();
ACLEntry entry = acl.getFirstEntry ();
while ( entry != null ) {
    System.out.println ( entry.getName() );
    if ( entry. isCanCreateDocuments() ) {
        System.out.println ( "    CreateDocuments: true" );
    } else {
        System.out.println ( "    CreateDocuments: false" );
    };
    entry = acl.getNextEntry();
};

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.92: Beispiel ACLEntry.CanCreateDocuments

4.4.3 Die Zugriffsrollen

Für jeden Eintrag in der Zugriffskontrollliste können in Notes Zugriffsrollen vergeben werden. Diese Zugriffsrollen können von einem Programmierer für eine Berechtigungsabfrage in den Gestaltungselementen in der Datenbank genutzt werden.

Eine Liste aller Rollen, die in der Datenbank vorkommen, wird in der Zugriffskontrollliste verwaltet. Wer welche Rolle in einer Datenbank besitzt, wird in den einzelnen Einträgen in der Zugriffskontrollliste gesteuert.

Die Liste der verfügbaren Rollen in einer Datenbank

Rollen auslesen

Die verfügbaren Rollen befinden sich in der Eigenschaft Roles in der Klasse ACL. Diese Eigenschaft hat den Datentyp Vector.

```

class ACL {

    java.util.Vector getRoles();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

```

```

        Session se = getSession();
        AgentContext ac = se.getAgentContext();
        Database db = ac.getCurrentDatabase();
        ACL acl = db.getACL();
        Vector vec = acl.getRoles();
        Enumeration enum = vec.elements();

        while ( enum.hasMoreElements() ) {
            String rolle = (String)enum.nextElement();
            System.out.println ( "      " + rolle );
        };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};

```

Listing 4.93: Beispiel ACL.getRoles

Rolle hinzufügen

Um eine Rolle in die Liste der möglichen Rollen hinzuzufügen, kann die Methode `addRole` in der Klasse `ACL` verwendet werden. Diese Methode erwartet als einzigen Parameter den Namen der neuen Rolle.

```

class ACL {

    void addRole(String name);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            acl.addRole ( "Admin" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.94: Beispiel ACL.addRole

Rolle löschen

Das Entfernen einer Rolle aus der Liste der möglichen Rollen kann mit der Methode `deleteRole` erfolgen. Bei dieser Methode wird auch nur der Name der Rolle als einziger Parameter erwartet.

```
class ACL {

    void deleteRole(String name);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();

            acl.deleteRole ( "Admin" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.95: Beispiel `ACL.deleteRole`

Die Rollen eines Eintrags in der Zugriffskontrollliste

Auslesen der zuerkannten Rollen eines Eintrags

Alle zuerkannten Rollen eines Eintrags befinden sich in der Eigenschaft `Roles` des Eintrags und können über diese in Form eines Vektors ausgelesen werden.

```
class ACLEntry {

    java.util.Vector getRoles();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {
```

```

Session se = getSession();
AgentContext ac = se.getAgentContext();
Database db = ac.getCurrentDatabase();
ACL acl = db.getACL();
ACLEntry entry = acl.getFirstEntry();

while ( entry != null ) {
    System.out.println ( "    " + entry.getName() );
    Vector vec = entry.getRoles();
    Enumeration enum = vec.elements();

    while ( enum.hasMoreElements() ) {
        String rolle = (String)enum.nextElement();
        System.out.println ( "        " + rolle );
    };
    entry = acl.getNextEntry();
};

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.96: Beispiel *ACLEntry.getRoles*

Überprüfen, ob dem Eintrag eine bestimmte Rolle zuerkannt wurde

Um performant prüfen zu können, ob einem Eintrag eine bestimmte Rolle zuerkannt wurde, steht die Methode *isRoleEnabled* in der Klasse *ACLEntry* zur Verfügung. Diese Methode erwartet den Namen der Rolle als Parameter.

```

class ACLEntry {

    boolean isRoleEnabled(String role);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getFirstEntry();

```

```

        while ( entry != null ) {
            if ( entry.isRoleEnabled("admin") )
                System.out.println (entry.getName() );
            entry = acl.getNextEntry();
        };

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};
};

```

Listing 4.97: Beispiel *ACLEntry.isRoleEnabled*

Dem Eintrag eine Rolle zuerkennen

Eine für die Datenbank eingetragene verfügbare Rolle kann einem Eintrag in der Zugriffskontrollliste mit der Methode `enableRole` der Klasse *ACLEntry* zuerkannt werden. Der Methode muss dazu in Form eines Parameters der Name der Rolle mitgeteilt werden.

```

class ACLEntry {

    void enableRole(String role);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            ACL acl = db.getACL();
            ACLEntry entry = acl.getFirstEntry();

            while ( entry != null ) {
                entry.enableRole ( "admin" );
                entry = acl.getNextEntry();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.98: Beispiel *ACLEntry.enableRole*

Dem Eintrag eine Rolle aberkennen

So wie einem Eintrag in der Zugriffskontrollliste einer Rolle zuerkannt werden kann, kann dieser auch wieder aberkannt werden. Dazu steht die Methode `disableRole` zur Verfügung. Die Methode erwartet den Namen der Rolle als Parameter.

```
class ACLEntry {  
    void disableRole(String role);  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            ACL acl = db.getACL();  
            ACLEntry entry = acl.getFirstEntry();  
  
            while ( entry != null ) {  
                entry.disableRole ( "admin" );  
                entry = acl.getNextEntry();  
            };  
  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.99: *Beispiel ACLEntry.disableRole*

4.4.4 Konsistente Zugriffskontrollliste

Wenn von einer Datenbank mehrere Repliken existieren, kann es vorkommen, dass die Zugriffskontrolllisten bei den verschiedenen Repliken unterschiedliche Einträge haben. Dies kann zu Problemen bei der Administration führen. Deshalb gibt es die Möglichkeit, zu fordern, dass alle Zugriffskontrolllisten bei allen Repliken identisch sein müssen. Diese Forderung kann über die Eigenschaft `UniformAccess` gesteuert werden.

```
class ACL {  
    boolean isUniformAccess();  
    void setUniformAccess(boolean flag);  
}
```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DbDirectory dir = se.getDbDirectory ( null );
            Database db = dir.getFirstDatabase(DbDirectory.DATABASE);

            while ( db != null ) {
                if ( db.getACL().isUniformAccess() )
                    System.out.println ( db.getFileName() );
                db = dir.getNextDatabase();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.100: Beispiel ACL.UniformAccess

4.4.5 Internetzugriffsrechte

Wenn der so genannte http-Task auf dem Server läuft, kann jede Notes Datenbank nicht nur über den Client genutzt werden, sondern auch über einen Webbrowser. Da der Zugriff über den Webbrowser im Regelfall ungeschützt ist als ein Zugriff aus dem Client heraus, gibt es die Möglichkeit, den Zugriff über den Webbrowser gesondert zu beschränken.

Außer den bereits bekannten Möglichkeiten gibt es zusätzlich eine Einstellung für das höchste Zugriffslevel, welches einen Zugriff über das Web für die entsprechende Datenbank nutzen kann. Auch wenn der entsprechende Anwender in der Zugriffs-kontrollliste ein viel höheres Recht besitzt, so kann er doch nur die Rechte bis zu dem maximalen Internetzugriffslevel nutzen.

Diese Einstellung findet sich in der Eigenschaft InternetLevel in der Klasse *ACL* wieder.

```

class ACL {

    int getInternetLevel();
    void setInternetLevel(int level);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

```

```
try {  
  
    Session se = getSession();  
    DbDirectory dir = se.getDbDirectory ( null );  
    Database db = dir.getFirstDatabase(DbDirectory.DATABASE);  
  
    while ( db != null ) {  
        if ( db.getACL().getInternetLevel() ==  
            ACL.LEVEL_MANAGER )  
            System.out.println ( db.getFileName() );  
        db = dir.getNextDatabase();  
    };  
  
} catch(Exception e) {  
  
    e.printStackTrace();  
  
};  
  
};  
};
```

Listing 4.101: *Beispiel ACL.InternetLevel*

4.5 Items

4.5.1 Einleitung

Jedes Dokument in Notes besteht aus einer Menge von so genannten Items. Jedes dieser Items repräsentiert ein Datenfeld eines Dokuments. Diese Items werden in Java entweder durch eine Instanz der Klasse *Item* oder der Klasse *RichTextItem* referenziert. Jedes Item besitzt einen Namen, allerdings können in einem Dokument beliebig viele Items mit demselben Namen existieren.

4.5.2 Die Items eines Dokuments

Die Daten in einem Dokument werden in Items gespeichert. Dabei wird in der Regel für jedes in der Maske vorhandene Feld, das mit einem Wert gefüllt ist, ein Item angelegt.

Zugriff auf alle Items eines Dokuments

Wenn ein Dokument in Java in Form einer Instanz der Klasse *Document* vorliegt, ist es ein Leichtes, in diesem Dokument alle Items auszulesen. Dazu wird die Methode *getItems* verwendet. Diese Methode liefert als Rückgabewert einen Vektor mit allen Items, die in dem Dokument gespeichert sind, zurück. Die Elemente in diesem Vektor haben alle den Datentyp *Item*. Dieser Vektor kann dann mit den üblichen Javabordmitteln durchwandert bzw. bearbeitet werden.

```

class Document {

    java.util.Vector.getItems();

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                System.out.println("Id: " + doc.getNoteID());
                Vector vec = doc.getItems();
                for (int i=0; i< vec.size(); i++) {
                    Item item = (Item) vec.elementAt(i);
                    System.out.println (
                        item.getName() + "=" + item.getValueString()
                    );
                };
                doc = dc.getNextDocument();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.102: Beispiel Document.Items

Zugriff auf ein namentlich bekanntes Item eines Dokuments

Natürlich ist es in Notes auch möglich, ein Item direkt über dessen Namen anzusprechen. Hierfür wird der Name des Items der Methode `getFirstItem` übergeben. Diese liefert dann eine Referenz auf das erste Item im Dokument mit dem angegebenen Namen. Sollte kein derartiges Item existieren, wird der Wert *null* zurückgeliefert.

```

class Document {

    Item getFirstItem(String name);

}

```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                Item item = doc.getFirstItem ( "Form" );
                System.out.println("Id: " + doc.getNoteID() +
                    " - " + item.getName() );
                doc = dc.getNextDocument();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.103: Beispiel Document.getFirstItem

Um zu überprüfen, ob in einem Dokument überhaupt ein Item mit dem angegebenen Namen vorhanden ist, kann die Funktion `hasItem` der Klasse *Document* aufgerufen werden. Dieser Funktion wird als Parameter der gewünschte Itemname übergeben. Der Rückgabewert ist dann vom Datentyp `Boolean`.

```

class Document {

    boolean hasItem(String name);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                if ( doc.hasItem ( "sendto" ) ) {
                    System.out.println( doc.getNoteID() );
                };
            };
        };
    };
};

```

```

        doc = dc.getNextDocument();
    };

    } catch(Exception e) {

        e.printStackTrace();

    };

};

};

```

Listing 4.104: Beispiel Document.hasItem

Zugriff auf einzelne Werte in einzelnen Items

Zugriff auf Datumswerte

Für das Lesen und Schreiben von Datumswerten in Items werden unter Notes die Methode `getDateTimeValue` und `setDateTimeValue` angeboten. Durch den Aufruf der Methode `getDateTimeValue` erhält man als Rückgabewert den Inhalt eines Items von Typ `DateTime`. Bei Aufruf dieser Methode für ein Item mit einem Inhalt der nicht vom Typ `DateTime` ist, erhält man den Wert `null`. Zum Setzen wird die Methode `setDateTimeValue` verwendet. Diese Methode erwartet als Parameter den zu setzenden Wert.

```

class Item {

    DateTime getDateTimeValue();
    void setDateTimeValue(DateTime value);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                if ( doc.hasItem ( "startdatum" ) ) {
                    Item item = doc.getFirstItem ( "startdatum" );
                    DateTime date = item.getDateTimeValue();
                    date.adjustDay(1);
                    item.setDateTimeValue(date);
                    doc.save();
                };
                doc = dc.getNextDocument();
            };
        };
    };
}

```

```

        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};

```

Listing 4.105: Beispiel Item.DateTimeValue

Zugriff auf numerische Werte

Bei dem Zugriff auf den Inhalt von numerischen Items kann zwischen den Methoden `getValueInteger` bzw. `setIntegerValue` und `getValueDouble` bzw. `setDoubleValue` gewählt werden.

Die jeweilige `get`-Methode liefert als Rückgabewert den Inhalt des Items. Bei beiden `set`-Methoden wird der zu setzende Wert als Parameter erwartet.

```

class Item {
    double getValueDouble();
    setValueDouble(double value);
    int getValueInteger();
    setValueInteger(int value);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                if ( doc.hasItem ( "anzahl" ) ) {
                    Item item = doc.getFirstItem ( "anzahl" );
                    int anzahl = item.getValueInteger();
                    item.setDateTimeValue(anzahl+1);
                    doc.save();
                };
                doc = dc.getNextDocument();
            };
        } catch(Exception e) {

```

```

        e.printStackTrace();

    };

};

};

```

Listing 4.106: Beispiel Item.getValueInteger

Zugriff auf alphanummerische Werte

Natürlich stehen auch Methoden für den Zugriff auf Items mit alphanummerischem Inhalt unter Notes zur Verfügung. Diese Methoden tragen den Namen `getValueString` und `setValueString`. Die Methode `setValueString` erwartet als Parameter den zu setzenden Wert. Die Methode `getValueString` liefert bei Aufruf den aktuellen Inhalt des Items zurück.

```

class Item {

    public String getValueString();
    public setValueString(String value);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getFirstDocument();
            while ( doc != null ) {

                Item item1 = doc.getFirstItem ( "name1" );
                Item item2 = doc.getFirstItem ( "name2" );
                String kette = item1.getValueString();
                item2.setValueString(kette);

                doc = col.getNextDocument();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.107: Beispiel Item.getValueString

Zugriff auf alle Werte in einem Item

Außer den Methoden um auf einzelne Werte von Items zuzugreifen, gibt es die Methoden `getValues` und `setValues`, um auf alle Werte eines Items zuzugreifen. Hierbei werden die zu setzenden Werte als Parameter in Form eines Vektors erwartet. Dementsprechend liefert die Methode `getValues` den aktuellen Inhalt des Items auch in Form eines Vektors zurück.

```
class Item {

    java.util.Vector getValues();
    void setValues(java.util.Vector values);

}
```

Suchen eines Werts in einem Item

Notes stellt eine Methode zur Überprüfung, ob ein Item einen bestimmten Wert enthält, zur Verfügung. Diese Methode der Klasse *Item* hat den Namen `containsValue` und erwartet als Parameter den gesuchten Inhalt. Als Rückgabewert erhält man einen booleschen Ausdruck, welcher *true* ist, falls der angegebene Wert in dem Item gespeichert ist. *False* liefert die Methode, falls der gesuchte Wert nicht im Item enthalten ist.

```
class Item {

    boolean containsValue(Object value);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getFirstDocument();
            while ( doc != null ) {

                Item item = doc.getFirstItem ( "Form" );
                if ( item.containsValue("Memo") ) {
                    System.out.println ( doc.getNoteID() );
                };

                doc = col.getNextDocument();
            };

        } catch(Exception e) {
```

```
        e.printStackTrace();  
    };  
};
```

Listing 4.108: Beispiel Item.containsValue

Die verschiedenen Typen eines Items

Für die verschiedenen Typen, die ein Item annehmen kann, stehen verschiedene Methoden zur Verfügung, die Items hinsichtlich dieser Typen zu überprüfen.

Typenüberprüfung von Items auf den Typ Author

In dem Item dieses Typs wird der Name des Autors des Dokuments gespeichert.

```
class Item {  
    boolean isAuthors() ;  
    void setAuthors(boolean flag) ;  
}
```

Typenüberprüfung des Items auf den Typ Names

Items dieses Typs beinhalten die Namen von Notes Anwendern oder Servern.

```
class Item {  
    boolean isNames();;  
    setNames(boolean flag);  
}
```

Typenüberprüfung von Items auf den Typ Reader

Dieser Typ wird genutzt, um Items zu definieren, in denen die Namen der Anwender oder Server eingetragen werden, die dieses Dokument lesen dürfen.

```
class Item {  
    boolean isReaders();  
    void setReaders(boolean flag);  
}
```

Verschlüsselung und Sicherheit eines Items

Überprüfung, ob das Item verschlüsselt ist

Jedes Element eines Dokuments kann verschlüsselt werden. Ob das jeweilige Item verschlüsselt, ist verrät die Eigenschaft Encrypted des Items.

```
class Item {  
  
    boolean isEncrypted();  
    void setEncrypted(boolean flag);  
  
}
```

Überprüfung, ob das Item geschützt ist

Ob ein Item geschützt ist, verrät die Eigenschaft Protected.

```
class Item {  
  
    boolean isProtected();  
    void setProtected(boolean flag);  
  
}
```

Die Überprüfung der Signierung eines Items

Die Signierung eines Items wird durch die Eigenschaft Signed überprüft.

```
class Item {  
  
    boolean isSigned();  
    void setSigned(boolean flag);  
  
}
```

Flag zu Steuerung des Speicherverhaltens eines Items

Mit der Eigenschaft SaveToDisk kann verhindert werden, dass der Inhalt eines Items permanent gespeichert wird. Wenn diese Eigenschaft den Wert *false* besitzt, geht der Inhalt des entsprechenden Items verloren.

```
class Item {  
  
    boolean isSaveToDisk() ;  
    void setSaveToDisk(boolean flag) ;  
  
}
```

Steuerung, ob ein Item in einer View oder einer Folderansicht auftauchen kann

Die Elemente, die in einer Ansicht erscheinen, werden besonders behandelt. Ob ein Item diese gesonderte Behandlung bekommt und somit in der Ansicht erscheinen kann, bestimmt die Eigenschaft Summary. Wenn die Eigenschaft den Wert *true* besitzt, kann das entsprechende Item in einer Ansicht verwendet werden.

```
class Item {  
  
    boolean isSummary() ;  
    void setSummary(boolean flag) ;  
  
}
```

Zugriff auf den Inhalt eines bekannten Items in einem Dokument

Um auf den Inhalt eines Items auch aus einem Dokument heraus zugreifen zu können, ohne extra eine Referenz auf ein Item zu ermitteln, werden von Notes die Methoden `getItemValueDouble`, `getItemValueInteger` und `getItemValueString` der Klasse *Document* angeboten. Durch diese Methoden ist es möglich, bei der Angabe des Namens des Items den entsprechenden Wert direkt auszulesen.

```
class Item {
    double getItemValueDouble(String name);
    int getItemValueInteger(String name);
    String getItemValueString(String name);
    void replaceItemValue ( String, Object );
}
```

Benutzerdefinierte Itemtypen

Eine Besonderheit in der neuesten Notes Version ist das Verarbeiten von benutzerdefinierten Daten in Items. Zum Lesen und Schreiben dieser benutzerdefinierten Daten werden die Methoden `replaceItemValueCustomData` und `getItemValueCustomData` verwendet.

```
class Item {
    void replaceItemValueCustomData(
        String itemname, Object userobj
    ) throws java.io.IOException;

    void replaceItemValueCustomData(
        String itemname, String datatype, Object userobj
    ) throws java.io.IOException;
    Object getItemValueCustomData(
        String itemname
    ) throws ClassNotFoundException, java.io.IOException;

    Object getItemValueCustomData(
        String itemname, String datatype
    ) throws ClassNotFoundException, java.io.IOException;
}
```

4.6 RichText-Items

4.6.1 Was ist RichText?

RichText-Felder sind formatierte Textfelder, die Texte mit mehr als 256 Zeichen beinhalten können. Außerdem können in RichText-Feldern eingebettete Objekte, Dateianhänge, Verknüpfungen zu Notes Elementen oder andere Objekte gespeichert werden.

4.6.2 Erstellen eines RichText-Items in einem Dokument

Ein RichText-Item kann in einem Dokument über die Methode `createRichTextItem` in der Klasse *Document* angelegt werden.

```
class Document {
    RichTextItem createRichTextItem ( String name )
}
```

Diese Methode liefert nach dem Aufruf eine Referenz auf das neu erstellte RichText-Item zurück.

4.6.3 Zugriff auf ein bestehendes RichText-Item

Der Zugriff auf ein bestehendes RichText-Item erfolgt genauso wie der Zugriff auf andere Items mit der Methode `getFirstItem` der Klasse *Document*. Allerdings muss im Fall eines RichText-Items der Rückgabewert explizit gecastet werden.

```
public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();

            Document doc = dc.getFirstDocument();
            while (doc != null) {
                System.out.println("Id: " + doc.getNoteID());
                RichTextItem rt = (RichTextItem)doc.getFirstItem("Body");

                doc = dc.getNextDocument();
            };
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.109: Beispiel Document.getItem (RichTextItem)

4.6.4 Zugriff auf den Wert eines RichText-Items

Die Klasse *RichTextItem* beinhaltet verschiedene Methoden, um den Inhalt eines RichTexts zu manipulieren. Allerdings können immer nur Daten an das Ende des Inhalts angehängt werden.

Anhängen von Text

Mit der Methode `appendText` kann ein Text an das Item angehängt werden. Die Methode erwartet den Text als Parameter.

```
class RichTextItem {  
    public void appendText ( String text ) ;  
}
```

Anhängen eines Zeilen- oder Seitenumbruchs

Ein Zeilenumbruch wird mit der Methode `addNewLine` und der Seitenumbruch mit der Methode `addPageBreak` angehängt. Beim Hinzufügen eines Zeilenumbruchs kann der Methode optional die Anzahl der Umbrüche als Parameter mitgegeben werden. Außerdem lässt sich über den zweiten Parameter steuern, ob vor dem Zeilenumbruch ein Abschnittswechsel erfolgen soll.

Bei der Methode für den Seitenwechsel kann eine Instanz der Klasse *RichTextParagraphStyle* angegeben werden, die die Formatierung beeinflusst.

```
class RichTextItem {  
    void addNewLine () ;  
    void addNewLine ( int count ) ;  
    void addNewLine ( int count, boolean parabreak);  
  
    void addPageBreak();  
    void addPageBreak(RichTextParagraphStyle pstyle);  
}
```

Anhängen von Tabstops

Ein Tabstop kann an das RichText-Item mit der Methode `addTab` hinzugefügt werden. Dabei kann der Methode optional die Anzahl der Tabstops, die hinzugefügt werden sollen, angegeben werden.

```
class RichTextItem {  
    void addTab () ;  
    void addTab ( int count ) ;  
}
```

Anhängen von Verknüpfungen

Mit der Methode `appendDocLink` können Verknüpfungen zu anderen Notes Elementen in das RichText-Item geschrieben werden. Es können dabei Verknüpfungen zu einem Dokument, einer Datenbank oder einer Ansicht angegeben werden. Des Weiteren bietet sich optional die Möglichkeit, einen Kommentar und einen Anzeigetext für die Verknüpfung anzugeben.

```

class RichTextItem {

    void appendDocLink(Document doc);
    void appendDocLink(Database db);
    void appendDocLink(View vw);
    void appendDocLink(Document doc, String comment);
    void appendDocLink(Database db, String comment);
    void appendDocLink(View vw, String comment);
    void appendDocLink(Document doc, String comment, String hotspottext);
    public void appendDocLink(Database db, String comment,
        String hotspottext);
    public void appendDocLink(View vw, String comment,
        String hotspottext);

}

```

Anhängen von Inhalten anderer RichText-Items

Um den Inhalt eines anderen RichText-Items anzuhängen, kann die Methode `appendRTItem` genutzt werden. Der Methode muss dazu nur eine Referenz auf das RichText-Item mit dem zu kopierenden Inhalt übergeben werden.

```

class RichTextItem {

    void appendRTItem(RichTextItem item);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextItem rt = doc.createRichTextItem("Body");
            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                "Aktuelle Datenbank" );
            doc.save();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.110: Beispiel `Document.createRichTextItem`

4.6.5 Auslesen des Texts eines RichText-Items

Damit das Auslesen des Inhalts eines RichText-Items in eine normale Zeichenkette auch ohne großen Aufwand erfolgen kann, gibt es für diese Situation eine besondere Methode. Diese Methode trägt den Namen `getFormattedText`. Der Methode müssen drei Parameter übergeben werden. Der erste Parameter ist ein boolescher Wert, der bestimmt, ob Tabstops aus dem Text entfernt werden sollen. Durch den zweiten Parameter wird die Zeilenlänge definiert. Der dritte und letzte Parameter bestimmt die Länge des zu lesenden Texts.

```
class RichTextItem {

    getFormattedText ( boolean tabflag, int linelen,
                      int maxlen ) ;

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();

            Document doc = col.getFirstDocument();
            while ( doc != null ) {
                RichTextItem rt =
                    (RichTextItem)doc.getFirstItem("Body");
                String kette = rt.getFormattedText ( true, 80, 200 );
                System.out.println ( kette );
                doc = col.getNextDocument();
            };

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.111: Beispiel `RichTextItem.getFormattedText`

4.6.6 Die Zeichenformatierung in einem RichText-Item

Der Text eines RichText-Items kann durch verschiedene Zeichenformatierungen beliebig gestaltet werden. Diese Zeichenformatierung wird durch die Klasse *RichTextStyle* in Java abgebildet.

Erstellen einer neuen Zeichenformatierung

Um eine Zeichenformatierung festlegen zu können, wird als erstes eine neue Instanz der Klasse *RichTextStyle* benötigt. Diese neue Instanz erhält man von der parameterlosen Methode *createRichTextStyle* der Klasse *Session*.

```
class Session {
    RichTextStyle createRichTextStyle();
}
```

Nachdem die neue Instanz der Klasse *RichTextStyle* vorliegt, muss diese nur noch in dem RichText-Item gesetzt werden. Dies erfolgt mit der Methode *appendStyle*. Die Methode erwartet dazu als Parameter die Instanz der Klasse *RichTextStyle*.

```
class RichTextItem {
    void appendStyle ( RichTextStyle style );
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextStyle rs = se.createRichTextStyle ();

            RichTextItem rt = doc.createRichTextItem("Body");
            rt.appendStyle ( rs );

            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                              "Aktuelle Datenbank" );

            doc.save();
        } catch(Exception e) {
```

```
e.printStackTrace();

};

};

};
```

Listing 4.112: Beispiel RichTextItem.appendStyle

Die möglichen Zeichenformatierungen

Dabei stehen in dieser Klasse verschiedene Eigenschaften zur Formatierung zur Verfügung.

Eigenschaft	Werte	Beschreibung
Bold	YES NO STYLE_NO_CHANGE MAYBE	Fettdruck
Color	COLOR_BLACK COLOR_BLUE COLOR_CYAN COLOR_DARK_BLUE COLOR_DARK_CYAN COLOR_DARK_GREEN COLOR_DARK_MAGENTA COLOR_DARK_RED COLOR_DARK_YELLOW COLOR_GRAY COLOR_GREEN COLOR_LIGHT_GRAY COLOR_MAGENTA COLOR_RED COLOR_WHITE COLOR_YELLOW STYLE_NO_CHANGE MAYBE	Zeichenfarbe
Effects	EFFECTS_EMBOSS EFFECTS_EXTRUDE EFFECTS_NONE EFFECTS_SHADOW EFFECTS_SUBSCRIPT EFFECTS_SUPERSCRIPT STYLE_NO_CHANGE MAYBE	Zeichenstil (Hervorgehoben, Schatten, ...)
Font	FONT_COURIER FONT_HELV FONT_ROMAN STYLE_NO_CHANGE MAYBE	Schriftart

Tabelle 4.8: Eigenschaften der Klasse RichTextStyle

Eigenschaft	Werte	Beschreibung
FontSize	1-250 STYLE_NO_CHANGE MAYBE	Schriftgröße
Italic	YES NO STYLE_NO_CHANGE MAYBE	Kursiv
StrikeThrough	YES NO STYLE_NO_CHANGE MAYBE	Durchgestrichen
Underline	YES NO STYLE_NO_CHANGE MAYBE	Unterstrichen

Tabelle 4.8: Eigenschaften der Klasse RichTextStyle (Forts.)

Die in dieser Tabelle aufgeführten Konstanten sind alle in der Klasse *RichTextStyle* definiert. Die Konstante YES bedeutet, dass die entsprechende Zeichenformatierung eingeschaltet und die Konstante NO, dass sie ausgeschaltet wird. Die Konstanten STYLE_NO_CHANGE und MAYBE sagen aus, dass sich die Einstellung der jeweiligen Formatierung nicht ändert.

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextStyle rs = se.createRichTextStyle ();

            RichTextItem rt = doc.createRichTextItem("Body");
            rps.setLeftMargin ( 100 );
            rt.appendParagraphStyle ( rps );

            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                               "Aktuelle Datenbank" );

            doc.save();

        } catch(Exception e) {
```

```

        e.printStackTrace();
    };
};
};

```

Listing 4.113: Beispiel RichTextStyle

4.6.7 Die Absatzformatierung in einem RichText-Item

Die Absatzformatierung eines RichText-Items wird in Java durch die Klasse *RichTextParagraphStyle* dargestellt.

Erstellen einer Absatzformatierung

Auch bei dieser Klasse wird zuerst in der Session eine Instanz der Klasse erzeugt. Hierzu wird die Methode *createRichTextParagraphStyle* aufgerufen.

```

class Session {

    RichTextParagraphStyle createRichTextStyle();

}

```

Um diese neue Instanz auch verwenden zu können, steht die Methode *appendParagraphStyle* zur Verfügung. Diese Methode funktioniert äquivalent zu der Methode *appendStyle*.

```

class RichTextItem {

    void appendParagraphStyle (
        RichTextParagraphStyle style );

}

```

Die möglichen Absatzformatierungen

Für die Absatzformatierung stehen die folgenden Eigenschaften zur Verfügung:

Eigenschaft	Werte	Beschreibung
Alignment	ALIGN_CENTER ALIGN_FULL ALIGN_LEFT ALIGN_NOWRAP ALIGN_RIGHT	Ausrichtung
FirstLineLeftMargin	numerisch	Linker Einzug der ersten Zeile
InterLineSpacing	SPACING_DOUBLE SPACING_ONE_POINT_50 SPACING_SINGLE	Zeilenabstand
LeftMargin	numerisch	Linker Einzug

Tabelle 4.9: Eigenschaften für die Absatzformatierung

Eigenschaft	Werte	Beschreibung
Pagination	PAGINATE_BEFORE PAGINATE_DEFAULT PAGINATE_KEEP_TOGETHER PAGINATE_KEEP_WITH_NEXT	Zeilenumbruchsverhalten zwischen den Absätzen
RightMargin	numerisch	Rechter Einzug
SpacingAbove	SPACING_DOUBLE SPACING_ONE_POINT_50 SPACING_SINGLE	Zeilenabstand vor dem Absatz
SpacingBelow	SPACING_DOUBLE SPACING_ONE_POINT_50 SPACING_SINGLE	Zeilenabstand nach dem Absatz

Tabelle 4.9: Eigenschaften für die Absatzformatierung (Forts.)

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextParagraphStyle rps = se.createRichTextStyle ();

            RichTextItem rt = doc.createRichTextItem("Body");
            rps.setLeftMargin ( 100 );
            rt.appendParagraphStyle ( rps );

            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                               "Aktuelle Datenbank" );

            doc.save();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.114: Beispiel RichTextParagraphStyle

Die Tabulatoren

Die Tabulatoren, die in einem RichText-Item zu finden sind, werden auch über die Klasse *RichTextParagraphStyle* bearbeitet. Für das Setzen und Löschen der Tabulatoren stehen verschiedene Methoden zur Verfügung.

Setzen von Tabulatoren

Für das Setzen von Tabulatoren stehen zwei verschiedene Methoden zur Auswahl. Der einfachste Weg ist der Aufruf der Methode `setTab`, diese erwartet die Position und den Typ des Tabulators als Parameter.

```
class RichTextParagraphStyle {
    void setTab ( int pos, int type );
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextStyle rs = se.createRichTextStyle ();

            RichTextItem rt = doc.createRichTextItem("Body");
            rps.setLeftMargin ( 100 );
            rps.setTab ( 50, RichTextParagraphStyle.TAB_CENTER );
            rt.appendParagraphStyle ( rps );

            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                "Aktuelle Datenbank" );

            doc.save();
        } catch(Exception e) {
            e.printStackTrace();
        }
    };
};
```

Listing 4.115: Beispiel *RichTextParagraphStyle.setTab*

Die andere Methode zum Setzen eines Tabulators hat den Namen `setTabs`. Im Unterschied zur Methode `setTab` wird hier nicht ein Tabulator gesetzt, sondern gleich mehrere. Dazu erwartet die Methode als Parameter die Anzahl, die Startposition und den Abstand der Tabulatoren. Als optionaler Parameter kann der Tabulatortyp angegeben werden.

```
class RichTextParagraphStyle {

    void setTabs ( int count, int pos, int inter, int type );
    void setTabs ( int count, int pos, int inter );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextStyle rs = se.createRichTextStyle ();

            RichTextItem rt = doc.createRichTextItem("Body");
            rps.setLeftMargin ( 100 );
            rps.setTabs ( 2, 50, 10 );
            rt.appendParagraphStyle ( rps );

            rt.appendText ( "Erste" );
            rt.addTab(2);
            rt.appendText( "Zeile" );
            rt.addNewLine(2);
            rt.appendDocLink ( db, "Aktuelle Datenbank",
                               "Aktuelle Datenbank" );

            doc.save();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.116: *Beispiel RichTextParagraphStyle.setTabs*

Löschen der Tabulatoren

Gelöscht werden die Tabulatoren mit der Methode `clearAllTabs`.

```
class RichTextParagraphStyle {
    public void clearAllTabs();
}
```

4.7 Datums- und Zeithandling

Für jede Software, die in großen Netzwerken auch über die Grenzen von Zeitzonen eingesetzt wird, ist das Verarbeiten von Datums- und Zeitwerten eine nicht zu vernachlässigende Aufgabe.

4.7.1 Datums- oder Zeitwerte

Datums- und Zeitwerte werden in Notes durch die Klasse *DateTime* repräsentiert.

Einen Datums- oder Zeitwert anlegen

Ein Datums- oder Zeitwert wird immer basierend auf einer Session erzeugt. Dafür gibt es in der Klasse *Session* die Methode `createDateTime`. Diese Methode hat als Parameter das gewünschte Datum in Form einer Zeichenkette vom Typ `java.util.Date` oder vom Typ `java.util.Calendar`.

```
class Session {
    DateTime createDateTime(String date)
    DateTime createDateTime(java.util.Date date)
    DateTime createDateTime(java.util.Calendar date)
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();

            DateTime dat1 = se.createDateTime ( "today" );

        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.117: Beispiel `Session.createDateTime`

Zugriff auf einen Datums- oder Zeitwert

Häufig kommt es vor, dass von einem Datums- und Zeitwert nur das Datum oder nur die Zeitangabe benötigt wird. Um diese auslesen zu können, werden die Eigenschaften `DateOnly` und `TimeOnly` in der Klasse *DateTime* angeboten.

```
class DateTime {  
    String getDateOnly()  
    String getTimeOnly()  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            DateTime dat = se.createDateTime ( "today" );  
            System.out.println ( "Datum: " + dat.getDateOnly() );  
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.118: Beispiel *DateTime.getDateOnly* und *DateTime.getTimeOnly*

Spezielle Zeitwerte

In bestimmten Situationen ist es notwendig, spezielle Datums- und Zeitwerte zu verwenden, die für unbestimmte Datums- oder Zeitwerte stehen.

Für eine Zeitangabe, bei der das Datum nicht von Interesse ist und unbestimmt sein soll, kann die Methode `setAnyDate` der Klasse *DateTime* verwendet werden, um dies zu kennzeichnen.

```
class DateTime {  
    void setAnyDate()  
}
```

Wenn nicht das Datum unbestimmt ist, sondern die Zeitangabe, muss dies mit der Methode `setAnyTime` gekennzeichnet werden.

```
class DateTime {
    void setAnyTime()
}
```

Ein anderer spezieller Wert, der gesetzt werden kann, ist das aktuelle Datum und die aktuelle Zeit. Mit der Methode `setNow` werden das aktuelle Datum und die aktuelle Zeit ermittelt und in die Instanz der Klasse *DateTime* geschrieben.

```
class DateTime {
    void setNow()
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            DateTime dt = se.createDateTime("Today");
            dt.setNow();
            System.out.println ( "Datum: " + dt.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dt.getTimeOnly() );
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.119: Beispiel *DateTime.setNow*

Anpassung des Datums- oder Zeitwerts

Häufig muss zu einer Datums- oder Zeitangabe ein bestimmter Zeitabstand addiert oder subtrahiert werden. Damit dies ohne großes Umrechnen möglich ist, stehen in der Klasse *DateTime* die Methoden `adjustDay`, `adjustHour`, `adjustMinute`, `adjustMonth`, `adjustSecond` und `adjustYear` zur Verfügung. Die Methoden erwarten jeweils einen numerischen Wert als Parameter mit der gewünschten Anzahl an Tagen, Stunden, Minuten, Monaten, Sekunden oder Jahren. Außerdem kann optional als zweiter Parameter angegeben werden, dass die Tageszeit bei der Berechnung berücksichtigt werden soll.

```
class DateTime {

    void adjustDay(int n)
    void adjustDay(int n, boolean preserveLocalTime)
    void adjustHour(int n)
    void adjustHour(int n, boolean preserveLocalTime)
    void adjustMinute(int n)
    void adjustMinute(int n, boolean preserveLocalTime)
    void adjustMonth(int n)
    void adjustMonth(int n, boolean preserveLocalTime)
    void adjustSecond(int n)
    void adjustSecond(int n, boolean preserveLocalTime)
    void adjustYear(int n)
    void adjustYear(int n, boolean preserveLocalTime)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DateTime dat = se.createDateTime("Today");
            dat.setNow();
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );
            System.out.println ( "Eine Sekunde später" );
            dat.adjustSecond(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );
            System.out.println ( "Eine Minute später" );
            dat.adjustMinute(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );
            System.out.println ( "Eine Stunde später" );
            dat.adjustHour(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );
            System.out.println ( "Eine Tag später" );
            dat.adjustDay(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );
            System.out.println ( "Eine Monat später" );
            dat.adjustMonth(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            dat.adjustYear(1);
            System.out.println ( "Datum: " + dat.getDateOnly() );
            System.out.println ( "Uhrzeit: " + dat.getTimeOnly() );

        } catch(Exception e) {
```

```

        e.printStackTrace();

    };

};
};

```

Listing 4.120: *Beispiel `DateTime.adjustSecond` bis `DateTime.adjustYear`*

Zeitdifferenz

Wenn zwei Zeitangaben vorliegen, wird häufig nach der Zeitspanne gefragt, die zwischen diesen beiden Zeiten liegt. Um diese Zeitspanne zu ermitteln, gibt es die Methode `timeDifference` in der Klasse `DateTime`. Die Methode erwartet als Parameter das Startdatum mit dem Datentyp `DateTime`.

```

class DateTime {

    int timeDifference(DateTime dt)
    double timeDifferenceDouble(DateTime dt)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DateTime dat1 = se.createDateTime("Today");
            DateTime dat2 = se.createDateTime("Today");
            dat2.adjustDay(2);
            System.out.println ( "Datum1: " + dat1.getDateOnly() );
            System.out.println ( "Datum2: " + dat2.getDateOnly() );
            System.out.println ( "Differenz: " +
                dat1.timeDifferenece(dat2) );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.121: *Beispiel `DateTime.timeDifference`*

4.7.2 Datums- oder Zeitbereich

Um in Notes einen Datums- oder Zeitbereich abzubilden, wird die Klasse `DateRange` verwendet. Der Datums- oder Zeitbereich basiert auf zwei Datums- oder Zeitwerten.

Einen Datums- oder Zeitbereich anlegen

Ähnlich wie bei den Datums- und Zeitwerten wird der Datums- oder Zeitbereich mit der Methode `createDateRange` der Klasse *Session* erstellt. Die Methode kann mit verschiedenen Parametern aufgerufen werden.

Wenn diese Methode ohne Parameter aufgerufen wird, wird ein leerer Datums- oder Zeitbereich angelegt. Der Methode kann aber auch ein Startzeitpunkt und ein Endzeitpunkt übergeben werden. Dabei können die Parameter entweder den Datentyp `DateTime` oder `java.util.Date` haben.

```
class Session {  
  
    DateRange createDateRange()  
    DateRange createDateRange(DateTime startt, DateTime endt)  
    DateRange createDateRange(java.util.Date startt,  
                               java.util.Date endt)  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            DateTime dat1 = se.createDateTime("Today");  
            DateTime dat2 = se.createDateTime("Today");  
            dat2.adjustDay(2);  
  
            DateRange rag = se.createDateRange ( dat1, dat2 );  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        };  
  
    };  
};
```

Listing 4.122: *Beispiel Session.createDateRange*

Zugriffe auf Start- und Endzeitpunkt

Der Zugriff auf den Start- und Endzeitpunkt eines Bereichs erfolgt über die Eigenschaften `StartDateTime` oder `EndDateTime`. Diese Eigenschaften können ausgelesen oder gesetzt werden.

```
class DateRange {  
  
    DateTime getStartDateTime()  
    void setStartDateTime(DateTime end)  
    DateTime getEndDateTime()  
    void setEndDateTime(DateTime end)
```

```

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            DateTime dat1 = se.createDateTime("Today");
            DateTime dat2 = se.createDateTime("Today");
            dat2.adjustDay(2);

            DateRange rag = se.createDateRange ( dat1, dat2 );
            dat1 = rag.getStartDateTime();
            dat2 = rag.getEndDateTime();
            rag.setStartDateTime(dat2);
            rag.setEndDateTime(dat1);

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.123: Beispiel DateRange.StartDateTime und DateRange.EndDateTime

4.8 Eingebundene Objekte und Dateianhänge

4.8.1 Was sind eingebundene Objekte, verknüpfte Objekte oder Dateianhänge?

Ein RichText-Item bietet die Möglichkeit, eingebundene Objekte, verknüpfte Objekte oder Dateianhänge in einem Dokument zu speichern.

Ein eingebettetes Objekt ist ein Objekt mit Daten einer OLE-fähigen Windows-Anwendung. Auf dieses Objekt wird durch die Anwendung, mit der das Objekt erstellt wurde, zugegriffen. Damit ist es möglich, fremde Anwendungsdaten in einer Windows-Anwendung zu verarbeiten. Die Daten werden dabei physikalisch im Notes Dokument gespeichert.

Bei einem verknüpften Objekt wird eine ähnliche Technik verwendet. Auch hier erfolgt der Zugriff über die fremde Anwendung. Nur wird bei der Speicherung eines verknüpften Objekts eine entsprechende Datei außerhalb des Dokuments auf der Festplatte gespeichert. Im Dokument selbst wird nur der Standort der Datei vermerkt.

Wenn eine Datei an ein Dokument in Notes angehängt wird, werden die Daten ähnlich wie beim eingebetteten Objekt gespeichert. Nur ist es bei dem Dateianhang

nicht möglich, direkt auf die Datei zuzugreifen. Die Datei wird von Notes nur als Speicherblock angesehen. Es besteht keine Verbindung zur Anwendung, mit der die Datei erstellt wurde.

4.8.2 Eingebettete Objekte

Erstellen von eingebetteten Objekten

Ein eingebettetes Objekt wird mit der Methode `embedObject` in der Klasse *RichTextItem* erstellt. Dabei muss der Methode durch den ersten Parameter mitgeteilt werden, dass ein eingebettetes Objekt erstellt werden soll. Dies erfolgt, indem man als Wert für den ersten Parameter die Konstante `EmbeddedObject.EMBED_OBJECT` angibt. Als zweiten Parameter wird der Name der Anwendung erwartet, mit der das Objekt bearbeitet werden soll (z. B. Microsoft Word). Diese Zeichenkette ist abhängig vom internen Namen der installierten Anwendung. Der dritte Parameter ist die Datei, aus der das Objekt erstellt werden soll. Durch diesen Parameter kann eine bestehende Datei als Vorlage für das Objekt genutzt werden. Der vierte und letzte Parameter ist der Name für das eingebettete Objekt.

```
class RichTextItem {  
    EmbeddedObject embedObject (int type, String class,  
        String source, String name);  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
  
            Document doc = db.createDocument();  
            RichTextItem rt = doc.createRichTextItem("Body");  
            rt.embedObject ( EmbeddedObject.EMBED_OBJECT,  
                "microsoft.word", "", "dateil" );  
            doc.save();  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    };  
};  
};
```

Listing 4.124: *Beispiel RichTextItem.embedObject*

Zugriff auf bestehende eingebettete Objekte

Um auf ein bestehendes eingebettetes Objekt zuzugreifen, gibt es verschiedene Wege. Einer davon führt über die Methode `getEmbeddedObject` der Klasse *RichTextItem*. Diese Methode erwartet für den Zugriff nur den Namen des Objekts als Parameter.

```
class RichTextItem {

    public EmbeddedObject getEmbeddedObject(String name)
        ;

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection col = db.getAllDocuments();
            Document doc = col.getFirstDocument();
            RichTextItem rt = doc.getFirstItem("Body");
            EmbeddedObject obj = rt.getEmbeddedObject ( "date1" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 4.125: Beispiel *RichTextItem.getEmbeddedObject*

Löschen von eingebetteten Objekten

Gelöscht wird ein eingebettetes Objekt durch den Aufruf der Methode `remove` in der Klasse *EmbeddedObject*.

```
class EmbeddedObject {

    public void remove () ;

}
```

Die Aktionen von eingebetteten Objekten

Wie schon erwähnt, nutzt ein eingebettetes Objekt OLE als Schnittstelle zur Kommunikation. Diese Schnittstelle legt fest, dass ein OLE-Objekt eine beliebige Anzahl von Aktionen ausführen kann. Welche und wie viele Aktionen dies umfasst, wird durch die jeweilige Anwendungssoftware bestimmt.

Die Liste dieser Aktionen befindet sich in der Eigenschaft `Verbs` der Klasse *EmbeddedObject*. Somit ist es möglich, sich über die Liste zu informieren, welche Aktion mit dem jeweiligen Objekt ausgeführt werden kann.

Um diese Aktion dann auszuführen, muss nur die Methode `doVerb` aufgerufen werden. Diese Methode erwartet den Namen der Aktion als Parameter. Nach Aufruf der Methode wird versucht, die Anwendungssoftware für das Objekt zu starten und die geforderte Aktion auszuführen.

```
class EmbeddedObject {  
    void doVerb ( String verb ) ;  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection col = db.getAllDocuments();  
            Document doc = col.getFirstDocument();  
            RichTextItem rt = doc.getFirstItem("Body");  
            EmbeddedObject obj = rt.getEmbeddedObject ( "object" );  
            obj.doVerb ( "print" );  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    };  
};
```

Listing 4.126: Beispiel *EmbeddedObject.doVerb*

4.8.3 Verknüpfte Objekte

Erstellen verknüpfter Objekte

Auch bei der Erstellung eines verknüpften Objekts wird die Methode `embedObject` der Klasse *RichTextItem* aufgerufen. Im ersten Parameter wird als Typ die Konstante `EmbeddedObject.EMBED_OBJECTLINK` übergeben. Der zweite Parameter ist in diesem Fall immer ein Leerstring. Im dritten Parameter wird der Standort bzw. der Dateiname mit Pfad der zu verknüpfenden Datei erwartet. Der letzte Parameter ist wieder der Name des Objekts.

```

class RichTextItem {

    EmbeddedObject embedObject (int type, String class,
                                String source, String name)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Document doc = db.createDocument();
            RichTextItem rt = doc.createRichTextItem("Body");
            rt.embedObject ( EmbeddedObject.EMBED_OBJECTLINK,
                            "", "C:\\Daten\\Dokument.doc", "datei1" );
            doc.save();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.127: *Beispiel RichTextItem.embedObject*

Zugriff auf und Löschen von verknüpften Objekten

Der Zugriff auf verknüpfte Objekte erfolgt genauso wie der Zugriff auf eingebundene Objekte. Auch das Löschen der verknüpften Objekte erfolgt durch denselben Methodenaufruf, nur dass bei verknüpften Objekten die Datei natürlich nicht gelöscht wird.

4.8.4 Dateianhänge

Anhängen

Auch beim Anhängen von Dateien wird die Methode `embedObject` verwendet. Wie auch bei den anderen beiden Varianten wird wieder durch den ersten Parameter bestimmt, welche Art von Objekt erstellt werden soll. Um einen Dateianhang zu erstellen, wird die Konstante `EmbeddedObject.EMBED_ATTACHMENT` angegeben. Als zweiter Parameter folgt ein Leerstring. Im dritten Parameter sollten der Dateiname und der Pfad der anzuhängenden Datei stehen. Der vierte Parameter beinhaltet wieder den Namen des Objekts.

```
class RichTextItem {
    EmbeddedObject embedObject (int type, String class,
        String source, String name)
}
```

Speichern

Das Speichern eines Dateianhangs auf einem Datenträger erfolgt mit der Methode `extractFile` der Klasse *EmbeddedObject*.

```
class EmbeddedObject {
    void extractFile ( String file ) ;
}
```

Die Methode erwartet als Parameter den Dateinamen. Der Inhalt des Dateianhangs wird nach dem Aufruf der Methode in die angegebene Datei geschrieben.

Abhängen

Das Abhängen erfolgt durch die Methode `remove` in der Klasse *EmbeddedObject*.

```
class EmbeddedObject {
    void remove ();
}
```

4.9 Internationalisierung

Da Notes ein international eingesetztes Softwareprodukt ist, ist es selbstverständlich, dass die Entwicklung von mehrsprachigen Anwendungen unterstützt wird. Für diese Unterstützung wird die Klasse *International* eingesetzt. In dieser Klasse findet man alles, was man für die geografische und sprachspezifische Behandlung von Zeitangaben und Währungen benötigt. Die Klasse besitzt dazu verschiedene lesbare Eigenschaften.

4.9.1 Zahlen- und Währungsangaben

Die folgenden Eigenschaften dienen zur länderspezifischen Darstellung von Währungsbeträgen:

Eigenschaft	Datentyp	Zugriff	Bedeutung
CurrencyDigits	int	getCurrencyDigits	Anzahl der Dezimalstellen
CurrencySymbol	String	getCurrencySymbol	Währungssymbol
CurrencySpace	boolean	IsCurrencySpace	Leerzeichen zwischen Betrag und Währungssymbol

Tabelle 4.10: Eigenschaften für die Formatierung von Zahlen- und Währungsangaben

Eigenschaft	Datentyp	Zugriff	Bedeutung
CurrencySuffix	boolean	isCurrencySuffix	Währungssymbol hinter Betrag
CurrencyZero	boolean	isCurrencyZero	Führende 0 bei Werten ohne Vorkommawerte
DecimalSep	String	getDecimalSep	Dezimaltrennzeichen
ThousandsSep	String	getDecimalSep	Tausenderseparator

Tabelle 4.10: Eigenschaften für die Formatierung von Zahlen- und Währungsangaben

4.9.2 Datumsangaben

Für die länderspezifische Darstellung der Datumsangaben dienen folgende Eigenschaften:

Eigenschaft	Datentyp	Zugriff	Bedeutung
DateSep	String	getDateSep	Datumstrennzeichen
DateDMY	boolean	isDateDMY	Datumsformat ist Tag/Monat/Jahr
DateMDY	boolean	isDateMDY	Datumsformat ist Monat/Tag/Jahr
DateYMD	boolean	isDateYMD	Datumsformat ist Jahr/Monat/Tag
Time24Hour	boolean	isTime24Hour	Zeitangabe im 24-Stunden-Format
TimeSep	String	getTimeSep	Trennzeichen für Zeitangabe
DST	boolean	isDst	Berücksichtigung der Sommerzeit
TimeZone	int	getTimeZone	Zeitzone, in den meisten Fällen die Anzahl der Stunden, die addiert oder subtrahiert werden müssen, um die GMT-Zeit zu erhalten

Tabelle 4.11: Eigenschaften für die Formatierung von Datumsangaben

Außerdem gibt es folgende sprachspezifische Bezeichnungen bei der Datumsdarstellung:

Eigenschaft	Datentyp	Zugriff	Bedeutung
AMString	String	getAMString	Zeichenkette für die Bezeichnung des Nachmittags
PMString	String	getPMString	Zeichenkette für die Bezeichnung des Vormittags
Today	String	getToday	Bezeichnung für Heute
Tomorrow	String	getTomorrow	Bezeichnung für Morgen
Yesterday	String	getYesterday	Bezeichnung für Gestern

Tabelle 4.12: Eigenschaften für die sprachspezifische Formatierung von Datumsangaben

4.10 Die Protokollierung

Der wohl wichtigste Grund für die Protokollierung ist die Fehlersuche. Kein Programm ist unfehlbar. Das Protokoll ist in vielen Fällen das wichtigste Hilfsmittel bei der Fehlersuche. Für das Protokollieren stellt Lotus einige hilfreiche Mechanismen zur Verfügung. Die Steuerung des Protokollierens erfolgt über die Klasse *Log*.

4.10.1 Das Protokollmedium

Bei der Protokollierung unter Lotus kann man zwischen verschiedenen Medien wählen, die zur Protokollierung verwendet werden sollen.

Erzeugen einer neuen Log-Instanz

Als Erstes muss eine neue Log-Instanz erzeugt werden. Dafür gibt es in der Klasse *Session* die Methode `createLog`.

```
class Session {  
  
    Log createLog(String name);  
  
}
```

Die Protokolldatenbank

Die erste Möglichkeit ist eine Protokolldatenbank. Die Protokolldatenbank ist aufgebaut wie jede andere Datenbank von Notes. Eine spezielle Protokolldatenbank haben Sie wahrscheinlich bereits kennen gelernt. Ein Beispiel für eine solche Protokolldatenbank ist nämlich das Notes Systemprotokoll in der Datenbank *log.nsf*. Um selbst eine Protokolldatenbank zu nutzen, muss man diese zuerst einmal anlegen. Das Anlegen der Protokolldatenbank unterscheidet sich nicht von dem Anlegen einer anderen Datenbank. Für eine Protokolldatenbank kann eine bereits bestehende Datenbankschablone verwendet werden. Mit Notes/Domino wird eine Schablone mit dem Namen *StrR4AgentLog* (ALOG4.NSF) mitgeliefert.

Öffnen einer Protokolldatenbank zum Protokollieren

Bevor mit dem Protokollieren begonnen werden kann, muss die Protokolldatenbank geöffnet werden. Dazu steht die Methode `openNotesLog` der Klasse *Log* zur Verfügung. Nach dem Öffnen der Datenbank können Protokolleinträge in die Datenbank erfolgen.

```
class Log {  
  
    void openNotesLog(String server, String database)  
  
}
```

Dieser Methode werden der Servername des Servers, auf dem sich die Protokolldatei befindet, sowie der Name der Protokolldatenbank übergeben.

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Log log = se.createLog("Testlog");
            log.openNotesLog ( "testserver", "log.nsf" );

            // Protokollierung der Ereignisse

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.128: Beispiel Log.openNotesLog

Aufbau der Protokolleinträge in der Datenbank

Jedes Mal, wenn eine Nachricht oder ein Fehler protokolliert werden soll, wird ein Dokument in der Datenbank erstellt. Dabei werden in dem Dokument folgende Einträge vorgenommen:

Name	Beschreibung
Form	Name der Maske
A\$PROGNAME	Der Programmname
A\$LOGTIME	Datum des Eintrags
A\$USER	Name des Anwenders
A\$LOGTYPE	Eintragstyp entweder Fehler („Error“) oder Nachricht („Action“)
A\$ACTION	Der Nachrichtentext bei einer Nachricht
A\$ERRCODE	Der Fehlercode bei einem Fehler
A\$ERRMSG	Der Fehlertext bei einem Fehler

Tabelle 4.13: Einträge im Protokolldokument

Die Protokolldatei

Die Protokolldatei ist eine Textdatei. Für jeden Protokolleintrag wird in der Protokolldatei eine Zeile angefügt.

Öffnen einer Protokolldatei

Auch die Protokolldatei muss wie die Protokolldatenbank vor dem ersten Gebrauch geöffnet werden. Dies erfolgt mit der Methode `openFileLog` der Klasse `Log`.

```
class Log {  
    openFileLog(String filepath)  
}
```

Diese Methode erwartet den Dateinamen der Protokolldatei, in der protokolliert werden soll.

```
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            Log log = se.createLog("Testlog");  
            log.openFileLog ( "logdatei.txt" );  
  
            // Protokollierung der Ereignisse  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.129: Beispiel Log.openFileLog

Anhängen oder Überschreiben der Protokolldatei

Anhängen oder überschreiben, das ist hier die Frage, wenn die Protokolldatei bereits existiert. Diese Frage wird von der Eigenschaft `OverwriteFile` der Klasse `Log` beantwortet. Abgefragt und gesetzt wird diese Eigenschaft mit den Befehlen `isOverwriteFile` und `setOverwriteFile`. Überschrieben wird die Protokolldatei, wenn diese Eigenschaft den Wert *true* hat. Wenn sie den Wert *false* hat, wird die Datei nicht überschrieben und die neuen Einträge an die bestehende Datei angehängt.

Die Protokollmail

Beim Protokollieren über das Mailsystem werden die Protokolleinträge in einer Mail gesammelt und diese an eine gewünschte Adresse gesendet.

Öffnen einer Protokollmail

Vor dem Erzeugen von Protokolleinträgen in einer Protokollmail muss eine leere Mail generiert werden. Dieses Generieren übernimmt die `OpenMailLog`-Methode der Klasse `Log`. Als erster Parameter wird ein Vektor mit Strings erwartet. In diesem Vektor stehen die Empfänger für die Protokollmail. Als zweiter Parameter kann das gewünschte Thema für die Mail angegeben werden.

```

class Log {

    void openMailLog(java.util.Vector recipients, String subject)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Log log = se.createLog("Testlog");
            Vector vec = new Vector();
            vec.addElement ( "admin/testdom" );
            log.openFileLog ( vec, "Logeintrag" );

            // Protokollierung der Ereignisse

        } catch(Exception e) {

            e.printStackTrace();

        };

    };
};

```

Listing 4.130: Beispiel Log.openMailLog

Absenden einer Protokollmail

Beim Schließen des Mailprotokolls mit der Methode `close` der Klasse `Log` wird die Protokollmail abgeschickt.

4.10.2 Meldung protokollieren

Um z. B. Statusänderungen oder Zwischenergebnisse zu protokollieren, steht die Methode `logAction` zur Verfügung. Diese Methode ist dazu gedacht, beliebige Meldungen zu protokollieren. Die Methode kann verwendet werden, sobald das entsprechende Protokollmedium geöffnet wurde. Die Art des Mediums wird beim Öffnen bestimmt. Als einzigen Parameter erwartet die Methode die Meldung, die in das Protokoll geschrieben werden soll.

```

class Log {

    void logAction(String action)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

```



```

        Session se = getSession();
        Log log = se.createLog("Testlog");
        log.openNotesLog ( "testserver", "log.nsf" );

        log.logAction ( "Aktion1" );

    } catch(Exception e) {

        e.printStackTrace();

    };

};
};

```

Listing 4.131: Beispiel Log.logAction

An- und Abschalten der Protokollierung von Meldungen

Das Protokollieren von Meldungen kann über die Eigenschaft `LogActions` an- und abgeschaltet werden. Wenn diese Eigenschaft den Wert *true* besitzt, werden die Meldungen protokolliert. Bei dem Wert *false* führt der Aufruf der Methode `LogAction` zu keinem Protokolleintrag.

```

class Log {

    boolean isLogActions()
    void setLogActions(boolean flag)

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            Log log = se.createLog("Testlog");
            log.openNotesLog ( "testserver", "log.nsf" );

            log.logAction ( "Aktion1" );
            log.setLogActions ( false );
            log.logAction ( "Aktion2" ); // wird nicht protokolliert
            log.setLogActions ( true );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
};

```

Listing 4.132: Beispiel Log.LogActions

4.10.3 Fehler protokollieren

Für das Protokollieren von Fehlermeldungen kann die Methode `logError` der Klasse *Log* verwendet werden. Diese Methode erwartet als ersten Parameter den numerischen Fehlercode und als zweiten Parameter den Text der Fehlermeldung.

```
class Log {
    void logError(int code, String text);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
            Log log = se.createLog("Testlog");
            log.openNotesLog ( "testserver", "log.nsf" );

            log.logError ( 30000, "Allgemeine Schmutzverletzung" );
        } catch(Exception e) {
            e.printStackTrace();
        };
    };
};
```

Listing 4.133: Beispiel Log.logError

An- und Abschalten der Protokollierung von Fehlermeldungen

Wie die Protokollierung von Meldungen, kann auch die Protokollierung von Fehlermeldungen an- und abgeschaltet werden. Dazu steht die Eigenschaft `LogErrors` zur Verfügung. Bei dem Wert *false* werden Aufrufe der Methode `LogError` ignoriert. Protokolliert werden die Fehlermeldungen, wenn die Eigenschaft den Wert *true* hat.

```
class Log {
    boolean isLogErrors();
    void setLogErrors(boolean flag);
}

public class Beispiel extends AgentBase {
    public void NotesMain() {
        try {
            Session se = getSession();
```

```

Log log = se.createLog("Testlog");
log.openNotesLog ( "testserver", "log.nsf" );

log.logError ( 30000, "Allgemeine Schmutzverletzung" );

log.setLogErrors ( false );

// Keine Protokollierung
log.logError ( 30001, "Sprezielle Dreckverletzung" );

log.setLogErrors ( false );

} catch(Exception e) {

    e.printStackTrace();

};

};
};

```

Listing 4.134: Beispiel Log.LogErrors

4.10.4 Anzahl der protokollierten Meldungen und Fehler

In der Klasse *Log* gibt es zwei Eigenschaften, die Auskunft geben, wie viele Meldungen und Fehler bereits im aktuellen Protokoll seit der Öffnung protokolliert wurden. Diese beiden Eigenschaften haben die Namen *NumActions* und *NumError*.

```

class Log {

    int getNumActions();
    int getNumErrors();

}

```

4.11 Gliederung

Gliederungen werden zur Strukturierung von Notes Anwendungen eingesetzt. In einer Gliederung stehen Referenzen auf andere Notes Elemente. Für die Javaprogrammierung werden diese Gliederungen durch die Klasse *Outline* repräsentiert.

4.11.1 Erstellen einer Gliederung

Eine Gliederung wird mit der Methode *createOutline* in der Klasse *Database* erstellt. Die Methode erwartet dabei als Parameter den gewünschten Namen der neuen Gliederung. Von der Methode wird dann die neu erzeugte Gliederung zurückgegeben.

```

class Database {

    Outline createOutline ( String name );

}

```

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Outline ol = db.createOutline ( "ErsteGliederung" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.135: *Beispiel Database.createOutline*

4.11.2 Zugriff auf eine bestehende Gliederung

Auf eine bestehende Gliederung kann mit der Methode `getOutline` in der Klasse *Database* zugegriffen werden.

```

class Database {

    Outline getOutline ( String name );

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();

            Outline ol = db.getOutline ( "ErsteGliederung" );

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 4.136: *Beispiel Database.getOutlinet*

4.11.3 Zugriff auf die Elemente der Gliederung

Die Elemente einer Gliederung sind in Instanzen der Klasse *OutlineEntry* gespeichert. Diese Instanzen können durch die Methoden *getFirst* und *getNext* bzw. *getFirstSibling* und *getNextSibling* ausgelesen werden.

Die Methode *getFirst* liest den ersten Eintrag aus der Gliederung aus. Der folgende Eintrag wird durch die Methode *getNext* ermittelt. Der Unterschied zwischen *getFirst* und *getFirstSibling* bzw. *getNext* und *getNextSibling* ist, dass bei dem Zusatz „Sibling“ nur Einträge bearbeitet werden, die in derselben Gliederungsebene stehen.

```
class Outline {  
  
    public OutlineEntry getFirst();  
    public OutlineEntry getNext();  
    public OutlineEntry getFirstSibling();  
    public OutlineEntry getNextSibling();  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {  
  
        try {  
  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
  
            Outline ol = db.getOutline ( "ErsteGliederung" );  
  
            OutlineEntry oe = ol.getFirst();  
  
            while ( oe != null ) {  
                System.out.println ( oe.getLable() );  
                oe = ol.getNextSibling();  
            };  
  
        } catch(Exception e) {  
  
            e.printStackTrace();  
  
        };  
  
    };  
  
};
```

Listing 4.137: *Beispiel Outline.getFirst und Outline.getNextSibling*

4.12 Namenshandling

Die Klasse *Name* dient unter Notes zum Handling von Anwender- oder Servernamen. In diesem Abschnitt wird gezeigt, wie mithilfe dieser Klasse der Name eines Anwenders oder Servers ausgewertet oder bearbeitet werden kann.

4.12.1 Der aktuelle Anwender

Eine Instanz der Klasse *Name* für den aktuellen Anwender bzw. Server einer Notes Sitzung lässt sich durch die Eigenschaft *UserNameObject* ermitteln. Diese Eigenschaft liefert nach ihrem Aufruf eine Referenz auf die Instanz für den aktuellen Anwender zurück.

```
class Session {  
  
    Name.getUserNameObject()  
  
}
```

Unter Notes besteht die Möglichkeit, dass ein Anwender oder Server unter weiteren alternativen Namen geführt wird. Die Liste dieser alternativen Namen bekommt man durch die Auswertung der Eigenschaft *UserNameList*. Diese Eigenschaft liefert einen Vektor mit Referenzen auf die Klasse *Name*.

```
class Name {  
  
    java.util.Vector.getUserNameList()  
  
}
```

4.12.2 Erzeugen einer neuen Instanz von Name

Außer einer Referenz für den aktuellen Anwender gibt es auch die Möglichkeit, über die Session eine neue Instanz der Klasse *Name* zu erstellen. Dies erfolgt mit der Methode *createName* der Klasse *Session*.

```
class Session {  
  
    Name createName(String name)  
    Name createName(String name, String lang)  
  
}
```

4.12.3 Der abgekürzte Name

Um für eine Instanz der Klasse *Name* den Namen in abgekürzter Form zu erhalten, kann die Eigenschaft *abbreviated* ausgelesen werden.

```
class Name {  
  
    public String Name.getAbbreviated()  
  
}
```

4.12.4 Der Name in kanonischer Form

Den Namen in kanonischer Form erhält man durch die Eigenschaft `Canonical` der Klasse `Name`. Diese kann über `getCanonical` ausgelesen werden.

```
class Name {  
    String getCanonical()  
}
```

4.12.5 Der allgemeine Name

Die kürzeste Form für einen Namen dürfte die allgemeine Form sein. Diese Form erhält man durch das Auslesen der Eigenschaft `Common`.

```
class Name {  
    String Name.Common()  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            Name user = se.getUserNameObject();  
            System.out.println ( user.getCommon() );  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.138: Beispiel Name.Common

4.12.6 Hierarchische Namen

Ob es sich bei dem Namen um einen Namen mit hierarchischer Struktur handelt, verrät die Eigenschaft `Hierarchical` der Klasse `Name`. Diese boolesche Eigenschaft kann mit `isHierarchical` abgefragt werden.

```
class Name {  
    boolean isHierarchical()  
}
```

Die einzelnen Bestandteile eines hierarchischen Namen kann durch folgende Eigenschaften ermittelt werden:

Eigenschaftsname	Abkürzung	Bedeutung
ADMD	A	Administrationsdomäne
Common	CN	Allgemein
Country	C	Land
Generation	Q	Generationskennzeichen
Given	G	Given
Initials	I	Initialen
Organization	O	Organisation
OrgUnit1	OU	Unterorganisation
OrgUnit2		
OrgUnit3		
OrgUnit4		
Surname	S	Familienname

Tabelle 4.14: Eigenschaften für die Verarbeitung von hierarchischen Bestandteilen von Namen

```
class Name {
    String Name.getADMD()
    String Name.getCommon()
    String Name.getCountry()
    String Name.getGeneration()
    String Name.getGiven()
    String Name.getInitials()
    String Name.getOrganization()
    String Name.getOrgUnit1()
    String Name.getOrgUnit2()
    String Name.getOrgUnit3()
    String Name.getOrgUnit4()
    String Name.getSurname()
}
```

Um alle Bestandteile des hierarchischen Namens zu bekommen, kann die Eigenschaft `Keyword` ausgelesen werden.

```
class Name {
    String getKeyword()
}
```

4.12.7 Die Sprache

Um herauszufinden, welche Muttersprache eine Person hat, kann die Eigenschaft `language` aus der Klasse `Name` verwendet werden. In dieser Eigenschaft befindet sich die Muttersprache der Person.


```
class Name {  
    String getLanguage();  
}
```

4.13 Newsletter

Bei der Anwendungsentwicklung gibt es viele Situationen, in denen eine bestimmte Auswahl von Dokumenten dem Anwender präsentiert werden soll. Um nicht immer das Rad neu erfinden zu müssen, bietet Notes mit der Klasse *Newsletter* eine fertige Technik für diese Aufgabenstellung an.

4.13.1 Erstellen einer neuen Instanz der Klasse Newsletter

Nachdem z. B. auf Grund einer Volltextsuche eine Dokumentensammlung vorliegt, wird eine neue Instanz der Klasse *Newsletter* erzeugt. Dafür wird die Methode *createNewsletter* in der Klasse *Session* aufgerufen. Die Methode erwartet dabei eine Instanz der Klasse *DocumentCollection*, die die aufzulistenden Dokumente beinhaltet.

```
class Session {  
    Newsletter get createNewsletter();  
}  
  
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
            Database db = ac.getCurrentDatabase();  
            DocumentCollection dc = db.getAllDocuments();  
            Newsletter nl = se.createNewsletter ( dc );  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 4.139: Beispiel *Session.createNewsletter*

4.13.2 Formatierung des Newsletters

Nachdem die Instanz der Klasse *Newsletter* vorliegt, können verschiedene Einstellungen bezüglich der Formatierung gemacht werden.

Angabe der Relevanz der Dokumente

Mit der booleschen Eigenschaft *DoScore* der Klasse *Newsletter* kann bestimmt werden, ob das Übersichtsdokument eine Bewertung der Dokumente bezüglich des Suchkriteriums beinhaltet.

```
class Newsletter {  
  
    boolean isDoScore();  
    void setDoScore(boolean flag);  
  
}
```

Thema der Dokumente

Durch die boolesche Eigenschaft *DoSubject* kann bestimmt werden, ob das Thema der Dokumente im Übersichtsdokument aufgenommen werden soll.

```
class Newsletter {  
  
    boolean isDoSubject();  
    void setDoSubject(boolean flag);  
  
}
```

Welches Feld aus den Dokumenten als Thema in das Übersichtsdokument übernommen werden soll, wird durch die Methode *SubjectItemName* bestimmt.

```
class Newsletter {  
  
    String getSubjectItemName();  
    void setSubjectItemName(String name);  
  
}
```

4.13.3 Erzeugen des Übersichtsdokuments

Das auf dem Newsletter basierende Übersichtsdokument wird durch die Methode *formatMsgWithDoclinks* erstellt. Diese Methode erwartet eine Referenz auf die Datenbank, in der das Dokument erzeugt werden soll.

```
class Newsletter {  
  
    Document formatMsgWithDoclinks(Database db);  
  
}  
  
public class Beispiel extends AgentBase {  
  
    public void NotesMain() {
```

```

try {
    Session se = getSession();
    AgentContext ac = se.getAgentContext();
    Database db = ac.getCurrentDatabase();
    DocumentCollection dc = db.getAllDocuments();
    Newsletter nl = se.createNewsletter ( dc );
    nl.SetDoScore ( true );
    nl.SetDoSubject ( true );
    nl.setSubjectItemName ( "titel" );
    nl.formatMsgWithDoclinks ( db );

} catch(Exception e) {

    e.printStackTrace();

};

};

};

```

Listing 4.140: *Beispiel Newsletter.formatMsgWithDoclinks*

4.13.4 Auswahl eines Dokuments im Newsletter

Wenn ein Anwender ein auf ein Newsletter basierendes Übersichtsdokument ausgegeben bekommt, kann es notwendig sein, ein bestimmtes Dokument aus der Aufstellung des Newsletters auszugeben. In dieser Situation kann die Methode `formatDocument` verwendet werden. Diese Methode erwartet als Parameter eine Positionsangabe für ein Dokument aus dem Newsletter. Nach dem Aufruf der Methode wird eine Kopie dieses Dokuments erzeugt und zurückgeliefert.

```

class Newsletter {

    Document formatDocument(Database db, int index);

}

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();
            Database db = ac.getCurrentDatabase();
            DocumentCollection dc = db.getAllDocuments();
            Newsletter nl = se.createNewsletter ( dc );
            for ( int i=1; i<=dc.Count(); i++ ) {
                Document doc = nl.formatDocument ( db, i );
                doc.appendItemValue ( "Form", "Memo" );
                doc.appendItemValue ( "Subject", "Newsletter Report" );
            }
        }
    }
}

```

```
        doc.send ( false, session.getUserName() );  
    };  
  
    } catch(Exception e) {  
        e.printStackTrace();  
    };  
  
};  
  
};
```

Listing 4.141: Beispiel Newsletter.formatDocument

5 JDBC

Auch wenn viele Anwender von Notes der Überzeugung sind ein Allheilmittel in Händen zu halten, so dürfte bald die Ernüchterung kommen. Auch Notes hat seine Grenzen. Häufig wird Notes als Datenbanksystem missverstanden. Der Versuch, Notes als Datenbanksystem zu missbrauchen, stößt schnell an seine Grenzen. So gut wie Notes mit unstrukturierten Datenmengen zurecht kommt, so groß sind die Probleme bei großen Mengen von strukturierten Daten. In so einem Fall ist ein traditionelles, relationales Datenbanksystem nicht zu schlagen.

In vielen Anwendungsfällen wird eine Kombination von unstrukturierten und strukturierten Daten benötigt. Die Aufgabe, die dann häufig besteht, ist die Verknüpfung eines relationalen Datenbanksystems mit dem Notes System. Um diese Aufgabe zu lösen, gibt es in Java den JDBC-Standard.

Der JDBC-Standard stellt einheitliche Klassen zur Verfügung, um Datenbankzugriffe zu unterschiedlichen relationalen Datenbanksystemen zu gewährleisten. Der Vorteil dabei ist, dass bei der Programmierung eine einheitliche Schnittstelle genutzt werden kann. Eine ähnliche Technik ist unter Windows mit ODBC verfügbar.

Keine professionelle Anwendung kommt heutzutage ohne eine Verbindung zu einem Datenbanksystem aus. Um für unterschiedliche Datenbanksysteme eine einheitliche Schnittstelle nutzen zu können, wurde der JDBC-Standard eingeführt. Der JDBC-Standard wird von Sun herausgegeben. JDBC steht dabei für Java Database Connectivity. JDBC ist entworfen worden, um mit relationalen Datenbanksystemen zu kommunizieren.

5.1 SQL

Die Sprache SQL wird für die Abfrage und die Manipulation der Daten sowie für die Definition der Datenstruktur genutzt. SQL ist die Abkürzung von Structured Query Language. Nahezu alle relationalen Datenbanksysteme nutzen SQL für den Zugriff auf Daten und Struktur.

5.1.1 Abfrage

Für die Abfrage von Daten wird in SQL die Select-Anweisung verwendet.

```
SELECT
    Ausdruck1, Ausdruck2, ...
FROM
    Tabelle1, Tabelle2, Tabelle3
WHERE
    Bedingung
```

Die Select-Anweisung lässt sich in drei Teile zerlegen. Der erste Teil beschreibt, welche Daten ausgegeben werden sollen. Im zweiten Teil wird festgelegt, aus welchen

Tabellen die Daten ausgelesen werden sollen. Im letzten Teil kann ein Filterkriterium angegeben werden, welches die Ausgabezeilen selektiert.

Um z. B. aus einer Tabelle *Person* die Spalten *Vorname*, *Nachname* und *Telefon* von allen Personen auszugeben, die mit Vornamen „Hans“ heißen, würde folgende Abfrage verwendet werden:

```
SELECT
    Vorname, Nachname, Telefon
FROM
    Person
WHERE
    Vorname = 'HANS'
```

5.1.2 Einfügen

Um Informationen in einer Tabelle zu ändern, wird die INSERT-Anweisung in SQL verwendet.

```
INSERT INTO Tabelle
    ( Spalte1, Spalte2, Spalte3, ... )
VALUES ( Wert1, Wert2, Wert3, ... )
```

Als Erstes wird der Tabellename der Tabelle erwartet, in der die Daten eingefügt werden sollen. Als Zweites folgt die Liste der Spalten, die mit Daten gefüllt werden sollen. Zuletzt wird bestimmt, dass die Werte in derselben Reihenfolge wie die Spaltennamen aufgeführt werden.

```
INSERT INTO Person
    ( Vorname, Nachname )
VALUES ( 'Hans', 'Meier' )
```

5.1.3 Änderung

Änderungen erfolgen bei SQL durch die UPDATE-Anweisung.

```
UPDATE Tabellenname
    SET Spaltenname = Wert
WHERE Bedingung
```

Zuerst wird der Tabellename der Tabelle angegeben, in der die Änderung durchgeführt werden soll. Danach folgt mit dem Spaltennamen und dem zu schreibenden Wert eine Wertzuweisung. Die Bedingung am Ende der Anweisung gibt Auskunft darüber, für welche Zeilen in der Tabelle die Änderung durchgeführt wird.

```
UPDATE Person
    SET Telefon = '911'
WHERE Vorname = 'Hans' AND Nachname = 'Meier'
```

5.1.4 Datenbankelemente erstellen

Für das Erstellen von Tabellen, Ansichten, Indizes und anderen Datenbankelementen wird die CREATE-Anweisung verwendet. Der Aufbau der CREATE-Anweisung hängt dabei teilweise von der eingesetzten Datenbanksoftware ab.

5.2 Aufbau einer Datenbankverbindung zur Datenabfrage mit JDBC

5.2.1 Schritt1 – JDBC-Treiberklassen für das Datenbanksystem laden

Als Erstes muss bei einer JDBC-Verbindung die Treiberklasse für das jeweilige Datenbanksystem geladen werden. Dies wird mit der statischen Methode `forName` der Klasse `Class` erreicht. Diese Methode erwartet als Parameter den Namen der Klasse, deren Klassendefinition geladen werden soll.

Auf Windows-Systemen gibt es eine besondere Klasse zur Umsetzung eines beliebigen ODBC-Treibers zu einem JDBC-Treiber. Diese Klasse heißt `sub.jdbc.odbc.JdbcOdbcDriver` und wird hier als Beispiel verwendet. Mit ihr kann jedes Datenbanksystem genutzt werden, für das ein ODBC-Treiber vorliegt.

Datenbanksystem	Klassenname
Allgemeiner Umsetzer für ODBC	<code>sun.jdbc.odbc.JdbcOdbcDriver</code>
Lotus Notes	<code>lotus.jdbc.domino.DominoDriver</code>

Tabelle 5.1: Datenbanksysteme und Namen der JDBC-Treiber

```
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
  
            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
};  
  
};
```

Listing 5.1: Beispiel JDBC – Schritt1

5.2.2 Schritt2 – Verbindung zum Datenbanksystem aufbauen

Als zweiten Schritt muss die Datenverbindung zu einem Datenbanksystem aufgebaut werden. Diese Verbindung wird mithilfe der Klasse `DriverManager` aufgebaut. Diese Klasse besitzt eine statische Methode `getConnection`, die als Parameter die Adresse des Datenbanksystems in Form einer URL erwartet. Außerdem können der Methode ein

Benutzername und ein Benutzerkennwort für die Datenbankverbindung angegeben werden. Am Ende des Programms sollte die Datenbankverbindung wieder geschlossen werden. Dafür wird die Methode `close` der Klasse *Connection* aufgerufen.

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );
            String cStr = "jdbc:odbc:Testdatenbank";
            Connection con = DriverManager.getConnection (
                cStr, "", "" );

            con.close();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 5.2: Beispiel JDBC – Schritt2

5.2.3 Schritt3 – Datenabfrage ausführen

Wenn erfolgreich eine Verbindung zu dem Datenbanksystem geknüpft werden konnte, kann als Nächstes ein Statement erzeugt werden. Dieses Statement wird dann im weiteren Verlauf genutzt, um die Abfrage an den Datenbankserver zu senden. Das Statement wird erzeugt, indem die Methode `createStatement` der Klasse *Connection* verwendet wird. Auch das Statement sollte, wie *Connection*, nach dem Gebrauch wieder geschlossen werden. Auch hier wird die Methode `close` verwendet.

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );
            String cStr = "jdbc:odbc:Testdatenbank";
            Connection con = DriverManager.getConnection (
                cStr, "", "" );
```



```

        Statement stmt = con.createStatement();

        stmt.close();
        con.close();

    } catch(Exception e) {

        e.printStackTrace();

    };

};

};

```

Listing 5.3: Beispiel JDBC – Schritt3

5.2.4 Schritt4 – Datenänderung ausführen

Jetzt kann die eigentliche Abfrage an den Server geschickt werden. Dies erfolgt mit der Methode `executeQuery` in der Klasse *Statement*. Diese Methode liefert als Rückgabewert das Ergebnis der Abfrage in einer Instanz der Klasse *ResultSet*.

```

public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );
            String cStr = "jdbc:odbc:Testdatenbank";
            Connection con = DriverManager.getConnection (
                cStr, "", "" );
            Statement stmt = con.createStatement();

            String sql = "SELECT * FROM Person";
            ResultSet set = stmt.executeQuery(sql);

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};

```

Listing 5.4: Beispiel JDBC – Schritt4

5.2.5 Schritt5 – Auslesen des Abfrageergebnisses

Das zurückgelieferte *ResultSet* kann mit einer Schleife durchlaufen werden. Dabei wird bei jedem Schleifendurchlauf eine Zeile des Ergebnisses bearbeitet. Auch das Ergebnis in der Instanz der Klasse *ResultSet* sollte am Programmende ordentlich beseitigt werden. Dies erreicht man durch den Aufruf der Methode *close*.

```
public class Beispiel extends AgentBase {

    public void NotesMain() {

        try {

            Session se = getSession();
            AgentContext ac = se.getAgentContext();

            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );
            String cStr = "jdbc:odbc:Testdatenbank";
            Connection con = DriverManager.getConnection (
                cStr, "", "" );
            Statement stmt = con.createStatement();

            String sql = "SELECT * FROM Person";
            ResultSet set = stmt.executeQuery(sql);
            while ( set.next() ) {
                System.out.println ( set.getString("Vorname"));
            };

            set.close();
            stmt.close();
            con.close();

        } catch(Exception e) {

            e.printStackTrace();

        };

    };

};
```

Listing 5.5: Beispiel JDBC – Schritt5

5.2.6 Aufbau einer Datenbankverbindung zur Manipulation von Daten mit JDBC

Im vorherigen Abschnitt wurde gezeigt, wie *SELECT*-Abfragen in Java mit JDBC an das Datenbanksystem geschickt werden. Alle anderen SQL-Anweisungen werden ähnlich eingesetzt. Nur bei den Anweisungen, die nicht *SELECT*-Anweisungen sind, wird nicht mit der Methode *executeQuery* eine Abfrage gestartet. Bei diesen Anweisungen wird die Methode *execute* in der Klasse *Statement* aufgerufen.

```
public class Beispiel extends AgentBase {  
    public void NotesMain() {  
        try {  
            Session se = getSession();  
            AgentContext ac = se.getAgentContext();  
  
            Class.forName ( "sub.jdbc.odbc.JdbcOdbcDriver" );  
            String cStr = "jdbc:odbc:Testdatenbank";  
            Connection con = DriverManager.getConnection (   
                cStr, "", "" );  
            Statement stmt = con.createStatement();  
  
            String sql = "UPDATE Person SET Vorname = 'Hans'";  
            stmt.execute (sql);  
  
            stmt.close();  
            con.close();  
  
        } catch(Exception e) {  
            e.printStackTrace();  
        };  
    };  
};
```

Listing 5.6: Beispiel JDBC-Anweisung

5.3 JDBC-Treiber für Notes

Eine interessante, aber teilweise nicht bekannte Möglichkeit ist der JDBC-Treiber für den Zugriff auf den Notes Server. Von den IBM Internetseiten kann der Notes JDBC-Treiber heruntergeladen werden. Mit diesem Treiber ist es dann möglich, auf die Notes Datenbanken genauso zuzugreifen wie auf relationale Datenbanken mit ihren JDBC-Treibern. Die URL-Adresse für eine Notes Verbindung ist z. B. *jdbc:domino:/names.nsf*.

6 HTML

Wenn vom Internet gesprochen wird, denkt jeder zuerst an einen Internetbrowser mit HTML-Seiten. Der Name HTML bedeutet Hypertext Mark-up Language. HTML ist die Sprache, mit der Webseiten gestaltet werden. Eine HTML-Seite besteht dabei aus ASCII-Text mit bestimmten Syntaxregeln.

Der HTML-Code wird z. B. genutzt, um Notes Seiten über den http-Server-Task in einem Webbrowser darzustellen. Des Weiteren erzeugen auch die Servlets HTML-Code zur Anzeige der Webseiten.

Die Syntax von HTML basiert dabei auf Tags. Ein Tag ist eine Markierung, die den Anfang und das Ende eines bestimmten Textbereichs markiert. Das erste und das letzte Zeichen eines HTML-Tags sind spitze Klammern, die den Ausdruck umschließen. Der Tag, der am Ende des Bereichs steht, fängt dabei in Klammern immer mit einem „/“-Zeichen an. Z. B. markiert der HTML-Tag die komplette HTML-Seite.

```
<HTML> </HTML>
<HTML>Hier steht der Seitentext</HTML>
```

6.1 Aufbau der Seite

Eine HTML-Seite lässt sich in zwei Bereiche unterteilen: den Kopf- (Head) und den Körperbereich (Body). Der Kopfbereich beinhaltet allgemeine Daten über die Seite wie Titel, Autor oder z. B. Copyright. Der Seitenkörper beinhaltet den eigentlichen Seitentext.

Um diese beiden Bereiche zu kennzeichnen, gibt es den Head-Tag und den Body-Tag. Der Head-Tag umschließt den Seitenkopf und der Body-Tag den Seitenkörper. Dabei befinden sich beide Bereiche innerhalb des HTML-Tags.

```
<HTML>
  <HEAD>
    Kopfinformationen
  </HEAD>
  <BODY>
    Seitenkörper
  </BODY>
</HTML>

<HTML>
  <HEAD>
    Hier stehen die allgemeinen Seiteninformationen
  </HEAD>
  <BODY>
    Hier steht der Seitenkörper
  </BODY>
</HTML>
```

6.2 Die Titelzeile der Seite

Eine HTML-Seite hat im Regelfall eine Titelzeile. Diese Titelzeile wird dabei im Kopfbereich der Seite definiert. Zum Definieren der Titelzeile wird dabei der Tag `TITLE` verwendet.

```
<TITLE>
  Titelzeile
</TITLE>
<HTML>
  <HEAD>
    <TITLE>Titelzeile</TITLE>
  </HEAD>
  <BODY>
    Hier steht der Seitenkörper
  </BODY>
</HTML>
```

6.3 Überschriften

Für die Überschriften im Text stehen die HTML-Tags `H1`, `H2`, `H3`, `H4`, `H5` und `H6` zur Verfügung. Durch diese Tags werden die Überschriften automatisch mit Absatz- und Zeichenformatierung versehen.

```
<H1> Überschrift1 </H1>
<H2> Überschrift1 </H2>
<H3> Überschrift1 </H3>
<H4> Überschrift1 </H4>
<H5> Überschrift1 </H5>
<H6> Überschrift1 </H6>
<HTML>
  <HEAD>
    <TITLE>Titelzeile</TITLE>
  </HEAD>
  <BODY>

    <H1>1. Überschrift</H1>
      <H2>1.1. Überschrift</H2>
        <H3>1.1.1. Überschrift</H3>
        <H3>1.1.2. Überschrift</H3>
      <H2>1.2. Überschrift</H2>
    <H1>2. Überschrift</H1>
      <H2>2.1. Überschrift</H2>
      <H2>2.2. Überschrift</H2>
      <H2>2.3. Überschrift</H2>
    <H1>3. Überschrift</H1>

    Hier steht der Seitenkörper

  </BODY>
</HTML>
```

6.4 Hyperlinks

Die allseits bekannten Links in einer Webseite werden durch Hyperlinks im HTML-Code realisiert. Diese Hyperlinks sind Tags, die mit der Zeichenfolge „A“ beginnen. Des Weiteren wird ein Parameter *HREF* erwartet, der angibt, wohin gesprungen werden soll. Als Ziel kann eine absolute oder relative Adresse auf dem eigenen oder einem fremden Server angegeben werden. Dem folgt dann der Text, der angezeigt werden soll.

```
<A HREF= Zieladresse > Anzeigetext </A>
<HTML>
  <HEAD>
    <TITLE>Titelzeile</TITLE>
  </HEAD>
  <BODY>
    <A HREF=http://www.google.de/index.html> Google </A><BR>
    <A HREF=/index.html > Home </A>
  </BODY>
</HTML>
```

6.5 Formulare

Auch Webseiten in Form von Formularen werden mit dem HTML-Code modelliert. Dazu wird ein `FORM`-Tag verwendet. Innerhalb des Formularbereichs können dann verschiedene Steuerelemente eingefügt werden.

```
<HTML>
  <HEAD>
    <TITLE>Titelzeile</TITLE>
  </HEAD>
  <BODY>
    <FORM>
      Vorname: <INPUT name=Vorname><BR>
      Nachname: <INPUT name=Nachname><BR>
    </FORM>
  </BODY>
</HTML>
```


7 Servlets

7.1 Was sind Servlets?

Mit dem Erscheinen der ersten HTML-Seiten im Internet kam schnell die Überlegung auf, die Seiten dynamisch zu generieren. Dies kann mit verschiedenen Techniken realisiert werden. Die bekanntesten Techniken sind hier CGI-Skripte mit z. B. Perl, PHP, ASP oder Servlets bzw. JSPs.

Die Programmierung von Servlets erfolgt in Java. Servlets sind Programme, die auf so genannten Application Servern laufen. Application Server sind z. B. WebSphere, Tomcat, BEA und natürlich auch der Domino Server.

Um ein Servlet in Java zu entwerfen, wird eine Javaklasse geschrieben, die vom vordefinierten Klassentyp *HttpServlet* oder *Servlet* abgeleitet wird. Dieser vordefinierte Klassentyp wird durch den Application Server zur Verfügung gestellt. Dabei weichen die beiden Klassen nur geringfügig voneinander ab. Des Weiteren wird hier die Klasse *HttpServlet* verwendet.

Der Application Server verhält sich ähnlich wie ein einfacher Webserver, nur dass bei einer Webanfrage keine statischen HTML-Seiten zurückgegeben, sondern die entsprechenden Servlets ausgeführt werden und diese den Text für die HTML-Seite liefern.

7.2 Wie funktioniert ein Servlet?

Für die Kommunikation zwischen einem Browser (z.B. Internet Explorer, Netscape oder Mozilla) und einem Server wird das http-Protokoll genutzt. Mit diesem Protokoll wird von dem Browser eine bestimmte Seite auf dem Server angefordert.

Dabei kann die Seite vom Browser auf zwei verschiedene Arten angefragt werden. Es gibt die Möglichkeit der GET- und der POST-Anfrage. Die GET-Anfrage wurde eingeführt, um Informationen abzufragen, und die POST-Methode, um Informationen zu senden. Aber das Prinzip der beiden Anfragen ist fast identisch. Sie unterscheiden sich hauptsächlich in der Handhabung der Parameter, die bei der Anfrage übergeben werden können. Für jede der beiden Zugriffsarten steht eine Methode in der Klasse *HttpServlet* zur Verfügung. Die Methoden heißen *doGet* und *doPost* und werden bei Aufruf des Servlets angesprochen.

Um jetzt ein eigenes Servlet zu entwickeln, wird wie oben erwähnt eine neue Klasse von der Klasse *HttpServlet* abgeleitet. Damit diese sich bei einer Anfrage wie gewünscht verhält, werden die Methoden *doGet* oder *doPost* in der abgeleiteten Klasse ersetzt. Diese Methode baut dann die gewünschte Seite mit HTML-Elementen auf.

Beispiel:

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;
```

```

public class HelloWorldServlet extends HttpServlet {
    public void doGet (
        HttpServletRequest req,
        HttpServletResponse res
    ) throws ServletException, IOException {
        res.setContentType ( "text/html" );
        PrintWriter out = res.getWriter();
        out.println ( "<HTML>" );
        out.println ( "<HEAD>" );
        out.println ( "<TITLE>" );
        out.println ( "Hello World Servlet" );
        out.println ( "</TITLE>" );
        out.println ( "</HEAD>" );
        out.println ( "<BODY>" );
        out.println ( "<H1>Hello World</H1>" );
        out.println ( "</BODY>" );
        out.println ( "</HTML>" );
    }
};

```

Listing 7.1: Beispiel Servlet

```

class HttpServlet {

    ...
    public void doGet ( HttpServletRequest req,
        HttpServletResponse res
    ) throws ServletException, IOException;

    public void doPost ( HttpServletRequest req,
        HttpServletResponse res
    ) throws ServletException, IOException;
    ...
}

```

7.3 Zustandslose Servlets

Etwas gewöhnungsbedürftig bei der Servletprogrammierung ist, dass die Servlets selbst zustandslos sind. Das bedeutet, dass das Servlet keine Informationen zwischen den Aufrufen über die Sitzung speichert. Wenn also ein Webbesucher z. B. eine Seite ein zweites Mal anfordert, wird dies vom Servlet nicht erkannt. Informationen, die für eine Sitzung gespeichert werden müssen, werden in anderen Klassen gespeichert.

7.4 Sitzung bzw. Session

Sobald ein Browser eine Verbindung mit einem Application Server aufbaut, wird eine Sitzung, auch Session genannt, auf dem Application Server erzeugt. So eine Session wird für jeden einzelnen Client, der eine Anfrage macht, angelegt. Diese Session ist dann auch der Speicherort für alle Informationen, die zwischen verschiedenen Aufrufen von Servlets gespeichert werden sollen. Die Session besteht so lange auf dem Application Server, bis eine bestimmte Zeit lang keine Anfragen von dem jeweiligen Client mehr registriert werden. Diese Session gilt während dieser

Zeit aber für alle Servlets, die vom Client angefordert werden. Diese Session wird in Java von der Klasse *HttpSession* repräsentiert. Damit alle Aufrufe eines Clients auch der richtigen Session zugeordnet werden können, wird eine SessionID erzeugt und bei den Anfragen automatisch mitgeführt. Nach der ersten Anfrage bekommt der Browser diese SessionID mitgeteilt und schickt diese – ohne dass der Anwender etwas bemerkt – bei allen folgenden Anfragen mit. Hierfür gibt es zwei Möglichkeiten: Als Erstes versucht der Browser, diese SessionID als Cookie zu speichern. Sollte dieses nicht zugelassen sein, so wird er versuchen, diese SessionID bei jeder Anfrage an die URL-Adresse anzuhängen.

7.5 Java Beans

Für die sessionspezifischen Anwendungsdaten bietet es sich an, eine Klasse zu entwerfen. Diese Klassen werden als Java Beans bezeichnet. Für die Gestaltung dieser Beans gibt es bestimmte Richtlinien. Die wichtigsten Richtlinien sind die so genannten Getter- und Setter-Methoden. Das bedeutet, dass auf keine Klassenvariable direkt zugegriffen werden kann, sondern nur auf Methoden.

Der Name und die Parameter der Methode für den Zugriff unterliegen dabei folgenden Regeln:

Datentyp der Eigenschaft	Zugriffsart	Präfix steht vor Eigenschaftsnamen	Parameter
boolean	Lesen	is	keine
boolean	Schreiben	set	boolean
nicht boolean	Lesen	get	keine
nicht boolean	Schreiben	set	Datentyp der Eigenschaft

Tabelle 7.1: Regeln für die Namensgebung von Eigenschaften in einer Java Bean

```
public class MyBean {
    private:
        int farbe;
        boolean sichtbar;
    public:
        void setFarbe ( int pfarbe ) {
            farbe = pfarbe;
        };
        int getFarbe () {
            return farbe;
        };
        void setSichtbar ( boolean psichtbar ) {
            sichtbar = psichtbar;
        };
        boolean isSichtbar () {
            return sichtbar;
        };
};
```

Listing 7.2: Beispiel Bean

7.6 Die Klasse `HttpServletRequest`

Wie vielleicht schon bemerkt, werden beim Aufruf der `doGet`- und der `doPost`-Methode zwei Parameter übergeben. Der erste Parameter ist vom Typ `HttpRequest`. In ihm werden Informationen für die Verarbeitung der Anfrage mitgeliefert.

Die Klasse `HttpServletRequest` besitzt verschiedene Methoden, um Informationen über die Anfrage auszulesen. Im Folgenden sind die wichtigsten Methoden erklärt.

7.6.1 Die aktuelle Sitzung

Die wohl wichtigste Methode der Klasse `HttpServletRequest` ist die Methode `getSession`. Diese Methode liefert eine Instanz der Klasse `HttpSession` zurück. Über diese Instanz kann auf die sessionspezifischen Daten zugegriffen werden. Der Aufbau der Klasse `HttpSession` wird weiter unten besprochen.

```
class HttpServletRequest {  
    HttpSession getSession();  
}
```

7.6.2 Die Parameter

Bei jeder Anfrage können in der URL-Adresse Parameter angegeben werden. Die Namen der angegebenen Parameter können mit der Methode `getParameterNames` ermittelt werden. Diese Methode liefert die Namen in einer Enumeration zurück.

```
class HttpServletRequest {  
    Enumeration getParameterNames();  
}
```

Der Wert der einzelnen Parameter wird entweder mit der Methode `getParameter` oder mit `getParameterValues` ausgelesen. Dabei muss der Name des Parameters angegeben werden. Als Rückgabewert bekommt man bei der Methode `getParameterValues` einen Array von Strings.

```
class HttpServletRequest {  
    String[] getParameterValues( String pname );  
}
```

Die Methode `getParameter` liefert einen String zurück.

```
class HttpServletRequest {  
    String getParameter( String pname );  
}
```

7.6.3 Cookies

Mit der Methode `getCookies` können Cookies ausgelesen werden, welche bei der Anfrage vom Browser mitgeschickt worden sind. Der Rückgabewert dieser Methode ist ein Array mit Elementen, die den Datentyp `Cookie` besitzen.

```
class HttpServletRequest {  
    Cookie[] getCookies();  
}
```

7.6.4 Die Anfrageart

Eine weitere Methode ist die `getMethod`, die Auskunft darüber gibt, ob es sich bei der Anfrage um eine `POST`- oder `GET`-Anfrage handelt. Die Methode liefert als Rückgabewert einen String mit der Bezeichnung `POST` oder `GET`.

```
class HttpServletRequest {  
    String getMethod();  
}
```

7.6.5 Die angeforderte Adresse

Die Methode `getPathInfo` gibt den Pfad für den Aufruf des Servlets im Browser als URL-Adresse an. Der Pfad wird als String zurückgegeben.

```
class HttpServletRequest {  
    String getPathInfo();  
}
```

7.6.6 SessionID

Die SessionID der Anfrage erfährt man durch die Methode `getRequestedSessionId`. In bestimmten Situationen kann es passieren, dass diese ID nicht die aktuelle SessionID ist, weil z. B. die angeforderte Session inzwischen auf Grund eines Timeouts nicht mehr gültig ist. Die SessionID wird als String geliefert.

```
class HttpServletRequest {  
    String getRequestedSessionId();  
}
```

Ob die angeforderte Sitzung noch gültig ist, verrät die Methode `isRequestedSessionIdValid`. Sie liefert `true`, wenn die Sitzung noch gültig ist.

```
class HttpServletRequest {  
    boolean isRequestedSessionIdValid();  
}
```

7.7 Die Klasse `HttpServletResponse`

Bei dem zweiten Parameter der Methoden `doGet` oder `doPost` wird der Datentyp `HttpServletResponse` genutzt. Dieser Parameter beinhaltet die Referenz auf eine Instanz der Klasse *HttpServletResponse*, die dazu verwendet wird, zusätzliche Informationen über die Seite an den Browser zu schicken.

7.7.1 Der Ausgabestream

Die wohl am häufigsten genutzte Methode der Klasse *HttpServletResponse* ist die Methode `getWriter`. Mit dieser Methode nämlich wird ein Stream erzeugt, der genutzt wird, um die zu erstellende HTML-Seite auszugeben. Am Anfang der `doGet` oder `doPost`-Methode wird in den meisten Fällen dieser Stream ermittelt, um dann alle Ausgaben für die Seite in diesen Stream zu schreiben. Die Methode liefert einen Stream vom Typ `PrintWriter` zurück.

```
class HttpServletResponse {  
    PrintWriter getWriter();  
}
```

7.7.2 Cookies

Mit der Methode `addCookie` werden neue Cookies an den Browser zurückgegeben. Die Methode erwartet als Parameter eine Referenz auf die Klasse *Cookie*.

```
class HttpServletResponse {  
    void addCookie();  
}
```

7.7.3 Weiterleitung

Eine weitere interessante Methode ist `sendRedirect`. Mit ihr kann eine Umleitung auf den Client ausgelöst werden. Die Methode erwartet als Parameter einen String mit der Adresse, zu welcher der Browser umgeleitet werden soll.

```
class HttpServletResponse {  
    void sendRedirect ( String url );  
}
```

7.7.4 http-Statuscode

Durch die Methode `setStatus` ist es möglich, einen http-Statuscode an den Browser zu schicken. Außerdem kann optional eine Nachricht mitgeschickt werden.

```

class HttpServletResponse {
    void setStatus ( int stcode );
    void setStatus ( int stcode, String stmsg );
}

```

Die folgende Tabelle beinhaltet einen Auszug von standardisierten Statuscodes. Die Statuscodes liegen als Konstante vor.

Statuscode	Wert	Bedeutung
SC_CONTINUE	100	Erster Teil der angeforderten Seite eingetroffen. Weitere Teile können vom Client angefordert werden.
SC_SWITCHING_PROTOCOLS	101	Server akzeptiert http-Versionswechsel.
SC_OK	200	OK
SC_NO_CONTENT	204	Für die Anfrage gibt es keine Seite.
SC_RESET_CONTENT	205	Die Seite muss neu geladen werden, um den Inhalt zu reinitialisieren.
SC_MULTIPLE_CHOICE	300	Die Anfrage war nicht eindeutig. Es gibt mehrere Elemente, die durch die angefragte URL-Adresse angesprochen werden.
SC_MOVED_PERMANENTLY	301	Die angeforderte Seite ist umgezogen.
SC_MOVED_TEMPORARILY	302	Die angeforderte Seite ist vorübergehend umgezogen.
SC_SEE_OTHER	303	Es wurde die falsche Anfrageart verwendet, z. B. POST GET statt GET POST.
SC_NOT_MODIFIED	304	Die angeforderte Seite wurde seit der letzten Anfrage nicht verändert. Das Datum der letzten Anfrage kann bei der Anfrage des Browsers angegeben werden.
SC_USE_PROXY	305	Die angeforderte Adresse muss über einen Proxy angefordert werden.
SC_BAD_REQUEST	400	Die Anfrage wurde vom Server nicht verstanden.
SC_UNAUTHORIZED	401	Für die angeforderte Seite liegt keine Berechtigung vor.
SC_FORBIDDEN	403	Der Zugriff wurde verweigert.
SC_NOT_FOUND	404	Die angeforderte Seite wurde nicht gefunden.
SC_REQUEST_TIMEOUT	408	Timeout während der Anfrage
SC_INTERNAL_SERVER_ERROR	500	Interner Fehler auf dem Server
SC_NOT_IMPLEMENTED	501	Nicht implementiert
SC_SERVICE_UNAVAILABLE	503	Der angeforderte Dienst ist vorübergehend nicht erreichbar.
SC_HTTP_VERSION_NOT_SUPPORTED	505	Die geforderte http-Version wird nicht unterstützt.

Tabelle 7.2: Statuscode für HTTP-Anfragen

7.8 Die Klasse HttpSession

In der Klasse *HttpSession* werden alle sessionspezifischen Informationen gespeichert. Eine Referenz auf die Instanz der Klasse *HttpSession* für die aktuelle Session wird durch die Methode *getSession* der Klasse *HttpServletRequest* zurückgegeben.

7.8.1 Sessionspezifische Daten

Die wohl am häufigsten genutzten Methoden der Klasse sind *getValue*, *putValue* und *removeValue*. Mit diesen Methoden können beliebige Information für die Sitzung gespeichert, gelesen oder wieder gelöscht werden. Zum Speichern der Information wird die Methode *putValue* verwendet. Sie erwartet zwei Parameter: zuerst den Namen der Information, die gespeichert werden soll und als Zweites die Information selbst. Mit dem angegebenen Namen wird die Information identifiziert. Der Name muss innerhalb der Session eindeutig sein. Die Information kann in einer beliebigen Javaklasse gespeichert werden.

```
class HttpSession {  
  
    void putValue ( String pname, Object pvalue );  
  
}
```

Ausgelesen wird die Information mit der Methode *getValue*. Diesem muss als Parameter der Name der gewünschten Information übergeben werden. Zurückgeliefert wird eine Referenz auf das gespeicherte Objekt mit der Information. Der Rückgabewert muss dann im Regelfall gecastet werden.

```
class HttpSession {  
  
    Object getValue ( String pname );  
  
}
```

Um eine gespeicherte Information wieder zu löschen, wird die Methode *removeValue* verwendet. Diese Methode erwartet den Namen der zu löschenden Information.

```
class HttpSession {  
  
    void removeValue ( String pname );  
  
}
```

7.8.2 Information über den Client

Über die Methode *getRemoteAddr* ist es möglich, die IP-Adresse des Clients zu ermitteln. Die Adresse wird als String zurückgegeben.

```
class HttpSession {  
  
    String getRemoteAddr();  
  
}
```


Die Methode `getRemoteHost` liefert den Rechnernamen des Clients als String.

```
class HttpSession {  
    String getRemoteHost();  
}
```

7.8.3 Informationen über den Server

Den Namen des Servers erhält man von der Methode `getServerName` als String zurück.

```
class HttpSession {  
    String getServerName();  
}
```

Den Port, auf dem die Anfrage an den Server gerichtet wurde, verrät die Methode `getServerPort`.

```
class HttpSession {  
    String getServerPort();  
}
```

7.8.4 Allgemeine Informationen über die Session

Das Erstelldatum einer Session verrät die Methode `getCreationTime` in Form eines long Werts.

```
class HttpSession {  
    long getCreationTime();  
}
```

Die SessionID kann über die Methode `getId()` ermittelt werden.

```
class HttpSession {  
    String getId();  
}
```

Wann der letzte Zugriff für die aktuelle Session erfolgt ist, verrät die Methode `getLastAccessTime`.

```
class HttpSession {  
    long getLastAccessedTime();  
}
```

Ob der Client eine neu erzeugte Session schon genutzt hat, verrät die Methode `isNew`. Wenn seit dem Erzeugen der Session der Client bezugnehmend auf die SessionID

noch keine Anfrage an den Server gerichtet hat, liefert die Methode *true*, ansonsten *false*.

```
class HttpSession {  
    boolean isNew();  
}
```

7.8.5 Zerstören einer Session

Außer durch einen Timeout für die Session kann diese auch mit der Methode *invalidate* als ungültig definiert werden. Wenn ein Client dann versucht, auf diese Session Bezug zu nehmen, wird eine neue Session erzeugt.

```
class HttpSession {  
    void invalidate();  
}
```

7.9 Die Klasse Cookie

Die Klasse *Cookie* wird genutzt, um Cookies zu generieren, die auf dem Client gespeichert werden sollen, um auf Informationen auch bei späteren Besuchen der Website zurückgreifen zu können. Ein Cookie wird dabei durch eine Instanz der Javaklasse *Cookie* repräsentiert. Angelegt wird die Referenz durch Erzeugen einer neuen Instanz der Klasse *Cookie* und der Methode *addCookie* von *HttpServletResponse*. Ein bestehendes Cookie wird dann mit der Methode *getCookie* in *HttpServletRequest* gespeichert.

Der Konstruktor der Klasse *Cookie* erwartet zwei Parameter: zuerst den Namen des Cookies und dann den zu speichernden Inhalt.

```
class Cookie {  
    Cookie ( String pname, String pvalue );  
}
```

Eine häufig verwendete Methode der Klasse *Cookie* ist die Methode *setMaxAge*. Diese Methode wird verwendet, um festzulegen, wie lange ein Cookie aufgehoben werden soll. Dabei gibt ein positiver Wert an, wie viele Sekunden der Cookie gespeichert werden soll. Der Wert 0 veranlasst den Client, den Cookie sofort zu löschen. Bei einem negativen Wert wird der Cookie bis zum Ende der aktuellen Session gespeichert und danach gelöscht. Als Standardwert wird ein negativer Wert genommen.

```
class Cookie {  
    void setMaxAge ( int expiry );  
}
```

Eine weitere Methode der Klasse *Cookie* trägt den Namen `setValue`. Mit dieser Methode kann nach dem Erstellen des Cookies der Wert geändert werden.

```
class Cookie {  
    void setValue ( String newValue );  
}
```

7.10 Das erste Servlet

Im Folgenden soll hier ein einfaches Servlet entstehen, das eine HTML-Seite ausgibt.

7.10.1 Schritt1 – Neue Klasse erstellen und vom *HttpServlet* ableiten

Zuerst wird eine neue Klasse benötigt, die von der Klasse *HttpServlet* abgeleitet sein muss.

```
import java.io.*;  
import javax.servlet.* ;  
import javax.servlet.http ;  
  
public class HelloWorldServlet  
    extends HttpServlet {  
};
```

Listing 7.3: Beispiel Servlet – Schritt1

7.10.2 Schritt2 – Methode für GET oder POST schreiben

Der nächste Schritt ist die Überlegung, ob eine Routine für eine GET- oder für eine POST-Anfrage benötigt wird. Es ist auch möglich, für beide Anfragearten unterschiedliches Verhalten in ein und demselben Servlet zu integrieren. In einem solchen Fall werden beide Methoden (`doGet` und `doPost`) überlagert.

In unserem Beispiel-Servlet soll nur die GET-Anfrage bearbeitet werden. Also muss die Methode `doGet` überlagert werden.

```
import java.io.*;  
import javax.servlet.* ;  
import javax.servlet.http ;  
  
public class HelloWorldServlet  
    extends HttpServlet {  
  
    public void doGet ( HttpServletRequest req,  
        HttpServletResponse res ) {  
  
    };  
};
```

Listing 7.4: Beispiel Servlet – Schritt2

7.10.3 Schritt3 – Anfordern des Ausgabestreams

Um die HTML-Seite auszugeben, wird der Ausgabestream benötigt. Eine Referenz auf diesen liefert der Aufruf der Methode `getWriter` in der als Parameter übergebenen *HttpServletResponse*-Instanz.

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        PrintWriter out = getWriter();
    };

};
```

Listing 7.5: Beispiel Servlet – Schritt3

7.10.4 Schritt4 – Ausgabe des HTML-Codes

Jede HTML-Seite sollte die Tags HTML, HEAD, BODY und TITLE beinhalten. Das Servlet soll jetzt diese Tags ausgeben.

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        PrintWriter out = getWriter();
        out.println ( "<HTML>" );
        out.println ( "<HEAD>" );
        out.println ( "<TITLE>" );
        out.println ( "Hello Word Servlet" );
        out.println ( "</TITLE>" );
        out.println ( "</HEAD>" );
        out.println ( "<BODY>" );
        out.println ( "<H1>Hello World</H1>" );
        out.println ( "</BODY>" );
        out.println ( "</HTML>" );
    };

};
```

Listing 7.6: Beispiel Servlet – Schritt4

7.10.5 Schritt5 – Ermitteln der Adresse des Clients

Als Nächstes soll die IP-Adresse des Client ermittelt werden.

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        PrintWriter out = getWriter();
        out.println ( "<HTML>");
        out.println ( "<HEAD>");
        out.println ( "<TITLE>");
        out.println ( "Hello Word Servlet");
        out.println ( "</TITLE>");
        out.println ( "</HEAD>");
        out.println ( "<BODY>");
        out.println ( "<H1>Hello World</H1>");
        out.println ( "Client IP:" +
            req.getRemoteAddr());
        out.println ( "</BODY>");
        out.println ( "</HTML>");
    };
};
```

Listing 7.7: Beispiel Servlet – Schritt5

7.10.6 Schritt6 – Parameter abfragen

Eine weitere Erweiterung für das Servlet ist nun das Auslesen der bei der Anfrage übergebenen Parameter. Die Parameter werden an die URL-Adresse angehängt. Am Ende der URL-Adresse wird – bevor die Parameter folgen – ein Fragezeichen angehängt. Danach folgt jeweils der Name des Parameters und nach einem Gleichheitszeichen der Parameterwert. Mehrere Parameter werden durch das kaufmännische „und“-Zeichen („&“) getrennt.

Beispiel URL:

`http://www.server.de/index.html?wert1=100&wert2=200&wert3=test`

```
import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        PrintWriter out = getWriter();
        out.println ( "<HTML>");
        out.println ( "<HEAD>");
```

```

        out.println ( "<TITLE>");
        out.println ( "Hello Word Servlet");
        out.println ( "</TITLE>");
        out.println ( "</HEAD>");
        out.println ( "<BODY>");
        out.println ( "<H1>Hello World</H1>");
        out.println ( "Client IP:" + req.getRemoteAdr());
        out.println ( "<H1>Parameter</H1>");
        Enumeration enum = req.getParameterNames();
        while ( enum.hasMoreElements() ) {
            String pname = enum.getNextElement();
            String pvalue = req.getParameter ( pname );
            out.println ( pname + " = " + pvalue + "<BR>" );
        };
        out.println ( "</BODY>");
        out.println ( "</HTML>");
    };

};

```

Listing 7.8: Beispiel Servlet – Schritt6

7.10.7 Schritt7 – Lesen und Schreiben von Sessiondaten

Nun soll das Servlet Sessiondaten lesen und schreiben. Als Beispiel soll ein Zähler fungieren, der bei jedem Aufruf um eins erhöht wird. Um den Wert zu speichern, wird eine Klasse *CBean* als Container für die benutzerdefinierten Sessiondaten angelegt. Eine Instanz dieser Klasse wird für jede Session generiert und bei weiteren Anfragen in einer Session zu Rate gezogen. Um auf dieses Datenelement zuzugreifen, wird eine Referenz auf die Instanz der Klasse *HttpSession* benötigt. Dazu wird die Methode *getSession* von *HttpServletRequest*, dem ersten Parameter der *doGet*-Methode, verwendet. Dann wird mit *getValue* und *putValue* die Referenz der Bean ermittelt. Danach kann mit der selbst geschriebenen Methode *getCounter* der Wert des Counters ermittelt werden.

```

import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class CBean {
    private:
        int counter;
    public:
        CBean() { counter = 1; };
        void inc() { counter++; };
        int getCounter() { return counter; };
};

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        HttpSession ses = req.getSession();

```

```

CBean bean = ses.getValue ( "counter" );
if ( bean == null ) {
    // Wenn bean noch nicht für dieser
    // Session existiert wird sie angelegt
    bean = new CBean();
    ses.putValue ( "counter", bean );
} ;
bean.inc();
PrintWriter out = getWriter();
out.println ( "<HTML>" );
out.println ( "<HEAD>" );
out.println ( "<TITLE>" );
out.println ( "Hello Word Servlet" );
out.println ( "</TITLE>" );
out.println ( "</HEAD>" );
out.println ( "<BODY>" );
out.println ( "<H1>Hello World</H1>" );
out.println ( "Client IP:" + req.getRemoteAdr() );
out.println ( "<H1>Parameter</H1>" );
Enumeration enum = req.getParameterNames();
while ( enum.hasMoreElements() ) {
    String pname = enum.nextElement();
    String pvalue = req.getParameter ( pname );
    out.println ( pname + " = " + pvalue + "<BR>" );
};
out.println ( "<H1>Sessiondaten</H1>" );
out.println ( "Aufruf Nr.:" + bean.getCounter() );
out.println ( "</BODY>" );
out.println ( "</HTML>" );
};
};

```

Listing 7.9: Beispiel Servlet – Schritt 7

7.10.8 Schritt 8 – Verwendung von Cookies

Nun soll das Servlet noch mit Cookies arbeiten. Dazu wird auf dem Client ein Cookie angelegt, in welchem das Datum des letzten Zugriffs gespeichert wird. Damit ist es möglich, bei dem Aufruf des Servlets zu ermitteln, wann das letzte Mal von diesem Client aus ein Zugriff erfolgt ist. Zuerst wird in der Methode versucht, ein bestehendes Cookie zu lesen. Danach wird das Cookie geschrieben.

```

import java.io.*;
import javax.servlet.* ;
import javax.servlet.http ;

public class CBean {
    private:
        int counter;
    public:
        CBean() { counter = 1; };
        void inc() { counter++; };
        int getCounter() { return counter; };
};

```

```

public class HelloWorldServlet
    extends HttpServlet {

    public void doGet ( HttpServletRequest req,
        HttpServletResponse res ) {
        HttpSession ses = req.getSession();
        CBean bean = ses.getValue ( "counter" );
        String last = null;
        if ( bean == null ) {
            // Wenn bean noch nicht für dieser
            // Session existiert wird sie angelegt
            bean = new CBean();
            ses.putValue ( "counter", bean );
        };
        Cookie[] cookies = req.getCookies();
        if ( cookies != null ) {
            for ( int i=0; i<cookies.length; i++ )
                if ( cookies[i].getName().equals("mycookie");
                    last = cookies[i].getValue();
        };
        if ( last == null )
            last = String.valueOf(System.currentTimeMillis());
        Cookie cookie = new Cookie ( "mycookie",
            String.valueOf(System.currentTimeMillis())
        );
        res.addCookie ( cookie );
        bean.inc();
        PrintWriter out = getWriter();
        out.println ( "<HTML>");
        out.println ( "<HEAD>");
        out.println ( "<TITLE>");
        out.println ( "Hello Word Servlet");
        out.println ( "</TITLE>");
        out.println ( "</HEAD>");
        out.println ( "<BODY>");
        out.println ( "<H1>Hello World</H1>");
        out.println ( "Client IP:" + req.getRemoteAdr());
        out.println ( "<H1>Parameter</H1>");
        Enumeration enum = req.getParameterNames();
        while ( enum.hasMoreElements() ) {
            String pname = enum.getNextElement();
            String pvalue = req.getParameter ( pname );
            out.println ( pname + " = " + pvalue + "<BR>" );
        };
        out.println ( "<H1>Sessiondaten</H1>");
        out.println ( "Aufruf Nr.:" + bean.getCounter() + "<BR>" );
        out.println ( "<H1>Cookie</H1>");
        out.println ( "Letzter Aufruf:" + last + "<BR>" );
        out.println ( "</BODY>");
        out.println ( "</HTML>");
    };
};

```

Listing 7.10: Beispiel Servlet – Schritt 8

8 JSP-Programmierung

8.1 Was sind JSPs?

Die Abkürzung JSP steht für Java Server Pages. Java Server Pages sind Textdateien, die in ihrem Aussehen HTML-Seiten sehr ähnlich sind. Bei dem Aufbau von JSP-Dateien werden die HTML-Regeln um die so genannten JSP-Tags erweitert.

JSP-Seiten werden auf Application Servern genutzt. Nicht alle Application Server können JSP-Seiten verarbeiten. Aber immer mehr unterstützen JSPs. Einige Application Server, die JSP unterstützen, sind z. B. WebSphere und Tomcat.

8.2 Der Aufbau von JSP-Seiten

Eine JSP-Seite wird, wie erwähnt, mit Tags aufgebaut. Einige dieser Tags sind Pflicht, andere sind optional. Die JSP-spezifischen Tags werden auf den Seiten zwischen dem normalen HTML-Code platziert. Sie sind sozusagen Platzhalter für den dynamisch generierten HTML-Code.

In einer JSP-Seite kann an beliebiger Stelle auch Javacode platziert werden. Dafür werden verschiedene JSP-Tags vom JSP-Standard zur Verfügung gestellt.

8.3 Was passiert, wenn eine JSP-Seite aufgerufen wird?

Wenn eine JSP-Seite entworfen wird, wird sie auf dem Application Server in einem bestimmten Verzeichniss abgelegt. Wenn dann eine Anfrage mit der entsprechenden Adresse an den Application Server gerichtet wird, wird diese JSP-Seite geladen. Der Application Server wandelt diese JSP-Seite dann in ein Servlet um. Das Servlet wird dann auf dem Application Server gespeichert und danach ausgeführt. Sollte die JSP-Seite ein weiteres Mal angefordert werden, ohne dass sich der JSP-Code ändert, wird das einmal kompilierte Servlet wiederverwendet. Eine JSP ist also nichts weiter als ein vereinfachtes Hilfsmittel, um Servlets zu entwickeln. Man kann sich das in etwa so vorstellen, dass alles, was in der JSP-Seite steht, in der `doGet`- oder `doPost`-Methode des Servlets ausgeführt wird.

8.4 Die JSP-Syntax

8.4.1 Ausdrücke und Direktiven

Bei der Gestaltung von JSP-Seiten wird zwischen Ausdrücken und Direktiven unterschieden. Ein Ausdruck beginnt immer mit „`<%=`“ und endet immer mit „`%>`“. Der Ausdruck zwischen den beiden Markierungen wird ausgewertet und als Text an dieser Stelle in der Seite ausgegeben.

Beispiel:

```
<HTML>
<HEAD><TITLE>Hello World JSP</TITLE></HEAD>
<BODY><%= "Hello" + " " + "World" %></BODY>
</HTML>
```

Im Unterschied zu den Ausdrücken dienen die Direktiven dazu, das Verhalten bei der Verarbeitung der Seite zu beeinflussen. Die Direktiven werden dabei durch „<%“ und „%>“ eingeklammert.

Beispiel:

```
<%@ method = „doPost“ %>
<HTML>
<HEAD><TITLE>Hello World JSP</TITLE></HEAD>
<BODY><%= "Hello" + " " + "World" %></BODY>
</HTML>
```

Das aus dieser JSP generierte Servlet behandelt nur POST-Anfragen.

8.4.2 Javacode

Außer den Ausdrücken und Direktiven kann durch „<%“ und „%>“ an beliebiger Stelle Javacode in der Seite eingefügt werden.

Beispiel:

```
<HTML>
<HEAD><TITLE>Hello World JSP</TITLE></HEAD>
<BODY><% out.println ( "Hello World!" );%>
</HTML>
```

Dieses Beispiel gibt die Zeichenkette Hello World! in der Seite aus. Dabei wird die Variable out verwendet, die innerhalb einer JSP-Seite zur Verfügung steht. Bevor die JSP abgearbeitet wird, werden verschiedene Variablen instanziiert und stehen innerhalb der JSP zur Verfügung.

8.4.3 Kommentare

Wenn in einer HTML-Seite ein Kommentar eingefügt wird, wird er auch an den Browser geschickt. Zwar wird der Kommentar nicht in der Webseite angezeigt, aber der Besucher der Webseite kann sich den Quelltext der HTML-Seite anschauen und den Kommentar so lesen.

Bei der Gestaltung von JSP-Seiten ist es möglich, Kommentare in den Seiten aufzunehmen, die nicht vom Server an den Browser herausgegeben werden. Man könnte diese als serverseitige Kommentare verstehen.

Um einen solchen Kommentar in die JSP-Seite einzufügen, muss der Kommentartext mit „<%--“ und „--%>“ umschlossen werden.

Beispiel:

```
<HTML>
<HEAD><TITLE>Hello World JSP</TITLE></HEAD>
<%-- Dieses Kommentar wird nicht an den Browser geschickt --%>
<BODY><%= "Hello" + " " + "World" %></BODY>
</HTML>
```

8.4.4 Deklarationen

Deklarationen von Variablen können durch die Ausdrücke „<%!“ und „%>“ markiert werden.

Definition:

```
<%! declaration; [ declaration; ] ... %>
```

Beispiel:

```
<%! int i=0; %>
<%! double a=0; %>
```

8.5 Die verschiedenen JSP-Direktiven

8.5.1 Die Page-Direktive

Mit der Page-Direktive können verschiedene Einstellungen für die JSP-Seite vorgenommen werden.

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
[ import="{package.class |
package.*}, ..." ]
[ session="true|false" ]
[ buffer="none|8kb|sizekb" ]
[ autoFlush="true|false" ]
[ isThreadSafe="true|false" ]
[ info="text" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType
[ ; charset=characterSet ]" |
"text/html ; charset=ISO-8859-1" ]
[ isErrorPage="true|false" ]
[ pageEncoding="characterSet |
ISO-8859-1" ]
%>
```

Der Parameter language

Der Parameter `language` wird in den meisten Seiten seinen Standardwert *java* behalten. Er dient dazu, die Scriptsprache der JSP-Seite zu bestimmen. Nur sehr wenige Application Server können eine andere Sprache als Java in den JSPs verarbeiten.

Der Parameter extends

Da die JSP-Seite vom Application Server in ein Servlet umgewandelt wird, ist das Resultat eine Javaklasse. Mit diesem Parameter kann bestimmt werden, von welcher Klasse diese automatisch generierte Klasse abgeleitet wird. Aus der Praxis kann ich nur warnen, diese Einstellung zu leichtfertig zu ändern, da dies in den meisten Fällen mit einigen unschönen Nebeneffekten quittiert wird.

Der Parameter include

Der Wert dieses Parameters ist die Liste der Javapackages, auf welche der Javacode der JSP zurückgreift. Diese Liste ist vergleichbar mit der Import-Anweisung in einem Javaprogramm.

Der Parameter isThreadSafe

Für diesen Parameter kann entweder *true* oder *false* angegeben werden. Dadurch wird definiert, ob die JSP multithread-sicheren Code beinhaltet. Das bedeutet, dieselbe JSP-Seite kann zur gleichen Zeit in zwei getrennten Threads, z. B. bei zwei gleichzeitigen Anfragen, bearbeitet werden.

Der Parameter errorPage

Für diesen Parameter kann eine URL-Adresse angegeben werden, welche die Fehlerseite bestimmt. Diese Fehlerseite wird an den Browser zurückgeliefert, wenn während der Verarbeitung in der JSP-Seite ein Fehler auftritt.

Der Parameter isErrorPage

Durch diesen Parameter kann definiert werden, ob die aktuelle JSP-Seite eine Fehlerseite ist.

Der Parameter session

Für diesen Parameter können die Werte *true* oder *false* verwendet werden. Wenn der Wert *true* gewählt wird, bedeutet dies, dass innerhalb der JSP-Seite auf das Session-Objekt zugegriffen wird. Bei dem Wert *false* führt ein solcher Zugriff bei der Übersetzung der Seite zu einem Fehler.

Der Parameter contentType

Mit diesem Parameter kann der Contenttyp der Seite angegeben werden, z. B. „text/plain“ oder „text/html“.

Der Parameter pageEncoding

Der Zeichensatz, der für die JSP-Seite verwendet werden soll, befindet sich in diesem Parameter.

8.5.2 Die Include-Direktive

Diese Direktive ermöglicht ein serverseitiges Include einer anderen Seite. An der Stelle, an der die Direktive verwendet wird, wird der Code der angegebenen Seite eingefügt.

```
<%@ include file="relativeURL" %>
```

Der Wert des Parameters `file` definiert die URL-Adresse der Seite, die eingefügt werden soll.

8.5.3 Die Taglib-Direktive

Mit dieser Direktive können benutzerdefinierte Tag-Bibliotheken eingebunden werden. Mit dieser Technik ist es möglich, eigene Erweiterungen für die JSP-Programmierung zu entwickeln und diese wie die anderen JSP-Elemente zu nutzen.

```
<%@ taglib uri="URIForLibrary" prefix="tagPrefix" %>
```

Der Parameter `uri`

Dieser Parameter dient zur Bestimmung des Standorts der Tag-Bibliothek (der TLD-Datei). Der Standort kann dabei der eigene oder ein fremder Server sein. Als Adresse kann somit eine `http`-Adresse oder ein absoluter bzw. relativer Pfad angegeben werden.

Der Parameter `prefix`

Der Wert, der für den Parameter `prefix` verwendet wird, dient zur eindeutigen Identifizierung der Tag-Bibliothek in der JSP-Seite. Dieser Wert muss dem jeweiligen JSP-Tag vorangestellt werden.

Verwendung der Taglib-Direktive

Ein Beispiel für die Verwendung einer Tag-Bibliothek ist:

```
<%@ taglib uri="http://www.server.de/tags" prefix="public" %>
<public:loop>
...
</public:loop>
```

8.6 Die verschiedenen JSP-Aktionen

8.6.1 Forwarding von Anfragen

Mit dieser Anweisung kann eine Anfrage weitergeleitet werden. Es wird eine Weiterleitung auf dem Client veranlasst. Das bedeutet, der Client bekommt eine neue URL und fordert diese automatisch an.

```
<jsp:forward page="{relativeURL |
    <%= expression %>}" />
```

oder

```
<jsp:forward page="{relativeURL |
    <%= expression %>}" >
    <jsp:param name="parameterName"
        value="{parameterValue |
            <%= expression %>}" /> +
</jsp:forward>
```

Der Parameter page gibt dabei die Adresse an, zu der umgeleitet werden soll. Außer der Adresse können weitere Parameter angegeben werden. Alle Zeilen nach der forward-Anweisung werden nicht mehr berücksichtigt.

Beispiel:

```
<jsp:forward page="http://www.google.de">
    <jsp:param q="Domino" >
</jsp:forward>
```

8.6.2 Java Beans

Bei der JSP-Programmierung gelten dieselben Richtlinien für die Beans wie bei der Servlet-Programmierung, d. h. dieselben Regeln für die Namensgebung und die Parameter der Getter- und Setter-Methoden.

Hier noch einmal die Regeln für die Namen und die Parameter:

Datentyp der Eigenschaft	Zugriffsart	Präfix steht vor Eigenschaftsnamen	Parameter
boolean	Lesen	is	keine
boolean	Schreiben	set	boolean
nicht boolean	Lesen	get	keine
nicht boolean	Schreiben	set	Datentyp der Eigenschaft

Tabelle 8.1: Regeln für die Namensgebung von Eigenschaften in einer Java Bean

```
public class CMyBean {
    private:
        int farbe;
        boolean sichtbar;
    public:
        void setFarbe ( int pfarbe ) {
            farbe = pfarbe;
        };
        int getFarbe () {
            return farbe;
        };
        void setSichtbar ( boolean psichtbar ) {
            sichtbar = psichtbar;
        };
        boolean isSichtbar () {
            return sichtbar;
        };
};
```

Listing 8.1: Beispiel Bean

Deklarieren einer Bean in der JSP

Bevor eine Bean in der JSP verwendet werden kann, muss sie deklariert werden. Dazu wird die Aktion `jsp:useBean` verwendet. Mit diesem Element kann auf einfachem Weg eine Bean für die JSP-Seite bekannt gemacht werden.

```
<jsp:useBean id="beanInstanceName"
scope="page|request|session|application" {
    class="package.class" [ type="package.class" ] |
    beanName="{package.class | <%= expression %>}"
    type="package.class" | type="package.class"
}
{ /> | >
other elements
</jsp:useBean> }
```

Beispiel:

```
<jsp:useBean
    id="mybean" scope="session"
    class="de.testprog.CMyBean"
>
...
</jsp:useBean>
```

Auslesen der Eigenschaft

Die Eigenschaften einer Bean können durch die Getter-Methoden ausgelesen werden. Dazu gibt es den Ausdruck `jsp:getProperty` in der JSP- Syntax.

Definition:

```
<jsp:getProperty
    name="mybean"
    property="Farbe"
/>
```

Beispiel:

```
<jsp:useBean
    id="mybean" scope="session"
    class="de.testprog.CMyBean"
>
    <jsp:getProperty
        name="mybean"
        property="farbe"
    />
</jsp:useBean>
```

Setzen von Eigenschaftswerten

Das Setzen der Eigenschaften erfolgt durch den Ausdruck `jsp:setProperty`.

Definition:

```
<jsp:setProperty name="beanInstanceName"
{
property="*" |
property="propertyName" [ param="parameterName" ] |
property="propertyName" value="{stringLiteral|
    <%= expression %>}"
}
/>
```

Für den Parameter `name` muss der Name der Beaninstanz eingetragen werden. Dieser Name muss dem Eintrag in der `useBean`-Anweisung entsprechen.

Beispiel:

```
<jsp:useBean
    id="mybean" scope="session"
    class="de.testprog.CMyBean"
>
    <jsp:setProperty
        name="mybean"
        property="farbe"
    />
</jsp:useBean>
```

8.7 Die Tag-Bibliothek von Notes

Eine Neuheit in der jüngsten Version von Notes ist die JSP-Tag-Bibliothek. Diese Bibliothek stellt JSP-Tags für den Zugriff auf das Notes System zur Verfügung.

8.7.1 Installation der Tag-Bibliothek

Um diese Tag-Bibliothek nutzen zu können, muss sie bei dem verwendeten Applikationsserver angemeldet werden.

Schritt1 – Kopieren der Jar-Datei

Zunächst muss die Jar-Datei, in der die Tags definiert sind, kopiert werden. Die Datei befindet sich im Verzeichnis `NOTES/DATA/DOMINO/JAVA` des Notes Servers und muss in das Verzeichnis `WEB-INF/LIB` des Applikationsservers dupliziert werden.

Schritt2 – Kopieren der TLD-Dateien

Als Nächstes müssen die Dateien *domtags.tld* und *domutil.tld* aus dem Verzeichnis `NOTES/DATA/JAVA` in das `WEB-INF`-Verzeichnis des Applikationsservers kopiert werden.

Schritt3 – Editieren der web.xml-Datei

Der nächste Schritt ist das Anmelden der Taglib-Bibliothek beim Applikationssserver durch das Editieren der *web.xml*-Datei.

```
<taglib>
  <taglib-uri>domtags.tld</taglib-uri>
  <taglib-location>/WEB-INF/domtags.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>domutil.tld</taglib-uri>
  <taglib-location>/WEB-INF/domutil.tld</taglib-location>
</taglib>
```

Schritt4 – Kopieren der notes.jar-Datei

Der letzte Schritt ist das Kopieren der *notes.jar*-Datei. Diese Datei muss im Classpath des Applikationsservers stehen. Danach kann die Tag-Bibliothek verwendet werden.

8.8 Übersicht über die Tags der Bibliothek

Die-Tag Bibliothek lässt sich in zwei Bereiche aufteilen. Der erste Teil wird durch Domino Tags und der zweite durch die *dom.util*-Tags gebildet. Die Beschreibung der einzelnen Tags befindet sich in der Referenz.

Hier ein Beispiel:

```
<%@ taglib uri="/WEB-INF/lib/domtags.tld" prefix="domino" %>
<%@ page info="Session Test" language="java"
  import="lotus.domino.*" errorPage="error.jsp" %>

<html><body>

  <h1>Session Test</h1>

  <domino:session id="sinfo">

    <h1>Serverplatform:</h1>
    <% Session se = sinfo; %>
    <h3>Serverplatform = <%=se.getPlatform() %></h3>

  </domino:session>

</body></html>
```


9 Referenz der Notes Javaklassen

9.1 Klasse ACL

9.1.1 Konstanten

Name	Wert	Beschreibung
ACL_MAX_ROLES	80	
LEVEL_AUTHOR	3	Zugriffslevel Autor
LEVEL_DEPOSITOR	1	Zugriffslevel Einlieferer
LEVEL_DESIGNER	5	Zugriffslevel Entwickler
LEVEL_EDITOR	4	Zugriffslevel Editor
LEVEL_MANAGER	6	Zugriffslevel Manager
LEVEL_NOACCESS	0	Zugriffslevel Kein Zugriff
LEVEL_READER	2	Zugriffslevel Leser

9.1.2 Eigenschaften

InternetLevel

Datentyp

Int

Deklaration Getter

int getInternetLevel() throws NotesException

Deklaration Setter

void setInternetLevel(int level) throws NotesException

Wertebereich

ACL.LEVEL_NOACCESS
ACL.LEVEL_DEPOSITOR
ACL.LEVEL_READER
ACL.LEVEL_AUTHOR
ACL.LEVEL_EDITOR
ACL.LEVEL_DESIGNER
ACL.LEVEL_MANAGER

Beschreibung

Diese Eigenschaft bestimmt den maximalen Zugriffslevel für Zugriffe durch einen HTML-Browser. Nach dem Ändern dieser Eigenschaft muss mit der Methode `save` die Änderung gespeichert werden.

IsUniformAccess**Datentyp**

Boolean

Deklaration Getter

`boolean isUniformAccess()` throws `NotesException`

Deklaration Setter

`void setUniformAccess(boolean flag)` throws `NotesException`

Wertebereich

true oder false

Beschreibung

Durch diese Eigenschaft wird bestimmt, ob eine konsistente Zugriffskontrollliste über alle Repliken der Datenbank gefordert wird. Nach dem Ändern dieser Eigenschaft muss die Änderung mit der Methode `save` gespeichert werden.

Parent**Datentyp**

Database

Deklaration Getter

`Database getParent()`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Datenbank, zu der diese Zugriffskontrollliste gehört.

Roles

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getRoles() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Anwenderrollen, die in dieser ACL definiert sind.

9.1.3 Methoden

AddRole

Deklaration

void addRole(String name) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Zugriffsrolle

Beschreibung

Mit dieser Methode können neue Rollen in der Zugriffskontrollliste definiert werden. Dazu muss dieser Methode der Name der neuen Rolle übergeben werden. Danach muss die Zugriffskontrollliste mit der Methode save gespeichert werden.

createACLEntry

Deklaration

ACLEntry createACLEntry(String name, int level) throws NotesException

Rückgabewert

ACLEntry

Der neu erstellte Eintrag der Zugriffskontrollliste

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Anwenders, Servers oder der Gruppe
level	nein	int	Zugriffslevel

Beschreibung

Mit dieser Methode kann ein neuer Eintrag in der Zugriffskontrollliste eingetragen werden. Dazu muss dieser Methode als Parameter der Name des Anwenders, eines Servers oder einer Gruppe übergeben werden. Außerdem wird von der Methode als Parameter der gewünschte Zugriffslevel erwartet. Nach dem Erstellen des Eintrags muss die Änderung noch mit der save-Methode gespeichert werden.

deleteRole

Deklaration

`void deleteRole(String name) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der zu löschenden Rolle

Beschreibung

Mit dieser Methode kann eine definierte Zugriffsrolle aus der Zugriffskontrollliste entfernt werden. Dazu muss der Name der zu entfernenden Zugriffsrolle angegeben werden. Nach dem Aufruf dieser Methode muss die Änderung durch den Aufruf der save-Methode gespeichert werden.

getEntry

Deklaration

`ACLEntry getEntry(String name) throws NotesException`

Rückgabewert

ACLEntry

Der gesuchte Eintrag in der Zugriffskontrollliste

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Anwenders, Servers oder der Gruppe

Beschreibung

Mit dieser Methode kann ein Eintrag aus der Zugriffskontrollliste gelesen werden. Dazu wird als Parameter der Name des Anwenders, eines Servers oder einer Gruppe erwartet. Der entsprechende Eintrag in der Zugriffskontrollliste wird, falls vorhanden, zurückgegeben.

getFirstEntry

Deklaration

ACLEntry getFirstEntry() throws NotesException

Rückgabewert

ACLEntry

Der erste Eintrag der Zugriffskontrollliste

Parameter

keine

Beschreibung

Wenn alle Einträge einer Zugriffskontrollliste gelesen werden sollen, kann mit dieser Methode der erste Eintrag der Zugriffskontrollliste gelesen werden.

getNextEntry

Deklaration

ACLEntry getNextEntry() throws NotesException

ACLEntry getNextEntry(ACLEntry entry) throws NotesException

Rückgabewert

ACLEntry

Der nächste Eintrag in der Zugriffskontrollliste

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ACLEntry	Wenn dieser Parameter angegeben wird, liefert diese Methode den auf dem angegebenen Eintrag folgenden Eintrag. Sollte kein Parameter angegeben sein, so wird der Eintrag zurückgeliefert, welcher dem zurückgelieferten Eintrag beim letzten Aufruf dieser Methode folgte.

Beschreibung

Beim Durchwandern aller Einträge der Zugriffskontrollliste wird mit dieser Methode der nächste Eintrag gelesen.

removeACLEntry

Deklaration

void removeACLEntry(String name) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Anwenders, Servers oder der Gruppe, deren Eintrag aus der Zugriffskontrollliste entfernt werden soll.

Beschreibung

Mit dieser Methode kann ein Eintrag aus der Zugriffskontrollliste entfernt werden. Dazu wird der Name des Anwenders, Servers oder der Gruppe angegeben, für den der Eintrag in der Zugriffskontrollliste steht.

renameRole

Deklaration

void renameRole(String oldname, String newname) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
oldname	nein	String	Alter Name der Rolle
newname	nein	String	Neuer Name der Rolle

Beschreibung

Durch den Aufruf dieser Methode kann ein Name einer Zugriffsrolle der Zugriffskontrollliste geändert werden.

save

Deklaration

void save() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Erst durch den Aufruf dieser Methode werden durchgeführte Änderungen in der Zugriffskontrollliste dauerhaft gespeichert.

9.2 Klasse *ACLEntry*

9.2.1 Konstanten

Name	Wert	Beschreibung
TYPE_MIXED_GROUP	3	Eintragstyp Gemischte Gruppe
TYPE_PERSON	1	Eintragstyp Einzelne Person
TYPE_PERSON_GROUP	4	Eintragstyp Gruppe von Personen
TYPE_SERVER	2	Eintragstyp Server
TYPE_SERVER_GROUP	5	Eintragstyp Gruppe von Servern
TYPE_UNSPECIFIED	0	Eintragstyp undefiniert

9.2.2 Eigenschaften

AdminReaderAuthor

Datentyp

Boolean

Deklaration Getter

boolean isAdminReaderAuthor() throws NotesException

Deklaration Setter

void setAdminReaderAuthor(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob ein Administrationsserver den Inhalt von Autoren- bzw. Leserfeldern ändern darf. Nach dem Ändern dieser Eigenschaft muss die Methode `save` der Klasse *ACL* aufgerufen werden, um die Änderung zu speichern.

AdminServer

Datentyp

boolean

Deklaration Getter

boolean isAdminServer() throws NotesException

Deklaration Setter

void setAdminServer(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft wird definiert, ob der Eintrag in der Zugriffskontrollliste einen Administrationsserver bezeichnet. Nach Änderung dieser Eigenschaft muss mit der Methode *save* der Klasse *ACL* die Änderung gespeichert werden.

CanCreateDocuments

Datentyp

boolean

Deklaration Getter

boolean isCanCreateDocuments() throws NotesException

Deklaration Setter

void setCanCreateDocuments(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft wird bestimmt, ob der Eintrag in der Zugriffskontrollliste es erlaubt, neue Dokumente in der Datenbank zu erstellen. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode *save* gespeichert.

CanCreateLSOrJavaAgent

Datentyp

boolean

Deklaration Getter

boolean isCanCreateLSOrJavaAgent() throws NotesException

Deklaration Setter

void setCanCreateLSOrJavaAgent(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird bestimmt, ob der Eintrag in der Zugriffskontrollliste das Erstellen von Lotus Script oder Java-Agents zulässt. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode *save* der Klasse *ACL* gespeichert.

CanCreatePersonalAgent**Datentyp**

boolean

Deklaration Getter

boolean isCanCreatePersonalAgent() throws NotesException

Deklaration Setter

void setCanCreatePersonalAgent(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob der Eintrag in der Zugriffskontrollliste das Erstellen von persönlichen Agents zulässt. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode *save* der Klasse *ACL* gespeichert.

CanCreatePersonalFolder**Datentyp**

boolean

Deklaration Getter

boolean isCanCreatePersonalFolder() throws NotesException

Deklaration Setter

void setCanCreatePersonalFolder(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird bestimmt, ob der Eintrag in der Zugriffskontrollliste das Erstellen von persönlichen Ordnern zulässt. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode `save` der Klasse *ACL* gespeichert.

CanCreateSharedFolder**Datentyp**

boolean

Deklaration Getter

boolean `isCanCreateSharedFolder()` throws *NotesException*

Deklaration Setter

void `setCanCreateSharedFolder(boolean flag)` throws *NotesException*

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob der Eintrag in der Zugriffskontrollliste das Erstellen von allgemeinen Ordnern zulässt. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode `save` der Klasse *ACL* gespeichert.

CanDeleteDocuments**Datentyp**

boolean

Deklaration Getter

boolean `isCanDeleteDocuments()` throws *NotesException*

Deklaration Setter

void `setCanDeleteDocuments(boolean flag)` throws *NotesException*

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird bestimmt, ob der Eintrag in der Zugriffskontrollliste das Löschen von Dokumenten zulässt. Eine Änderung dieser Eigenschaft wird erst durch den Aufruf der Methode `save` der Klasse *ACL* gespeichert.

Group**Datentyp**

boolean

Deklaration Getter

boolean `isGroup()` throws *NotesException*

Deklaration Setter

void `setGroup(boolean flag)` throws *NotesException*

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft kann ermittelt werden, ob der Eintrag in der Zugriffskontrollliste für eine Anwender-, Server- oder gemischte Gruppe steht.

Person**Datentyp**

boolean

Deklaration Getter

boolean `isPerson()` throws *NotesException*

Deklaration Setter

void `setPerson(boolean flag)` throws *NotesException*

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft kann ermittelt werden, ob der Eintrag in der Zugriffskontrollliste für einen einzelnen Anwender steht.

PublicReader

Datentyp

boolean

Deklaration Getter

boolean isPublicReader() throws NotesException

Deklaration Setter

void setPublicReader(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob der Eintrag in der Zugriffskontrollliste das Lesen von öffentlichen Dokumenten zulässt. Nach einer Änderung dieser Eigenschaft muss die Änderung mit dem Aufruf der Methode `save` gespeichert werden.

PublicWriter

Datentyp

boolean

Deklaration Getter

boolean isPublicWriter() throws NotesException

Deklaration Setter

void setPublicWriter(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob der Eintrag in der Zugriffskontrollliste das Schreiben von öffentlichen Dokumenten zulässt. Nach einer Änderung dieser Eigenschaft muss die Änderung mit dem Aufruf der Methode `save` gespeichert werden.

Server

Datentyp

boolean

Deklaration Getter

boolean isServer() throws NotesException

Deklaration Setter

void setServer(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft kann ermittelt werden, ob der Eintrag in der Zugriffskontrollliste für einen einzelnen Server steht.

Level**Datentyp**

int

Deklaration Getter

int getLevel() throws NotesException

Deklaration Setter

void setLevel(int level) throws NotesException

Wertebereich

ACL.LEVEL_NOACCESS
ACL.LEVEL_DEPOSITOR
ACL.LEVEL_READER
ACL.LEVEL_AUTHOR
ACL.LEVEL_EDITOR
ACL.LEVEL_DESIGNER
ACL.LEVEL_MANAGER

Beschreibung

In dieser Eigenschaft findet man den Zugriffslevel für den Eintrag in der Zugriffskontrollliste. Nach einer Änderung dieser Eigenschaft muss die Methode save zum Speichern aufgerufen werden.

Name**Datentyp**

String oder Name

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

`void setName(String name)` throws `NotesException`

`void setName(Name n)` throws `NotesException`

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name des Anwenders, des Servers oder der Gruppe für den Eintrag. Nach der Änderung muss die Methode `save` der Klasse *ACL* zum Speichern aufgerufen werden.

NameObject

Datentyp

Name

Deklaration Getter

`Name getNameObject()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Namen des Eintrags der Zugriffskontrollliste.

Parent

Datentyp

ACL

Deklaration Getter

`ACL getParent()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Referenz in Form einer Instanz der Klasse *ACL* auf die zum Eintrag gehörende Zugriffskontrollliste.

Roles**Datentyp**

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getRoles() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Methode liefert als Rückgabewert alle Zugriffsrollen der Zugriffskontrollliste in einem Vektor von Strings.

UserType**Datentyp**

`int`

Deklaration Getter

`int getUserType() throws NotesException`

Deklaration Setter

`void setUserType(int tp) throws NotesException`

Wertebereich

`ACLEntry.TYPE_MIXED_GROUP`
`ACLEntry.TYPE_PERSON`
`ACLEntry.TYPE_PERSON_GROUP`
`ACLEntry.TYPE_SERVER`
`ACLEntry.TYPE_SERVER_GROUP`
`ACLEntry.TYPE_UNSPECIFIED`

Beschreibung

Diese Methode liefert den Typ des Eintrags als numerischen Wert zurück.

9.2.3 Methoden

disableRole

Deklaration

`void disableRole(String role) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
role	nein	String	Name der Rolle, die abgewählt werden soll.

Beschreibung

Mit dieser Methode kann dem Eintrag die angegebene Rolle zugeordnet werden. Diese Änderung muss mit der Methode `save` der Klasse *ACL* gespeichert werden.

enableRole

Deklaration

`void enableRole(String role) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
role	nein	String	Name der Rolle, die zugeordnet werden soll

Beschreibung

Mit dieser Methode kann für dem Eintrag die angegebene Rolle abgewählt werden. Diese Änderung muss mit der Methode `save` der Klasse *ACL* gespeichert werden.

isRoleEnabled

Deklaration

`boolean isRoleEnabled(String role) throws NotesException`

Rückgabewert

boolean

Liefert true, wenn die Entsprechende Rolle dem Eintrag zugeordnet ist, ansonsten false.

Parameter

Name	Optional	Typ	Beschreibung
role	nein	String	Name der Rolle, die überprüft werden soll

Beschreibung

Mit dieser Methode kann überprüft werden, ob die angegebene Rolle für den Eintrag ausgewählt ist.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode kann der Eintrag aus der Zugriffskontrollliste entfernt werden. Danach muss die Methode save der Klasse ACL aufgerufen werden, damit diese Änderung gespeichert wird.

9.3 Klasse Agent

9.3.1 Konstanten

Name	Wert
TARGET_ALL_DOCS	1
TARGET_ALL_DOCS_IN_VIEW	5
TARGET_NEW_DOCS	2
TARGET_NEW_OR_MODIFIED_DOCS	3
TARGET_NONE	0
TARGET_RUN_ONCE	8
TARGET_SELECTED_DOCS	4

Name	Wert
TARGET_UNREAD_DOCS_IN_VIEW	6
TRIGGER_AFTER_MAIL_DELIVERY	2
TRIGGER_BEFORE_MAIL_DELIVERY	6
TRIGGER_DOC_PASTED	3
TRIGGER_DOC_UPDATE	5
TRIGGER_MANUAL	4
TRIGGER_NONE	0
TRIGGER_SCHEDULED	1

9.3.2 Eigenschaften

Comment

Datentyp

String

Deklaration Getter

String getComment() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft steht ein Kommentar zu dem jeweiligen Agent.

CommonOwner

Datentyp

String

Deklaration Getter

String getCommonOwner() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name des Anwenders, der den Agent erstellt oder zuletzt gespeichert hat.

Enabled

Datentyp

boolean

Deklaration Getter

boolean isEnabled() throws NotesException

Deklaration Setter

void setEnabled(boolean value) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft wird die Aktivierung des Agents gesteuert.

NotesAgent

Datentyp

boolean

Deklaration Getter

boolean isNotesAgent() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Durch das Auslesen dieser Eigenschaft kann festgestellt werden, ob der Agent über einen Notes Client ausgeführt werden kann.

Public

Datentyp

boolean

Deklaration Getter

boolean isPublic() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* besitzt, ist der Agent für alle Anwender der Datenbank verfügbar. Ansonsten handelt es sich um einen privaten Agent, der nur von dessen Ersteller genutzt werden kann.

WebAgent

Datentyp

boolean

Deklaration Getter

boolean isWebAgent() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob es sich bei dem Agent um einen Agent für die Webumgebung handelt.

LastRun

Datentyp

DateTime

Deklaration Getter

DateTime getLastRun() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet das Datum der letzten Ausführung dieses Agents.

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Name des Agents ist in dieser Eigenschaft zu finden.

Owner

Datentyp

Name

Deklaration Getter

String getOwner() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Name des Besitzers des Agents kann aus dieser Eigenschaft ausgelesen werden.

ParameterDocID

Datentyp

String

Deklaration Getter

public String getParameterDocID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die NoteID des Parameterdokuments, welches bei den Methoden `Run` oder `RunOnServer` übergeben werden kann.

Parent

Datentyp

Database

Deklaration Getter

Database getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Referenz auf die Datenbank, in der sich der Agent befindet.

Query

Datentyp

String

Deklaration Getter

String getQuery() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Formel mit dem Filterkriterium für die Auswahl der Dokumente, auf denen dieser Agent angewendet wird.

ServerName

Datentyp

String

Deklaration Getter

String getServerName() throws NotesException

Deklaration Setter

void setServerName(String serverName) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Bei einem Agent, der periodisch ausgeführt wird, liefert diese Eigenschaft den Namen des Servers, auf dem der Agent läuft oder einen * als Zeichen dafür, dass der Agent auf jedem Server läuft. Bei einem Agent, der in einer Workstation läuft, liefert diese Eigenschaft den Wert *null*. Sollte es sich bei dem Agent um keinen periodischen Agent handeln, so ist der Wert dieser Eigenschaft ein Leerstring. Um Änderungen dieser Eigenschaft zu speichern, muss die Methode *save* aufgerufen werden.

Target

Datentyp

int

Deklaration Getter

int getTarget() throws NotesException

Deklaration Setter

kein

Wertebereich

Agent.TARGET_ALL_DOCS
Agent.TARGET_ALL_DOCS_IN_VIEW
Agent.TARGET_NEW_DOCS
Agent.TARGET_NEW_OR_MODIFIED_DOCS
Agent.TARGET_NONE
Agent.TARGET_SELECTED_DOCS
Agent.TARGET_UNREAD_DOCS_IN_VIEW
Agent.TARGET_RUN_ONCE

Beschreibung

Diese Eigenschaft beinhaltet einen numerischen Wert, durch den bestimmt werden kann, auf welche Dokumente dieser Agent angewendet wird.

Trigger**Datentyp**

int

Deklaration Getter

int getTrigger() throws NotesException

Deklaration Setter

kein

Wertebereich

Agent.TRIGGER_AFTER_MAIL_DELIVERY
Agent.TRIGGER_BEFORE_MAIL_DELIVERY
Agent.TRIGGER_DOC_PASTED
Agent.TRIGGER_DOC_UPDATE
Agent.TRIGGER_MANUAL
Agent.TRIGGER_NONE
Agent.TRIGGER_SCHEDULED

Beschreibung

Der numerische Wert, der sich in dieser Eigenschaft befindet, gibt Auskunft darüber, ob der Agent durch ein bestimmtes Ereignis automatisch ausgelöst oder über einen manuellen Aufruf gestartet wird.

9.3.3 Methoden

remove

Deklaration

`void remove()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Durch Aufruf dieser Methode wird der Agent entfernt.

run

Deklaration

`void run()` throws `NotesException`

`void run(String noteID)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
noteID	ja	String	Die NoteID des Parameterdokuments

Beschreibung

Mit dieser Methode kann manuell die Ausführung des Agents veranlasst werden. Dabei kann die NoteID eines Parameterdokuments übergeben werden. Mithilfe dieses Parameterdokuments ist es möglich, verschiedene Werte an das Programm des Agents zu übergeben. Der Agent läuft auf dem Computer, auf dem diese Methode aufgerufen wird.

runOnServer

Deklaration

`int runOnServer()` throws `NotesException`

`int runOnServer(String noteID)` throws `NotesException`

Rückgabewert

int

Der Fehlerstatus wird als Rückgabewert zurückgegeben. Dabei kennzeichnet der Wert 0 das erfolgreiche Ausführen.

Parameter

Name	Optional	Typ	Beschreibung
noteID	ja	String	Die NoteID des Parameterdokuments.

Beschreibung

Mit dieser Methode wird die Ausführung des Agents auf dem Server, auf dem sich die Datenbank befindet, ausgelöst. Dabei kann die NoteID eines Parameterdokuments angegeben werden. Das Parameterdokument kann dazu genutzt werden, verschiedene Werte an das Programm des Agents zu übergeben.

save

Deklaration

void save() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode speichert Änderungen der Eigenschaften des Agents permanent in der Notes Datenbank. Nach Änderung einer Eigenschaft muss diese Methode aufgerufen werden, um die aktuellen Eigenschaften zu speichern.

9.4 Klasse AgentContext

9.4.1 Eigenschaften

CurentAgent

Datentyp

Agent

Deklaration Getter

Agent getCurrentAgent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich eine Referenz auf den aktuell laufenden Agent.

CurrentDatabase

Datentyp

Database

Deklaration Getter

Database getCurrentDatabase() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Referenz auf die Datenbank, in der sich der Agent befindet, der gerade ausgeführt wird.

DocumentContext

Datentyp

Document

Deklaration Getter

Document getDocumentContext() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Referenz auf das aktuelle Dokument, welches beim Start des Agents ausgewählt wurde. Dabei hängt der Inhalt der Eigenschaft von der Aufrufsituation ab.

Situation	Referenziertes Dokument
Agent aus einer Ansicht in einem Notes Client gestartet	Das aktuell markierte Dokument
Agent per URL-Kommando OpenAgent aus einem Browser gestartet	Ein neues Dokument mit den CGI-Variablen als Items
Agent mit dem @Command (RunAgent oder ToolsRunMacro)	Aktuelles Dokument
WebQueryOpen	Das Dokument, bevor es in HTML umgewandelt wird
WebQuerySave	Das Dokument, bevor es umgewandelt und gespeichert wird

EffectiveUserName

Datentyp

Name

Deklaration Getter

String getEffectiveUserName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet bei einem Agent, der auf einem Arbeitsplatzrechner ausgeführt wird, den Namen des aktuellen Anwenders. Bei einem Agent, der auf einem Server ausgeführt wird, enthält sie den Namen des Besitzers des Agents, also den Namen des Anwenders, der den Agent erstellt bzw. als Letzter geändert hat.

LastExitStatus

Datentyp

int

Deklaration Getter

int getLastExitStatus() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Statuscode der letzten Ausführung des Agents.

LastRun

Datentyp

DateTime

Deklaration Getter

DateTime getLastRun() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann ausgelesen werden, wann der aktuelle Agent das letzte Mal ausgeführt wurde. Sollte der Agent noch nie ausgeführt worden sein, so befindet sich in dieser Eigenschaft der Wert *null*.

SavedData

Datentyp

Document

Deklaration Getter

Document getSavedData() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft kann genutzt werden, um eine Referenz auf ein Dokument zu erhalten, das vom Agent genutzt werden kann, um Daten zwischen seinen Ausführungen zu speichern. Dieses Dokument wird beim Erstellen des Agents neu angelegt. Bei jeder Änderung des Agents wird dieses Dokument gelöscht und wieder neu angelegt.

UnprocessedDocuments

Datentyp

DocumentCollection

Deklaration Getter

public DocumentCollection getUnprocessedDocuments() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befinden sich alle Dokumente, die vom aktuellen Agent noch nicht behandelt worden sind.

Agenttyp	Die Eigenschaft beinhaltet alle Dokumente, die den folgenden Kriterien entsprechen.
Alle Dokumente in der Datenbank	Alle Dokumente, die dem Suchkriterium in der Agenteigenschaft entsprechen.
Alle neuen oder geänderten Dokumente	Alle Dokumente, die dem Suchkriterium in der Agenteigenschaft entsprechen; seit dem letzten Agentlauf geändert oder angelegt wurden und beim letzten Agentlauf nicht bearbeitet wurden.
Alle ungelesenen Dokumente	Alle Dokumente, die dem Suchkriterium in der Agenteigenschaft entsprechen; noch nicht gelesen wurden und noch nicht durch den Agent bearbeitet wurden.
Alle Dokumente in der Ansicht	Alle Dokumente der Ansicht, die dem Suchkriterium in der Agenteigenschaft entsprechen.
Alle ausgewählten Dokumente	Alle ausgewählten Dokumente, die dem Suchkriterium in der Agenteigenschaft entsprechen.
Keine Dokumente	Nur das aktuell ausgewählte Dokument

9.4.2 Methoden

unprocessedFTSearch

Deklaration

DocumentCollection unprocessedFTSearch(String query, int maxdocs) throws NotesException

DocumentCollection unprocessedFTSearch(String query, int maxdocs, int sortopt, int otheropt) throws NotesException

Rückgabewert

DocumentCollection

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Volltextkriterium
maxdocs	nein	int	Anzahl der Dokumente, die maximal zurückgeliefert werden sollen. Der Wert 0 steht hierbei für alle entsprechenden Dokumente (maximal 2.147.483.647).
sortopt	ja	int	Das Sortierkriterium der zurückgelieferten Dokumentensammlung. Der Defaultwert ist FT_SCORES. FT_SCORES Sortiert nach Relevanz entsprechend dem Suchkriterium. FT_DATE_DES Abwärts sortiert nach Erstellungsdatum FT_DATE_ASC Aufwärts sortiert nach Erstellungsdatum
otheropt	ja	int	Zusätzliche Option für die Suche. Kombiniert werden können die Konstanten mit dem logischen Oder. FT_DATABASE Bei der Suche werden die Notes Datenbanken durchsucht. FT_FILESYSTEM Mit dieser Option werden bei der Suche alle Dateien außer Notes Datenbanken berücksichtigt. FT_FUZZY Es wird ein Fuzzy-Algorithmus für die Suche verwendet. FT_STEMS Es wird der jeweilige Wortstamm der Suchwörter für die Suche verwendet.

Beschreibung

Diese Methode wird verwendet, um eine Volltextsuche durchzuführen. Dabei werden als Parameter die Suchkriterien und Optionen bestimmt. Es werden nach dem Aufruf alle Dokumente in einer Dokumentensammlung zurückgeliefert, welche

noch nicht nach ihrer Erstellung oder der letzten Änderung von dem aktuellen Agent bearbeitet worden sind.

unprocessedSearch

Deklaration

`DocumentCollection unprocessedSearch(String formula, DateTime limit, int maxdocs) throws NotesException`

Rückgabewert

`DocumentCollection`

Parameter

Name	Optional	Typ	Beschreibung
formula	nein	String	Das Suchkriterium in Form einer Notes Formel
limit	nein	DateTime	Es werden nur Dokumente zurückgeliefert, die nach dem angegebenen Datum erstellt oder geändert worden sind.
maxdocs	nein	int	Anzahl der Dokumente, die maximal zurückgeliefert werden sollen. Der Wert 0 steht hierbei für alle entsprechenden Dokumente (maximal 2.147.483.647).

Beschreibung

Diese Methode liefert nach dem Aufruf alle Dokumente, die der angegebenen Formel entsprechen und nach dem angegebenen Datum erstellt oder geändert wurden. Außerdem dürfen die Dokumente noch nicht von dem aktuellen Agent seit ihrer Erstellung oder letzten Änderung bearbeitet worden sein. Die Anzahl der zurückgelieferten Dokumente wird durch den Parameter `limit` beschränkt.

updateProcessedDoc

Deklaration

`void updateProcessedDoc(Document doc) throws NotesException`

Rückgabewert

keinen

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Eine Referenz auf das Dokument, das als bearbeitet markiert werden soll.

Beschreibung

Diese Methode markiert das angegebene Dokument als von dem Agent bearbeitet.

9.5 Klasse Database

9.5.1 Konstanten

Name	Wert
FT_DATABASE	8192
FT_DATE_ASC	64
FT_DATE_DES	32
FT_FILESYSTEM	4096
FT_FUZZY	16384
FT_SCORES	8
FT_STEMS	512
FT_THESAURUS	1024

9.5.2 Eigenschaften

ACL

Datentyp

ACL

Deklaration Getter

ACL getACL() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich eine Referenz auf die Zugriffskontrollliste der Datenbank. Die Datenbank muss zuerst geöffnet werden, um diese Eigenschaft auslesen zu können.

Agents

Datentyp

java.util.Vector(Agent)

Deklaration Getter

java.util.Vector getAgents() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkungen

Beschreibung

Diese Eigenschaft liefert einen Vektor, in dem sich Referenzen auf alle in der Datenbank vorhandenen Agents befinden. Wenn diese Eigenschaft auf einem Server ausgelesen wird, so befinden sich in dem zurückgelieferten Vektor nur die gemeinsamen Agents. Bei dem Auslesen dieser Eigenschaft von einer Workstation aus oder über einen IOP-Zugriff findet man im Vektor die gemeinsamen Agents sowie die Agents des aktuellen Anwenders. Nur nach vorherigem Öffnen ist es möglich, diese Eigenschaft auszulesen.

AllDocuments

Datentyp

DocumentCollection

Deklaration Getter

DocumentCollection getAllDocuments() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Durch diese Eigenschaft erhält man eine Dokumentensammlung mit allen Dokumenten der Datenbank. Der Zugriff auf diese Eigenschaft ist nur möglich, wenn die Datenbank zuvor geöffnet wurde.

Categories

Datentyp

String

Deklaration Getter

String getCategories() throws NotesException

Deklaration Setter

void setCategories(String categories) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Mit dieser Eigenschaft kann bestimmt werden, unter welcher Kategorie die Datenbank in der Datenbankbibliothek auftaucht. Die Datenbank muss für den Zugriff auf diese Eigenschaft zuvor geöffnet worden sein, es sei denn, die Referenz auf die Datenbank ist mit den Methoden `getFirstDatabase` oder `getNextDatabase` aus der Klasse *DbDirectory* ermittelt worden.

Created

Datentyp

DateTime

Deklaration Getter

DateTime getCreated() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich das Erstelldatum der Datenbank. Für einen Zugriff auf diese Eigenschaft ist die Datenbank zuvor zu öffnen.

CurrentAccessLevel

Datentyp

int

Deklaration Getter

int getCurrentAccessLevel throws NotesException

Deklaration Setter

kein

Wertebereich

ACL.LEVEL_AUTHOR
ACL.LEVEL_DEPOSITOR
ACL.LEVEL_DESIGNER
ACL.LEVEL_EDITOR
ACL.LEVEL_MANAGER
ACL.LEVEL_NOACCESS
ACL.LEVEL_READER

Beschreibung

In dieser Eigenschaft befindet sich der Zugriffslevel des aktuellen Anwenders oder – wenn das Programm auf einem Server ausgeführt wird – der Zugriffslevel des Autors des jeweiligen Programms.

DesignTemplateName**Datentyp**

String

Deklaration Getter

String getDesignTemplateName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Hier findet man den Namen der Datenbankvorlage, aus der die Gestaltungselemente für die Datenbank übernommen werden. Der Zugriff auf diese Eigenschaft ist nur möglich, wenn die Datenbank zuvor geöffnet worden ist.

FileName**Datentyp**

String

Deklaration Getter

String getFileName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nach dem Öffnen einer Datenbank befindet sich in dieser Eigenschaft der Dateiname der Datenbank.

FilePath

Datentyp

String

Deklaration Getter

String getFilePath() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nach dem Öffnen einer Datenbank kann aus dieser Eigenschaft der Dateipfad der Datenbank ermittelt werden.

FolderReferencesEnabled

Datentyp

boolean

Deklaration Getter

boolean getFolderReferencesEnabled() throws NotesException

Deklaration Setter

void setFolderReferencesEnabled(boolean bEnable) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob die Ordnerreferenztechnik in der Datenbank unterstützt wird.

Forms**Datentyp**

java.util.Vector(Form)

Deklaration Getter

java.util.Vector getForms() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Vektor mit Referenzen auf alle in der Datenbank befindlichen Masken. Ein Zugriff auf diese Eigenschaft kann nur nach vorherigem Öffnen der Datenbank erfolgen.

IsDelayUpdates**Datentyp**

boolean

Deklaration Getter

boolean isDelayUpdates() throws NotesException

Deklaration Setter

void setDelayUpdates(boolean delay) throws NotesException

Wertebereich

true, false

Beschreibung

Der Wert dieser Eigenschaft gibt Auskunft darüber, ob Änderungen in der Datenbank verzögert gespeichert werden. Das bedeutet, dass Änderungen, die gemacht

werden, nicht sofort auf die Festplatte geschrieben, sondern im Arbeitsspeicher zwischengespeichert werden. Die Einstellung dieser Eigenschaft wird nach dem Schließen der Datenbank nicht gespeichert und muss nach jedem Öffnen neu gesetzt werden.

IsFTIndexed

Datentyp

boolean

Deklaration Getter

boolean isFTIndexed() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Nach dem Öffnen der Datenbank lässt sich anhand des Inhalts dieser Eigenschaft erkennen, ob die Datenbank volltextindiziert ist.

IsMultiDbSearch

Datentyp

boolean

Deklaration Getter

boolean isMultiDbSearch() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt nach dem Öffnen einer Datenbank Auskunft, ob die Datenbank bei einer datenbankübergreifenden Suche berücksichtigt werden kann.

IsOpen

Datentyp

boolean

Deklaration Getter

boolean isOpen() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob die Datenbank für weitere Zugriffe bereits geöffnet wurde, kann durch diese Eigenschaft bestimmt werden. Sollte die Datenbank z. B. mit der Methode open bereits geöffnet worden sein, liefert die Eigenschaft den Wert *true*, ansonsten den Wert *false*.

IsPrivateAddressBook

Datentyp

boolean

Deklaration Getter

boolean isPrivateAddressBook() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Sollte es sich bei der Datenbank um ein privates Adressbuch handeln, liefert die Eigenschaft den Wert *true*, ansonsten den Wert *false*.

IsPublicAddressBook

Datentyp

boolean

Deklaration Getter

boolean isPublicAddressBook() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Handelt es sich bei der Datenbank um ein öffentliches Adressbuch, liefert die Eigenschaft den Wert *true*, ansonsten den Wert *false*.

LastFTIndexed

Datentyp

DateTime

Deklaration Getter

DateTime getLastFTIndexed() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich das Datum der letzten Aktualisierung des Volltext-index. Vor dem Zugriff auf diese Eigenschaft muss die Datenbank geöffnet werden.

LastModified

Datentyp

DateTime

Deklaration Getter

DateTime getLastModified() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Datum der letzten Änderung in der Datenbank befindet sich in dieser Eigenschaft. Vor dem Zugriff auf diese Eigenschaft muss die Datenbank geöffnet werden.

Managers

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getManagers()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nach dem Öffnen einer Datenbank findet man in dieser Eigenschaft eine Liste in Form eines Vektors mit den Namen der Manager für diese Datenbank.

MaxSize

Datentyp

`long`

Deklaration Getter

`long getMaxSize()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

ein Wert größer 0

Beschreibung

Nach dem Öffnen einer Datenbank kann aus dieser Eigenschaft ihre maximale Größe in Kbyte ausgelesen werden.

Parent

Datentyp

Session

Deklaration Getter

Session getParent()

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich eine Referenz auf die Session, in der die Datenbank geöffnet wurde.

PercentUsed

Datentyp

double

Deklaration Getter

double getPercentUsed() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich nach dem Öffnen der Datenbank eine Prozentangabe, welche bestimmt, wie viel des für die Datenbank reservierten Speicherbereichs bereits mit physikalischen Daten ausgenutzt wird.

ReplicaID

Datentyp

String

Deklaration Getter

String getReplicaID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die ReplikationsID der Datenbank als Zeichenkette befindet sich in dieser Eigenschaft.

ReplicationInfo

Datentyp

Replication

Deklaration Getter

Replication getReplicationInfo() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nachdem eine Datenbank geöffnet wurde, befindet sich in dieser Eigenschaft eine Referenz auf die Klasse *Replicate*. Diese gibt Auskunft über die Replikation der Datenbank.

Server

Datentyp

String

Deklaration Getter

String getServer() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Um den Namen des Servers herauszufinden, auf dem sich die Datenbank befindet, kann diese Eigenschaft genutzt werden. Sollte die Datenbank sich auf einer Workstation befinden, liefert diese Eigenschaft einen Leerstring.

Size

Datentyp

double

Deklaration Getter

double getSize() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Größe der Datenbank in Byte.

SizeQuota

Datentyp

int

Deklaration Getter

int getSizeQuota() throws NotesException

Deklaration Setter

void setSizeQuota(int quota) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft kann ein Administrator die maximale Größe einer Datenbank bestimmen oder ein Anwender diese auslesen. Sollte für die Datenbank keine Größenbeschränkung bestehen, liefert diese Eigenschaft den Wert 0.

TemplateName

Datentyp

String

Deklaration Getter

String getTemplateName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn es sich bei der Datenbank um eine Datenbankschablone handelt, kann über diese Eigenschaft der Schablonenname ermittelt werden.

Title

Datentyp

String

Deklaration Getter

String getTitle() throws NotesException

Deklaration Setter

void setTitle(String title) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft kann der Titel für die Datenbank gesetzt oder ausgelesen werden.

Views

Datentyp

java.util.Vector(View)

Deklaration Getter

java.util.Vector getViews() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nach dem Öffnen einer Datenbank befindet sich in dieser Eigenschaft eine Liste in Form eines Vektors mit allen Ansichten und Ordnern der Datenbank. Sollte der Aufruf auf einem Server erfolgen, so werden nur die gemeinsamen Ansichten oder Ordner geliefert. Bei Aufruf von einer Arbeitsstation werden ebenso die persönlichen Ansichten oder Ordner geliefert.

9.5.3 Methoden

compact

Deklaration

int compact() throws NotesException

Rückgabewert

int

Zurückgeliefert wird die Differenz des Speicherbedarfs der Datenbank vor und nach der Komprimierung.

Parameter

keine

Beschreibung

Diese Datenbank komprimiert eine lokale Datenbank. Sollte die Datenbank nicht lokal gespeichert sein, wird eine Exception geworfen.

createCopy

Deklaration

Database createCopy(String server, String dbfile) throws NotesException

Database createCopy(String server, String dbfile, int maxsize) throws NotesException

Rückgabewert

Database

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Server, auf dem die Kopie angelegt werden soll. Ein Leerstring steht dabei für den aktuellen Computer.
dbfile	nein	String	Dateiname der neuen Kopie
maxsize	ja	int	Maximale Größe der neu kopierten Datenbank in Gigabyte. Dieser Parameter ist nur relevant für Datenbanken für die Notes Version 4. Ein Wert größer als 4 führt zu einem Laufzeitfehler.

Beschreibung

Diese Methode kann genutzt werden, um eine Kopie einer bestehenden Datenbank anzulegen. Sollte die Methode auf einem Server verwendet werden, ist es nicht möglich, auf andere Server zuzugreifen. Sollte bereits unter dem angegebenen Dateinamen eine Datenbank existieren, wird ebenfalls eine Ausnahme geworfen.

createDocument

Deklaration

Document createDocument() throws NotesException

Rückgabewert

Document

Parameter

keine

Beschreibung

Diese Methode legt in der Datenbank ein neues Dokument an und gibt eine Referenz darauf zurück. Um das neu angelegte Dokument permanent in der Datenbank zu speichern, muss die Methode save des Dokuments aufgerufen werden.

createFromTemplate

Deklaration

Database createFromTemplate(String server, String dbfile, boolean inherit)
throws NotesException

Database createFromTemplate(String server, String dbfile, boolean inherit,
int maxsize) throws NotesException

Rückgabewert

Database

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des Servers, auf dem die neue Datenbank angelegt werden soll. Ein Leerstring steht dabei für den aktuellen Computer.
dbfile	nein	String	Dateiname der neuen Datenbank
inherit	nein	boolean	Dieser Wert spezifiziert, ob spätere Gestaltungsänderungen in der neuen Kopie übernommen werden sollen.
maxsize	ja	int	Maximale Größe der neu kopierten Datenbank in Gigabyte. Dieser Parameter ist nur relevant für Datenbanken für die Notes Version 4. Ein Wert größer als 4 führt zu einem Laufzeitfehler.

Beschreibung

Diese Methode erzeugt neue Datenbanken und übernimmt die Gestaltungselemente der aktuellen Datenbank. Durch den Parameter `inherit` kann bestimmt werden, ob spätere Gestaltungsänderungen in der Quelldatenbank in die Zieldatenbank übernommen werden. Sollte unter dem angegebenen Dateinamen bereits eine Datenbank existieren, wird eine Exception geworfen. Bei Verwendung auf einem Server ist zu beachten, dass dieser nicht auf andere Server zugreifen darf.

createOutline

Deklaration

Outline createOutline(String name) throws NotesException

Outline createOutline(String name, boolean defaultOutline) throws NotesException

Rückgabewert

Outline

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des neuen Gliederungselements
defaultOutline	ja	boolean	Dieser Parameter bestimmt, ob eine leere Gliederung angelegt werden soll oder eine Gliederung mit einer Standardstruktur. Der Defaultwert ist <i>false</i> .

Beschreibung

Durch diese Methode wird ein neues Gliederungselement in der Datenbank angelegt. Eine permanente Speicherung der neuen Gliederung erfolgt erst nach dem Aufruf der Methode `save` für die zurückgelieferte Referenz.

createReplica

Deklaration

Database createReplica(String server, String dbfile) throws NotesException

Rückgabewert

Database

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des Servers, auf dem die Replik erstellt werden soll. Ein Leerstring steht dabei für den aktuellen Computer.
dbfile	nein	String	Dateiname der neuen Replik

Beschreibung

Mit dieser Methode wird eine Replik einer Datenbank angelegt. Ein Server kann dabei keine Repliken auf anderen Server anlegen. Ebenso wird eine Exception geworfen, sollte unter dem angegebenen Dateinamen bereits eine Datenbank existieren.

enableFolder

Deklaration

void enableFolder(String folder) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
folder	nein	String	Name des neuen Ordners

Beschreibung

Diese Methode legt einen neuen Ordner in der Datenbank an. Sollte bereits ein Ordner mit dem entsprechenden Namen existieren, führt diese Methode keine Aktion aus.

FTDomainSearch

Deklaration

Document FTDomainSearch(String query, int max, int sortopt, int otheropt, int start, int count, String entryForm) throws NotesException

Rückgabewert

Document

Dokument mit den gefundenen Einträgen

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Suchformel
max	nein	int	Maximale Anzahl an Dokumenten, die zurückgeliefert werden. Der Wert 0 steht für keine Einschränkung. Maximal werden 2.147.483.647 Dokumente zurückgeliefert.
sortopt	ja	int	Das Sortierkriterium der zurückgelieferten Dokumentensammlung. Der Defaultwert ist FT_SCORES. FT_SCORES Sortiert nach Relevanz entsprechend dem Suchkriterium FT_DATE_DES Abwärts sortiert nach Erstellungsdatum FT_DATE_ASC Aufwärts sortiert nach Erstellungsdatum
otheropt	ja	int	Zusätzliche Option für die Suche. Kombiniert werden können die Konstanten mit dem logischen Oder. FT_DATABASE Bei der Suche werden die Notes Datenbanken durchsucht. FT_FILESYSTEM Mit dieser Option werden bei der Suche alle Dateien außer Notes Datenbanken berücksichtigt. FT_FUZZY Es wird ein Fuzzy-Algorithmus für die Suche verwendet. FT_STEMS Es wird der jeweilige Wortstamm der Suchwörter für die Suche verwendet.
start	nein	int	Startseite
count	nein	int	Anzahl der Seiten
entryForm	nein	String	

Beschreibung

Diese Methode kann in einer Domänenkatalogdatenbank aufgerufen werden, um eine Volltextsuche über die Datenbanken durchzuführen. Das Ergebnis wird in Form eines Dokuments zurückgeliefert.

FTSearch

Deklaration

DocumentCollection FTSearch(String query) throws NotesException

DocumentCollection FTSearch(String query, int max) throws NotesException

DocumentCollection FTSearch(String query, int max, int sortopt, int otheropt)
throws NotesException

Rückgabewert

DocumentCollection

eine Dokumentensammlung mit den gefundenen Dokumenten

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Suchformel
max	nein	int	Maximale Anzahl an Dokumenten, die zurückgeliefert werden. Der Wert 0 steht für keine Einschränkung. Maximal werden 2.147.483.647 Dokumente zurückgeliefert.
sortopt	ja	int	Das Sortierkriterium der zurückgelieferten Dokumentensammlung. Der Defaultwert ist FT_SCORES. FT_SCORES Sortiert nach Relevanz entsprechend dem Suchkriterium. FT_DATE_DES Abwärts sortiert nach Erstellungsdatum FT_DATE_ASC Aufwärts sortiert nach Erstellungsdatum
otheropt	ja	int	Zusätzliche Option für die Suche. Kombiniert werden können die Konstanten mit dem logischen Oder. FT_FUZZY Es wird ein Fuzzy-Algorithmus für die Suche verwendet. FT_STEMS Es wird der jeweilige Wortstamm der Suchwörter für die Suche verwendet.

Beschreibung

Durch den Aufruf dieser Methode wird die Datenbank mit der angegebenen Suchformel durchsucht und das Ergebnis in Form einer Dokumentensammlung zurückgegeben.

FTSearchRange

Deklaration

DocumentCollection FTSearchRange(String query, int max, int sortopt, int otheropt, int start) throws NotesException

Rückgabewert

DocumentCollection

eine Dokumentensammlung mit den gefundenen Dokumenten

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Suchformel
max	nein	int	Maximale Anzahl an Dokumenten, die zurückgeliefert werden. Der Wert 0 steht für keine Einschränkung. Maximal werden 2.147.483.647 Dokumente zurückgeliefert.
sortopt	nein	int	Das Sortierkriterium der zurückgelieferten Dokumentensammlung. Der Defaultwert ist FT_SCORES. FT_SCORES Sortiert nach Relevanz entsprechend dem Suchkriterium. FT_DATE_DES Abwärts sortiert nach Erstellungsdatum FT_DATE_ASC Aufwärts sortiert nach Erstellungsdatum
otheropt	nein	int	Zusätzliche Option für die Suche. Kombiniert werden können die Konstanten mit dem logischen Oder. FT_FUZZY Es wird ein Fuzzy-Algorithmus für die Suche verwendet. FT_STEMS Es wird der jeweilige Wortstamm der Suchwörter für die Suche verwendet.
start	nein	int	Das Startdokument, das zurückgegeben werden soll.

Beschreibung

Durch den Aufruf dieser Methode wird die Datenbank mit der angegebenen Suchformel durchsucht und das Ergebnis in Form einer Dokumentensammlung zurückgegeben.

getAgent

Deklaration

Agent getAgent(String name) throws NotesException

Rückgabewert

Agent

wenn vorhanden, eine Referenz auf den geforderten Agent

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Agents

Beschreibung

Durch den Aufruf dieser Methode erhält man, falls vorhanden, eine Referenz auf den Agent in der Datenbank mit dem angegebenen Namen. Sollte es sich um einen privaten Agent handeln und der aktuelle Anwender nicht der Besitzer sein, liefert der Methodenaufruf *null* zurück.

getDocumentByID

Deklaration

Document getDocumentByID(String noteid) throws NotesException

Rückgabewert

Document

wenn vorhanden, das Dokument mit der angegebenen NodeID in der Datenbank

Parameter

Name	Optional	Typ	Beschreibung
noteid	nein	String	NodeID des gesuchten Dokuments

Beschreibung

Die Methode liefert nach dem Aufruf, falls in der Datenbank vorhanden, das Dokument mit der angegebenen NodeID zurück.

getDocumentByUNID

Deklaration

Document getDocumentByUNID(String unid) throws NotesException

Rückgabewert

Document

das Dokument mit der entsprechenden UniversalID

Parameter

Name	Optional	Typ	Beschreibung
unid	nein	String	Die UniversalID des gesuchten Dokuments

Beschreibung

Diese Methode liefert nach ihrem Aufruf, falls in der Datenbank vorhanden, das Dokument mit der angegebenen UniversalID zurück.

getDocumentByURL

Deklaration

Document getDocumentByURL(String url, boolean reload) throws NotesException
 Document getDocumentByURL(String url, boolean reload, boolean relifmod, boolean urllist, String charset, String webuser, String webpasswd, String proxy-user, String proxypasswd, boolean returnimmediately) throws NotesException

Rückgabewert

Document

die angeforderte Internetseite umgewandelt in ein Notes Dokument

Parameter

Name	Optional	Typ	Beschreibung
url	nein	String	Die URL-Adresse- die gelesen werden soll.
reload	nein	boolean	Falls dieser Parameter false ist, wird das Dokument nur erneut angefordert, wenn es sich nicht in der Webnavigator-Datenbank befindet. Wenn der Parameter den Wert true hat, wird die Seite immer neu vom Internetserver gelesen.
relifmod	ja	boolean	Falls dieser Parameter false ist, wird das Dokument nur erneut angefordert, wenn es sich nicht in aktueller Version in der Webnavigator-Datenbank befindet.
urllist	ja	boolean	Mit diesem Parameter kann gesteuert werden, ob alle Links auf der Webseite gesondert in einem Feld mit dem Namen URLLinks gespeichert werden sollen. (Default: false)
charset	ja	String	Über diesen Parameter kann eine Zeichensatztafel angegeben werden, die für den Aufruf genutzt werden soll. (Default: null)
webuser	ja	String	Falls notwendig, kann hier ein Username für die Anmeldung an einem Internetserver angegeben werden. (Default: null)
webpasswd	ja	String	Falls notwendig, kann hier ein Passwort für die Anmeldung an einem Internetserver angegeben werden. (Default: null)

Name	Optional	Typ	Beschreibung
proxyuser	ja	String	Falls notwendig, kann hier ein Username für die Anmeldung an dem Proxyserver angegeben werden. (Default: null)
proxy-passwd	ja	String	Falls notwendig, kann hier ein Passwort für die Anmeldung an dem Proxyserver angegeben werden. (Default: null)
returnimmediately	ja	boolean	Wenn dieser Parameter den Wert false hat, wartet die Methode mit dem Rücksprung, bis die Seite komplett gelesen wurde. Bei dem Wert true kehrt die Methode sofort zurück. Das Lesen des Dokuments erfolgt dann im Hintergrund.

Beschreibung

Bei Aufruf der Methode wird die angegeben Internetseite gelesen und in Form eines Notes Dokuments zurückgegeben. Wenn für den Parameter `returnimmediately` der Wert *true* zurückgeliefert wird, wird das Dokument nicht zurückgeliefert, sondern im Hintergrund geladen.

getForm

Deklaration

Form `getForm(String name)` throws `NotesException`

Rückgabewert

Form

eine Referenz auf die Maske mit dem angegebenen Namen

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der gesuchte Maske

Beschreibung

Mit dieser Methode erhält man eine Referenz auf die Maske mit dem angegebenen Namen.

getOutline

Deklaration

Outline `getOutline(String name)` throws `NotesException`

Rückgabewert

Outline

Referenz auf die gesuchte Gliederung

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der gesuchten Gliederung

Beschreibung

Durch die Methode wird eine Referenz auf die gesuchte Gliederung mit dem angegebenen Namen zurückgeliefert.

getProfileDocCollection

Deklaration

DocumentCollection getProfileDocCollection(String name) throws NotesException

Rückgabewert

DocumentCollection

Zurückgeliefert wird eine Dokumentensammlung mit den Profildokumenten.

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Maske für die gesuchten Profildokumente

Beschreibung

Die Methode sucht alle Profildokumente in der Datenbank, welche auf der angegebenen Maske basieren, und liefert diese in der Dokumentensammlung zurück.

getProfileDocument

Deklaration

Document getProfileDocument(String profile, String profilekey) throws NotesException

Rückgabewert

Document

eine Referenz auf das Profildokument

Parameter

Name	Optional	Typ	Beschreibung
profil	nein	String	Maskenname für das Profildokument
profilkey	nein	String	Schlüsselwert des Profildokuments

Beschreibung

Nach dem Aufruf dieser Methode wird nach einem entsprechenden Profildokument in der Datenbank gesucht. Sollte dort kein entsprechendes Dokument vorhanden sein, wird eines angelegt. Als Rückgabewert dieser Methode bekommt man eine Referenz auf das Profildokument zurück.

getURLHeaderInfo

Deklaration

String getURLHeaderInfo(String url, String header, String webuser, String webpasswd, String proxyuser, String proxypasswd) throws NotesException

Rückgabewert

String

Wert der gesuchten Headerinformation

Parameter

Name	Optional	Typ	Beschreibung
url	nein	String	URL der gesuchten Internetseite
header	nein	String	Name der gesuchten Information
webpasswd	ja	String	Falls notwendig, kann hier ein Passwort für die Anmeldung an einem Internetserver angegeben werden. (Default null)
proxyuser	ja	String	Falls notwendig, kann hier ein Username für die Anmeldung an dem Proxyserver angegeben werden. (Default null)
proxypasswd	ja	String	Falls notwendig, kann hier ein Passwort für die Anmeldung an dem Proxyserver angegeben werden. (Default null)

Beschreibung

Mit dieser Methode können URL-Headerinformationen einer beliebigen Internetseite ermittelt werden. Der Wert der gesuchten Information wird als String zurückgeliefert.

getView

Deklaration

View getView(String name) throws NotesException

Rückgabewert

View

gesuchte Ansicht

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der gesuchten Ansicht

Beschreibung

Diese Methode liefert die Referenz auf die Ansicht mit dem angegebenen Namen zurück.

grantAccess

Deklaration

void grantAccess(String name, int level) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Person, des Servers oder der Gruppe, für die/den das Recht vergeben werden soll.
level	nein	int	Gewünschter Zugriffslevel: ACL.LEVEL_AUTHOR, ACL.LEVEL_DEPOSITOR, ACL.LEVEL_DESIGNER, ACL.LEVEL_EDITOR, ACL.LEVEL_MANAGER, ACL.LEVEL_NOACCESS, ACL.LEVEL_READER

Beschreibung

Diese Methode ermöglicht es, für die angegebene Person, den Server oder die Gruppe Zugriffsrechte zu erteilen. Eine gleichzeitige Nutzung dieser Methode und des Zugriffs über die Klasse *ACL* birgt das Risiko der Inkonsistenz der Daten.

open

Deklaration

boolean open() throws NotesException

Rückgabewert

boolean

Liefert true, wenn die Datenbank geöffnet werden konnte, ansonsten den Wert false.

Parameter

keine

Beschreibung

Diese Methode öffnet die Datenbank.

queryAccess

Deklaration

int queryAccess(String name) throws NotesException

Rückgabewert

int

Das Zugriffsrecht der angegebenen Person, Servers oder Gruppe. Mögliche Werte sind (ACL.LEVEL_AUTHOR, ACL.LEVEL_DEPOSITOR, ACL.LEVEL_DESIGNER, ACL.LEVEL_EDITOR, ACL.LEVEL_MANAGER, ACL.LEVEL_NOACCESS, ACL.LEVEL_READER).

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Person, des Servers oder der Gruppe, für die/den das Recht abgefragt werden soll.

Beschreibung

Diese Methode fragt die Zugriffsrechte für eine Person, einen Server oder eine Gruppe ab.

queryAccessRoles

Deklaration

java.util.Vector queryAccessRoles(String name) throws NotesException

Rückgabewert

java.util.Vector

ein Vektor, in dem die Rollen für die angegebene Person, den Server bzw. die Gruppe aufgelistet sind

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Person, des Servers oder der Gruppe, für die/den die Rollen abgefragt werden sollen.

Beschreibung

Diese Methode liefert die Rollen für die angegebene Person, den Server oder die Gruppe zurück.

remove

Deklaration

`void remove()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode wird die Datenbank gelöscht.

replicate

Deklaration

`boolean replicate(String server)` throws `NotesException`

Rückgabewert

boolean

Falls die Replikation erfolgreich war, liefert die Methode `true` zurück, ansonsten `false`.

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des Servers, mit dem die Datenbank repliziert werden soll.

Beschreibung

Durch den Aufruf dieser Methode wird die Datenbank mit den Repliken auf den angegebenen Server repliziert.

revokeAccess

Deklaration

`void revokeAccess(String name)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name der Person, des Servers oder der Gruppe deren Eintrag in der Zugriffsrechtliste entfernt werden sollen

Beschreibung

Durch den Aufruf dieser Methode wird der Eintrag der angegebenen Person, des Servers oder der Gruppe aus der Zugriffskontrollliste entfernt.

search

Deklaration

DocumentCollection search(String formula) throws NotesException

DocumentCollection search(String formula, DateTime dt) throws NotesException

DocumentCollection search(String formula, DateTime dt, int max) throws NotesException

Rückgabewert

DocumentCollection

Dokumentensammlung mit den Dokumenten, die den Suchkriterien entsprechen

Parameter

Name	Optional	Typ	Beschreibung
formula	nein	String	Notes Formel mit dem Suchkriterium
dt	ja	DateTime	Es werden nur Dokumente zurückgeliefert, die nach dem angegebenen Datum erstellt oder geändert worden sind.
max	ja	int	Maximale Anzahl an Dokumenten, die zurückgeliefert werden. Der Wert 0 steht für keine Einschränkung. Maximal werden 2.147.483.647 Dokumente zurückgeliefert.

Beschreibung

Diese Methode durchsucht die Datenbank und liefert in einer Dokumentensammlung alle Dokumente, die der angegebenen Formel entsprechen, zurück. Außerdem wird berücksichtigt, dass nur Dokumente zurückgeliefert werden, die nach dem angegebenen Datum erstellt oder geändert worden sind.

updateFTIndex

Deklaration

void updateFTIndex(boolean create) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
create	nein	boolean	Bei dem Wert true wird der Volltextindex neu angelegt, sollte er noch nicht existieren. Wenn der Wert false angegeben wird, wird kein neuer Volltextindex angelegt, sondern nur ein bereits existierender aktualisiert.

Beschreibung

Diese Methode aktualisiert oder legt ggf. den Volltextindex für eine Datenbank an.

9.6 Klasse *DateRange*

9.6.1 Eigenschaften

EndTime

Datentyp

DateTime

Deklaration Getter

DateTime getEndTime()

Deklaration Setter

void setEndTime(*DateTime* end)

Wertebereich

keine Einschränkung

Beschreibung

Das Enddatum des Datumsbereichs.

Parent

Datentyp

Session

Deklaration Getter

Session getParent()

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Eine neue Instanz der Klasse *DateRange* wird immer innerhalb einer gültigen Instanz der Klasse *NotesSession* z. B. durch die Methode `createDateRange` erzeugt. Diese Eigenschaft beinhaltet eine Referenz auf die entsprechende Instanz der Klasse *NotesSession*.

StartDateTime

Datentyp

`DateTime`

Deklaration Getter

`DateTime getStartDateTime()`

Deklaration Setter

`setStartDateTime(DateTime start)`

Wertebereich

keine Einschränkung

Beschreibung

Das Startdatum des Datumsbereichs.

Text

Datentyp

`Object(String)`

Deklaration Getter

`String getText()` throws `NotesException`

Deklaration Setter

`void setText(String text)` throws `NotesException`

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Datumsbereich in Textform.

9.6.2 Methoden

Keine

9.7 Klasse *DateTime*

9.7.1 Eigenschaften

DateOnly

Datentyp

String

Deklaration Getter

String getDateOnly() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Beim Auslesen dieser Eigenschaft erhält man das Datum in Textform.

GMTTime

Datentyp

String

Deklaration Getter

String getGMTTime() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Beim Auslesen dieser Eigenschaft erhält man die Zeit umgerechnet auf GMT in Textform.

IsDST

Datentyp

boolean

Deklaration Getter

boolean isDST() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft, ob der Zeit- und Datumsausdruck die Sommerzeit berücksichtigt.

LocalTime

Datentyp

String

Deklaration Getter

String getLocalTime() throws NotesException

Deklaration Setter

public void setLocalTime(String dt) throws NotesException

public void setLocalTime(java.util.Date dt) throws NotesException

public void setLocalTime(java.util.Calendar dt) throws NotesException

public void setLocalDate(int year, int month, int day) throws NotesException

public void setLocalDate(int year, int month, int day, boolean preserveLocalTime)
throws NotesException

public void setLocalTime(int hour, int minute, int second, int hundredth)
throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft ermöglicht das Auslesen und Setzen des Zeitwerts unter Berücksichtigung der lokalen Zeitrechnung.

Parent

Datentyp

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Notes Session, in der die Instanz erstellt wurde.

TimeOnly

Datentyp

String

Deklaration Getter

String getTimeOnly() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft erlaubt das Auslesen des Zeitanteils des Zeit- und Datumswertes.

TimeZone

Datentyp

Integer

Deklaration Getter

int getTimeZone() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft verrät, wie viele Stunden zu der Zeitrechnung für dieses Datum addiert bzw. von diesem subtrahiert werden müssen, um die GMT-Zeit zu erhalten.

ZoneTime

Datentyp

String

Deklaration Getter

String getTimeZone() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Uhrzeit in Form einer Zeichenkette. Dabei werden die Zeitzonen sowie die Sommerzeiteinstellung berücksichtigt.

9.7.2 Methoden

adjustDay

Deklaration

void adjustDay(int n) throws NotesException

void adjustDay(int n, boolean preserveLocalTime) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Tage, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Mit dieser Methode können Tage zu einem Datumswert addiert oder von diesem subtrahiert werden.

adjustHour

Deklaration

`void adjustHour(int n) throws NotesException`

`void adjustHour(int n, boolean preserveLocalTime) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Stunden, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Mit dieser Methode können Stunden zu einem Datumswert addiert oder von diesem subtrahiert werden.

adjustMinute

Deklaration

`void adjustMinute(int n) throws NotesException`

`void adjustMinute(int n, boolean preserveLocalTime) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Minuten, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Mit dieser Methode können Minuten zu einem Datumswert addiert oder von diesem subtrahiert werden.

adjustMonth

Deklaration

void adjustMonth(int n) throws NotesException

void adjustMonth(int n, boolean preserveLocalTime) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Monate, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

int n

boolean preserveLocalTime (optional)

Beschreibung

Mit dieser Methode können Monate zu einem Datumswert addiert oder von diesem subtrahiert werden.

adjustSecond

Deklaration

void adjustSecond(int n) throws NotesException

void adjustSecond(int n, boolean preserveLocalTime) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Sekunden, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Mit dieser Methode können Sekunden zu einem Datumswert addiert oder von diesem subtrahiert werden.

adjustYear

Deklaration

`void adjustYear(int n) throws NotesException`

`void adjustYear(int n, boolean preserveLocalTime) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Anzahl der Jahre, die addiert oder subtrahiert werden sollen
preserveLocalTime	ja	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Mit dieser Methode können Jahre zu einem Datumswert addiert oder von diesem subtrahiert werden.

convertToZone

Deklaration

`void convertToZone(int zone, boolean isDST) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
zone	nein	int	Zeitzone
isDST	nein	boolean	Berücksichtigung der Sommerzeit

Beschreibung

Diese Methode konvertiert das Datum entsprechend der neuen Angaben von Zeitzone und Sommerzeit.

setAnyDate

Deklaration

`void setAnyDate() throws NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode setzt den Datumswert auf ein konstantes Datum, welches für jedes beliebige Datum steht.

setAnyTime

Deklaration

`void setAnyTime()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode setzt den Datumswert auf ein konstantes Datum, welches für jedes beliebige Datum steht.

setNow

Deklaration

`void setNow()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode setzt das Datum auf das aktuelle Datum und die aktuelle Zeit.

timeDifference

Deklaration

`public int timeDifference(DateTime dt)` throws `NotesException`

`public double timeDifferenceDouble(DateTime dt)` throws `NotesException`

Rückgabewert

int oder double

Parameter

Name	Optional	Typ	Beschreibung
dt	nein	DateTime	Zweiter Zeitwert

Beschreibung

Diese Methode liefert die Differenz zwischen dem aktuellen Datum und dem angegebenen Datumswert in Sekunden.

toJavaDate

Deklaration

java.util.Date toJavaDate() throws NotesException

Rückgabewert

java.util.Date

Parameter

keine

Beschreibung

Diese Methode liefert das Datum in Form einer Referenz auf die Klasse *java.util.Date*.

9.8 Klasse DbDirectory

9.8.1 Konstanten

Name	Wert
DATABASE	1247
REPLICA_CANDIDATE	1245
TEMPLATE	1248
TEMPLATE_CANDIDATE	1246

9.8.2 Eigenschaften

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Name des Servers des Datenbankverzeichnisses.

Parent

Datentyp

Session

Deklaration Getter

Session getParent()

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Notes Session, in der diese Referenz der Klasse *DbDirectory* erzeugt wurde.

9.8.3 Methoden

createDatabase

Deklaration

Database createDatabase(String dbfile) throws NotesException

Database createDatabase(String dbfile, boolean open) throws NotesException

Rückgabewert

Database

eine Referenz auf die neue Datenbank

Parameter

Name	Optional	Typ	Beschreibung
dbfile	nein	String	Dateiname der Datenbank
open	ja	boolean	True, wenn die Datenbank geöffnet werden soll

Beschreibung

Diese Methode erlaubt das Anlegen einer Datenbank.

getFirstDatabase**Deklaration**

Database getFirstDatabase(int type) throws NotesException

Rückgabewert

Database

erste gefundene Datenbank

Parameter

Name	Optional	Typ	Beschreibung
type	nein	int	Der Typ der Elemente, die durchsucht werden sollen. Mögliche Werte sind: DbDirectory.DATABASE, DbDirectory.TEMPLATE, DbDirectory.REPLICA_CANDIDATE, DbDirectory.TEMPLATE_CANDIDATE

Beschreibung

Mit dieser Methode wird die erste Datenbank, die den Suchkriterien entspricht, zurückgeliefert.

getNextDatabase**Deklaration**

Database getNextDatabase() throws NotesException

Rückgabewert

Database

die nächste gefundene Datenbank

Parameter

keine

Beschreibung

Nach dem Aufruf der Methode getFirstDatabase liefert diese Methode die weiteren Datenbanken.

openDatabase

Deklaration

Database openDatabase(String dbfile) throws NotesException

Database openDatabase(String dbfile, boolean failover) throws NotesException

Rückgabewert

Database

Referenz auf die geöffnete Datenbank. Null, falls die Datenbank nicht geöffnet werden konnte.

Parameter

Name	Optional	Typ	Beschreibung
dbfile	nein	String	Dateiname der Datenbank
failover	ja	boolean	Wenn für diesen Parameter der Wert true angegeben wird, kann bei Problemen in einem Cluster die Datenbank auf einem anderen als dem eigenen Server geöffnet werden. Bei Angabe des Wertes false ist dies nicht möglich.

Beschreibung

Mit dieser Methode kann eine Datenbank geöffnet werden. Dabei wird die Datenbank anhand des übergebenen Dateinamens spezifiziert.

openDatabaseByReplicaID

Deklaration

Database openDatabaseByReplicaID(String rid) throws NotesException

Rückgabewert

Database

die geöffnete Datenbank mit der angegebenen Replikations-ID

Parameter

Name	Optional	Typ	Beschreibung
rid	nein	String	Replikations-ID der Datenbank, die geöffnet werden soll

Beschreibung

Diese Methode öffnet die Datenbank mit der angegebenen Replikations-ID.

openDatabaseModified

Deklaration

Database openDatabaseSelfModified(String dbfile, DateTime date) throws
NotesException

Rückgabewert

Database
geöffnete Datenbank

Parameter

Name	Optional	Typ	Beschreibung
dbfile	nein	String	Dateiname der Datenbank
date	nein	DateTime	Es wird die angegebene Datenbank nur geöffnet, wenn in ihr Änderungen nach diesem Datum durchgeführt wurden.

Beschreibung

Diese Methode öffnet die angegebene Datenbank, falls in ihr Änderungen nach dem angegebenen Datum erfolgt sind. Nach dem Aufruf wird eine Referenz auf die geöffnete Datenbank zurückgegeben.

openMailDatabase

Deklaration

Database openMailDatabase() throws NotesException

Rückgabewert

Database
die geöffnete Maildatenbank, falls sie geöffnet werden konnte, oder der Wert null

Parameter

keine

Beschreibung

Diese Methode öffnet die Maildatenbank des aktuellen Anwenders. Wenn diese Methode an einer Arbeitsstation aufgerufen wird, so wird die in der *notes.ini*-Datei angegebene Maildatenbank geöffnet. Sollte die Methode aus einem Agent heraus auf einem Server aufgerufen werden, so wird die Maildatenbank des Besitzers des Agents geöffnet. Allerdings muss die Datenbank sich dazu auf dem aktuellen Server befinden. Wenn diese Methode in einem Programm aufgerufen wird, welches per Remote-Zugriff arbeitet, so wird die Maildatenbank des Anwenders aufgerufen, der die Sitzung initiiert hat.

9.9 Klasse Document

9.9.1 Eigenschaften

Authors

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getAuthors() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Liste der Anwender, die das Dokument bisher gespeichert haben. Die Liste wird als Vektor von Zeichenketten zurückgegeben.

ColumnValues

Datentyp

`java.util.Vector(Object)`

Deklaration Getter

`java.util.Vector getColumnValues() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Vektor der Inhalte der Spalten, welche in der übergeordneten Ansicht angezeigt werden.

Created

Datentyp

DateTime

Deklaration Getter

DateTime getCreated() throws NotesExceptio

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Erstelldatum des Dokuments befindet sich in dieser Eigenschaft.

EmbeddedObjects

Datentyp

java.util.Vector(EmbeddedObject)

Deklaration Getter

java.util.Vector getEmbeddedObjects() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet in einem Vektor alle eingebetteten OLE-Objekte des Dokuments.

EncryptionKeys

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getEncryptionKeys() throws NotesException

Deklaration Setter

`void setEncryptionKeys(java.util.Vector keys) throws NotesException`

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft des Dokuments ist es möglich, die Liste der Schlüssel, die zum Verschlüsseln des Dokuments verwendet werden, zu setzen und zu lesen. Diese Schlüssel werden bei Aufruf der `encrypt`-Methode genutzt, um das Dokument zu verschlüsseln und ebenso beim Lesen des Dokuments zum Entschlüsseln. Wird hier kein Schlüssel angegeben, so wird das Dokument mit dem öffentlichen Schlüssel des aktuellen Anwenders verschlüsselt. Allerdings wird diese Eigenschaft nicht bei der Verschlüsselung beim Mailverkehr berücksichtigt.

FolderReferences**Datentyp**

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getFolderReferences() throws NotesException`

Deklaration Setter

keine Einschränkung

Wertebereich

In dieser Eigenschaft befindet sich eine Liste mit Universal-IDs der Ordner, in denen sich das Dokument befindet. Diese Eigenschaft liefert diese IDs allerdings nur, wenn die Datenbank Ordnerreferenzen zulässt. Dafür müssen die zwei versteckten Ansichten *\$FolderInfo* und *\$FolderRefInfo* in der Datenbank vorhanden sein.

FTSearchScore**Datentyp**

`int`

Deklaration Getter

`int getFTSearchScore() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Nach einer Volltextsuche werden die Dokumente in der Reihenfolge ihrer Relevanz basierend auf der Suchanfrage zurückgegeben. Dabei hängt die Bewertung für die Relevanz eines Dokuments unter anderem von der Anzahl der gefundenen Suchwörter ab. Die Relevanz eines Dokuments für die letzte Suche kann aus dieser Eigenschaft ausgelesen werden. Sollte das Dokument bei der letzten Suche nicht als Ergebnis zurückgeliefert worden sein, so ist dieser Wert *0*.

HasEmbedded

Datentyp

boolean

Deklaration Getter

boolean hasEmbedded() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

In dieser Eigenschaft befindet sich der Wert *true*, wenn das Dokument eingebettete, gelinkte Objekte oder Dateianhänge besitzt. Ist das nicht der Fall, steht in dieser Eigenschaft der Wert *false*.

IsDeleted

Datentyp

boolean

Deklaration Getter

boolean isDeleted() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob das Dokument ein gültiges oder ein gelöschtes Dokument ist.

IsEncryptOnSend

Datentyp

boolean

Deklaration Getter

boolean isEncryptOnSend() throws NotesException

Deklaration Setter

void setEncryptOnSend(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft kann gesteuert werden, ob ein Dokument beim Versenden verschlüsselt werden soll. Wenn die Eigenschaft den Wert *true* besitzt, wird versucht, das Dokument an jeden Empfänger verschlüsselt zu senden. Dabei wird der jeweilige öffentliche Schlüssel des Empfängers zum Verschlüsseln verwendet. Sollte für einen Empfänger kein öffentlicher Schlüssel existieren, so wird das Dokument unverschlüsselt versendet.

IsNewNotes

Datentyp

boolean

Deklaration Getter

boolean isNewNote() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ein *true* in dieser Eigenschaft sagt aus, dass das Dokument neu erstellt und seitdem noch nicht gespeichert wurde. Ansonsten befindet sich in dieser Eigenschaft der Wert *false*.

IsProfile

Datentyp

boolean

Deklaration Getter

boolean isProfile() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Über diese Eigenschaft kann man erfahren, ob das Dokument ein Profildokument ist.

IsResponse

Datentyp

boolean

Deklaration Getter

boolean isResponse() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Bei einem Antwortdokument befindet sich der Wert *true* in dieser Eigenschaft, ansonsten der Wert *false*.

IsSaveMessageOnSend

Datentyp

boolean

Deklaration Getter

boolean isSaveMessageOnSend() throws NotesException

Deklaration Setter

`void setSaveMessageOnSend(boolean flag)` throws `NotesException`

Wertebereich

true, false

Beschreibung

Bei neu erstellten und noch nicht gespeicherten Dokumenten bestimmt diese Eigenschaft, ob sie bei dem Versenden als Mail vorher gespeichert werden.

IsSentByAgent

Datentyp

boolean

Deklaration Getter

`boolean isSentByAgent()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Bei einem Dokument, das gemailt wurde, gibt diese Eigenschaft Auskunft, ob der Mailversand durch ein Programm erfolgt ist. Ein per Agent verschicktes Dokument besitzt das versteckte Feld *\$AssistMail*.

IsSigned

Datentyp

boolean

Deklaration Getter

`boolean isSigned()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* besitzt, ist das Dokument unterschrieben.

IsSignOnSend

Datentyp

boolean

Deklaration Getter

boolean isSignOnSend() throws NotesException

Deklaration Setter

void setSignOnSend(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft kann gesteuert werden, ob das Dokument beim Versand unterschrieben wird.

IsValid

Datentyp

boolean

Deklaration Getter

boolean isValid() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft wird beim Erstellen der Objektinstanz gesetzt und gibt Auskunft darüber, ob das Dokument zu diesem Zeitpunkt ein gültiges, nicht gelöscht Dokument ist.

Items

Datentyp

java.util.Vector(Item)

Deklaration Getter

java.util.Vector.getItems() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkungen

Beschreibung

In dieser Eigenschaft befindet sich ein Vektor mit Referenzen für alle im Dokument enthaltenen Felder.

Key

Datentyp

String

Deklaration Getter

String getKey() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Bei Profildokumenten befindet sich in dieser Eigenschaft der Schlüssel für das Profildokument.

LastAccessed

Datentyp

DateTime

Deklaration Getter

DateTime getLastAccessed() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Datum in dieser Eigenschaft gibt Auskunft über den letzten Schreib- oder Lesezugriff auf dieses Dokument.

LastModified

Datentyp

DateTime

Deklaration Getter

DateTime getLastModified() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Datum in dieser Eigenschaft gibt Auskunft über den letzten Schreibzugriff auf dieses Dokument.

NameOfProfile

Datentyp

String

Deklaration Getter

String getNameOfProfile() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Bei einem Profildokument befindet sich in dieser Eigenschaft der Name des Profils.

NoteID

Datentyp

String

Deklaration Getter

String getNoteID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Note-ID des Dokuments.

ParentDatabase

Datentyp

Database

Deklaration Getter

Database getParentDatabase() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Referenz auf die Datenbank, in der sich das Dokument befindet.

ParentDocumentUNID

Datentyp

String

Deklaration Getter

String getParentDocumentUNID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Bei einem Dokument, das eine Antwort darstellt, befindet sich in dieser Eigenschaft die Universal-ID des übergeordneten Dokuments. Sollte es sich bei dem Dokument um kein Antwortdokument handeln, liefert diese Eigenschaft den Wert *null*.

ParentView

Datentyp

View

Deklaration Getter

View getParentView() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn das Dokument über eine Ansicht ermittelt wurde, befindet sich in dieser Eigenschaft eine Referenz auf diese Ansicht. Ansonsten erhält man den Wert *null*.

Response

Datentyp

DocumentCollection

Deklaration Getter

DocumentCollection getResponses() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Eine Liste aller Antwortdokumente zu dem jeweiligen Dokument befindet sich in dieser Eigenschaft. Sollten zu dem Dokument keine Antwortdokumente existieren, wird ein leerer Vektor geliefert.

Signer

Datentyp

String

Deklaration Getter

String getSigner() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Rückgabe von Wert null

Beschreibung

Wenn das Dokument unterschrieben wurde, beinhaltet diese Eigenschaft den Namen des Anwenders, der unterschrieben hat.

Size

Datentyp

int

Deklaration Getter

int getSize() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft lässt sich die Größe des Dokuments in Byte auslesen.

UniversalID

Datentyp

String

Deklaration Getter

String getUniversalID() throws NotesException

Deklaration Setter

void setUniversalID(String unid) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft kann die Universal-ID eines Dokuments gesetzt oder ausgelesen werden. Sollte die Universal-ID eines Dokuments geändert und das Dokument gespeichert werden, entsteht ein neues Dokument. Wenn versucht wird, in einer Datenbank zwei Dokumenten dieselbe ID zu geben, wird eine Exception geworfen.

Verifier

Datentyp

String

Deklaration Getter

String getVerifier() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung (null möglich)

Beschreibung

Der Name des Zertifikats für das Überprüfen der Unterschrift eines Dokuments lässt sich aus dieser Eigenschaft ermitteln. Wenn das Dokument nicht unterschrieben ist, liefert die Eigenschaft den Wert *null*.

9.9.2 Methoden

appendItemValue

Deklaration

Item appendItemValue(String name) throws NotesException

Item appendItemValue(String name, int value) throws NotesException

Item appendItemValue(String name, double value) throws NotesException

Item appendItemValue(String name, Object value) throws NotesException

Rückgabewert

Item

eine Referenz auf das Element

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements
value	ja	int	Wert, der in dem Element gespeichert werden soll
value	ja	double	Wert, der in dem Element gespeichert werden soll
value	ja	Object	Wert, der in dem Element gespeichert werden soll

Beschreibung

Mit dieser Methode kann ein neues Element in einem Dokument angelegt oder an die Werteliste für ein bestehendes Element ein Wert angehängt werden. Nach dem Aufruf liefert die Methode eine Referenz auf das Element zurück.

closeMIMEEntities

Deklaration

boolean closeMIMEEntities() throws NotesException

boolean closeMIMEEntities(boolean savechanges) throws NotesException

boolean closeMIMEEntities(boolean savechanges, String entityitemname) throws NotesException

Rückgabewert

boolean

Der Rückgabewert gibt Auskunft, ob das Schließen und ggf. das Speichern des MIME-Objekts erfolgreich war.

Parameter

Name	Optional	Typ	Beschreibung
savechange	ja	boolean	Dieser Parameter bestimmt, ob Änderungen in dem MIME-Objekt gespeichert werden oder verworfen werden sollen.
entityitemname	ja	String	Der Name des Elements, in dem das MIME-Objekt gespeichert werden soll. Standardmäßig wird der Name Body für das Element genommen.

Beschreibung

Mit dieser Methode wird ein geöffnetes MIME-Objekt geschlossen und ggf. gespeichert.

computeWithForm

Deklaration

boolean computeWithForm(boolean dodatatypes, boolean raiseerror) throws
NotesException

Rückgabewert

boolean

Als Rückgabewert wird true geliefert, wenn Validierung und Berechnung erfolgreich waren. Ansonsten wird der Wert false zurückgegeben.

Parameter

Name	Optional	Typ	Beschreibung
dodatatypes	nein	boolean	Dieser Parameter wird ignoriert.
raiseerror	nein	boolean	Wenn dieser Parameter den Wert true hat, führt ein Validierungsfehler zum Wurf einer Exception.

Beschreibung

Diese Methode führt zur Berechnung von Feldwerten anhand der Formeln für Standardwerte und einer Validierung des Dokuments. Wenn der Parameter raiseerror den Wert *true* besitzt, hat ein Widerspruch bei der Validierung den Wurf einer Exception zur Folge.

copyAllItems

Deklaration

void copyAllItems(Document doc, boolean replace) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Zieldokument, in das die Elemente kopiert werden sollen.
replace	nein	boolean	Bei true für diesen Parameter werden gleichnamige Elemente ersetzt. Sollte der Wert false angegeben werden, so werden die Werte an die bestehenden Werte angehängt.

Beschreibung

Mit dieser Methode werden alle Elemente des Dokuments in das angegebene Dokument kopiert. Sollten in dem Zieldokument bereits Elemente mit demselben Namen vorhanden sein, so werden diese bei dem Wert *true* für den Parameter `replace` ersetzt und bei dem Wert *false* für `replace` an die bestehenden Wertelisten angehängt.

copyItem

Deklaration

Item `copyItem(Item item)` throws `NotesException`

Item `copyItem(Item item, String newname)` throws `NotesException`

Rückgabewert

Item

eine Referenz auf das kopierte Element

Parameter

Name	Optional	Typ	Beschreibung
item	nein	Item	Referenz auf das Quellelement
newname	ja	String	Neuer Name des Elements in dem Zieldokument

Beschreibung

Mit dieser Methode wird das angegebene Element in das Dokument kopiert. Dabei kann mithilfe des zweiten Parameters gleichzeitig das Element umbenannt werden.

copyToDatabase

Deklaration

Document `copyToDatabase(Database db)` throws `NotesException`

Rückgabewert

Document

eine Referenz auf das neu erstellte Dokument in der Zieldatenbank

Parameter

Name	Optional	Typ	Beschreibung
db	nein	Database	Zieldatenbank

Beschreibung

Diese Methode kopiert das aktuelle Dokument in die angegebene Datenbank.

createMIMEEntity

Deklaration

MIMEEntity createMIMEEntity() throws NotesException

MIMEEntity createMIMEEntity(String itemName) throws NotesException

Rückgabewert

MIMEEntity

eine Referenz auf das neu erstellte MIME-Objekt

Parameter

Name	Optional	Typ	Beschreibung
itemName	ja	String	Name des Elements für die Aufnahme des MIME-Objekts. Standardmäßig ist das der Name Body.

Beschreibung

Diese Methode erzeugt ein neues MIME-Objekt und speichert es in dem Element mit dem angegebenen Namen oder standardmäßig in dem Element mit dem Namen „Body“.

createReplyMessage

Deklaration

Document createReplyMessage(boolean toall) throws NotesException

Rückgabewert

Document

das neu erstellte Antwortdokument

Parameter

Name	Optional	Typ	Beschreibung
toall	Nein	boolean	Bestimmt, ob die Antwort nur an den Sender gehen soll oder auch an alle anderen Empfänger des Dokuments

Beschreibung

Diese Methode erstellt für das aktuelle Dokument ein Antwortdokument. Dieses neu erstellte Dokument kann dann mit der Methode `send` versendet werden. Der Parameter steuert dabei, ob das Antwortdokument nur an den Sender des Dokuments oder auch an alle Empfänger geschickt werden soll.

createRichTextItem

Deklaration

`RichTextItem createRichTextItem(String name) throws NotesException`

Rückgabewert

`RichTextItem`

das neu erstellte RichText-Element

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des neuen RichText-Elements

Beschreibung

Mit dieser Methode wird ein neues RichText-Element mit dem angegebenen Namen in dem Dokument erstellt.

encrypt

Deklaration

`void encrypt() throws NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode dient zum Entschlüsseln von Dokumenten. Allerdings kann sie nicht für verschlüsselte Mails genutzt werden.

generateXML

Deklaration

String generateXML() throws NotesException

void generateXML(java.io.Writer w) throws NotesException, java.io.IOException

void generateXML(Object style, XSLTResultTarget result) throws NotesException, java.io.IOException, org.xml.sax.SAXException

Rückgabewert

String oder kein Rückgabewert

Die Zeichenkette ist der XML Code.

Parameter

Name	Optional	Typ	Beschreibung
w	ja	java.io.Writer	Der Stream, der den XML-Code erhalten soll
style	ja	Object	Der Stil der Transformation in XML
result	ja	XSLTResultTarget	Das Zielobjekt für den XML-Code

Beschreibung

Diese Methode liefert das in XML-transformierte Dokument.

getAttachment

Deklaration

EmbeddedObject getAttachment(String filename) throws NotesException

Rückgabewert

EmbeddedObject

eine Referenz auf den gesuchten Anhang.

Parameter

Name	Optional	Typ	Beschreibung
filename	nein	String	Name des Anhangs. der ermittelt werden soll

Beschreibung

Diese Methode ermittelt die angegebene angehängte Datei und liefert eine Referenz für den weiteren Zugriff darauf zurück.

getFirstItem

Deklaration

Item getFirstItem(String name) throws NotesException

Rückgabewert

Item

das erste Element mit dem angegebenen Namen

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode liefert eine Referenz auf das erste Element in dem Dokument mit dem angegebenen Namen.

getItemValue

Deklaration

`java.util.Vector getItemValue(String name) throws NotesException`

Rückgabewert

`java.util.Vector`

die Werteliste in Form eines Vektors des angegebenen Elements

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode liefert eine Liste in Form eines Vektors mit allen Werten des angegebenen Elements.

getItemValueCustomData

Deklaration

```
public Object getItemValueCustomData(String itemname)
    throws ClassNotFoundException, NotesException, java.io.IOException
public Object getItemValueCustomData(String itemname, String datatypename)
    throws ClassNotFoundException, NotesException, java.io.IOException
```

Rückgabewert

Object

der Inhalt des gesuchten Elements

Parameter

Name	Optional	Typ	Beschreibung
itemname	nein	String	Name des Elements
datatype	ja	String	Name des Datentyps des Elementinhalts. Der Datentyp beim Lesen muss dem Datentyp beim Schreiben entsprechen.

Beschreibung

Diese Methode liest den Inhalt aus dem Element und gibt ihn in Form des Datentyps, welcher bei dem Schreiben verwendet wurde, zurück.

getItemValueCustomDataBytes

Deklaration

```
public bytes[] getItemValueCustomDataBytes(String itemname,  
    String datatype) throws NotesException, java.io.IOException
```

Rückgabewert

bytes[]
der Inhalt des gesuchten Elements

Parameter

Name	Optional	Typ	Beschreibung
itemname	Nein	String	Name des Elements
datatype	Nein	String	Name des Datentyps des Elementinhalts. Der Datentyp beim Lesen muss dem Datentyp beim Schreiben entsprechen.

Beschreibung

Diese Methode liest den Inhalt aus dem Element und gibt ihn in Form des Datentyps, welcher bei dem Schreiben verwendet wurde, zurück.

getItemValueDouble

Deklaration

```
double getItemValueDouble(String name) throws NotesException
```

Rückgabewert

double
Der Inhalt des angegebenen Elements als double. Sollte kein Element mit dem Namen in dem Dokument vorhanden sein, wird 0.0 zurückgegeben.

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Mit dieser Methode wird der Inhalt eines Elements in Form eines *double*-Wertes ausgelesen.

getItemValueInteger

Deklaration

int getItemValueInteger(String name) throws NotesException

Rückgabewert

int

Der Inhalt des angegebenen Elements als int. Sollte kein Element mit dem Namen in dem Dokument vorhanden sein, wird 0 zurückgegeben.

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Mit dieser Methode wird der Inhalt eines Elements in Form eines *int*-Wertes ausgelesen.

getItemValueString

Deklaration

String getItemValueString(String name) throws NotesException

Rückgabewert

String

Der Inhalt des angegebenen Elements als String. Sollte kein Element mit dem Namen in dem Dokument vorhanden oder das bestehende nicht in eine Zeichenkette umwandelbar sein, so wird null zurückgegeben.

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode liest den Inhalt eines Elements aus und liefert, wenn möglich, diesen als Zeichenkette zurück.

getMIMEEntity

Deklaration

```
public MIMEEntity getMIMEEntity() throws NotesException  
public MIMEEntity getMIMEEntity(String name) throws NotesException
```

Rückgabewert

MIMEEntity

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode liefert eine Referenz auf das MIME-Objekt in dem angegebenen Element zurück. Sollte das Element nicht existieren oder nicht den entsprechenden Typ haben, so wird *null* zurückgegeben.

hasItem

Deklaration

```
boolean hasItem(String name) throws NotesException
```

Rückgabewert

boolean

Wenn ein Element mit dem angegebenen Namen existiert, wird *true* zurückgeliefert, ansonsten *false*.

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode überprüft, ob das Dokument mindestens ein Element mit dem angegebenen Namen besitzt.

makeResponse

Deklaration

`void makeResponse(Document doc) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Referenz auf das Dokument, welches das neue übergeordnete Dokument für das aktuelle Dokument sein soll.

Beschreibung

Das Dokument ordnet das aktuelle Dokument als Antwortdokument dem angegebenen Dokument unter. Dazu müssen sich beide Dokumente in derselben Datenbank befinden.

putInFolder

Deklaration

`void putInFolder(String name) throws NotesException`

`void putInFolder(String name, boolean createonfail) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Ordners
createonfail	ja	boolean	Bei true wird, falls der angegebene Ordner nicht existiert, dieser entsprechend angelegt.

Beschreibung

Diese Methode ordnet das Dokument in den angegebenen Ordner ein. Außerdem legt die Methode ggf. den Ordner neu an.

remove

Deklaration

`boolean remove(boolean force) throws NotesException`

Rückgabewert

boolean

Parameter

Name	Optional	Typ	Beschreibung
force	nein	boolean	Wenn dieser Parameter den Wert true hat, wird das Dokument auch entfernt, wenn inzwischen ein anderer Anwender es verändert hat.

Beschreibung

Diese Methode entfernt das Dokument aus der Datenbank.

removeFromFolder

Deklaration

void removeFromFolder(String name) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Ordners

Beschreibung

Mit dieser Methode wird das Dokument aus einem Ordner entfernt.

removeItem

Deklaration

void removeItem(String name) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements

Beschreibung

Diese Methode entfernt das angegebene Element aus dem Dokument. Diese Änderung muss mit der Methode save gespeichert werden.

renderToRTItem

Deklaration

boolean renderToRTItem(RichTextItem ritem) throws NotesException

Rückgabewert

boolean

Parameter

Name	Optional	Typ	Beschreibung
ritem	nein	RichTextItem	Referenz auf das Zielelement

Beschreibung

Diese Methode erzeugt ein Abbild des aktuellen Dokuments und speichert dieses in dem angegebenen RichText-Element ab.

replacelItemValue

Deklaration

Item replacelItemValue(String name, Object value) throws NotesException

Rückgabewert

Item

eine Referenz auf das Element

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Elements
value	nein	Object	Neuer Inhalt des Elements

Beschreibung

Mit dieser Methode wird der angegebene Wert in das angegebene Element geschrieben und ggf. ältere Inhalte in dem Element überschrieben. Sollte das Element noch nicht existieren, wird es jetzt angelegt.

replacelItemValueCustomData

Deklaration

void replacelItemValueCustomData(String itemname, Object userobj) throws NotesException, java.io.IOException

void replacelItemValueCustomData(String itemname, String datatypename, Object userobj) throws NotesException, java.io.IOException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
itemname	nein	String	Name des Elements
datatype	nein	String	Name des Datentyps bzw. Klassenname
userobj	nein	Object	Der neue Wert für das Element

Beschreibung

Mit dieser Methode können benutzerdefinierte Daten in ein Item geschrieben werden.

replaceItemValueCustomDataBytes

Deklaration

```
void replaceItemValueCustomData(String itemname, String datatype, byte[]  
bytearray) throws NotesException, java.io.IOException
```

Rückgabewert

kein

Parameter

String itemname
String datatype
byte[] bytearray

Beschreibung

Mit dieser Methode können benutzerdefinierte Daten in ein Item geschrieben werden.

save

Deklaration

```
boolean save() throws NotesException  
boolean save(boolean force) throws NotesException  
boolean save(boolean force, boolean makeResponse) throws NotesException  
boolean save(boolean force, boolean makeResponse, boolean markRead) throws  
NotesException
```

Rückgabewert

boolean
gibt Auskunft, ob das Dokument erfolgreich gespeichert werden konnte

Parameter

Name	Optional	Typ	Beschreibung
force	nein	boolean	Bei dem Wert false gibt der Parameter make-response an, wie das Speichern ablaufen soll. Sollte der Wert force true sein, so kann das Dokument nur gespeichert werden, wenn es inzwischen von keinem anderen Anwender geändert wurde. Wenn doch eine Änderung gemacht wurde, erfolgt keine Speicherung. Bei dem Wert true wird der Parameter makesresponse ignoriert.
makesresponse	nein	boolean	Wenn true, wird nach einer Änderung in dem Dokument die neue Version als Antwortdokument der alten Version gespeichert.
markread	nein	boolean	Wenn true, wird das neue Dokument als gelesen markiert und ansonsten als ungelesen.

Beschreibung

Diese Methode speichert das Dokument.

send

Deklaration

void send() throws NotesException

void send(String recipient) throws NotesException

void send(java.util.Vector recipients) throws NotesException

void send(boolean attachform) throws NotesException

void send(boolean attachform, String recipient) throws NotesException

void send(boolean attachform, java.util.Vector recipients) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
recipient	ja	String	Empfänger für das Dokument
recipients	ja	java.util.Vector	Liste mit Empfängern für das Dokument
attachform	ja	boolean	Wenn true, wird die Maske beim Versand in dem Dokument gespeichert.

Beschreibung

Diese Methode versendet das Dokument an den oder die angegebenen Empfänger.

sign

Deklaration

void sign() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode unterschreibt das Dokument.

9.10 Klasse DocumentCollection

9.10.1 Eigenschaften

Count

Datentyp

int

Deklaration Getter

int getCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Bechreibung

In dieser Eigenschaft steht die Anzahl der Dokumente in der Dokumentensammlung.

IsSorted

Datentyp

boolean

Deklaration Getter

boolean isSorted() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob die Dokumentensammlung sortiert ist.

Parent

Datentyp

Database

Deklaration Getter

Database getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft enthält eine Referenz auf die Datenbank, in der sich die Dokumente der Dokumentensammlung befinden.

Query

Datentyp

String

Deklaration Getter

String getQuery() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet, falls vorhanden, die Formel, die bei der Erstellung der Dokumentensammlung benutzt wurde.

9.10.2 Methoden

addDocument

Deklaration

void addDocument(Document doc) throws NotesException

void addDocument(Document doc, boolean checkDups) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Dokument, das hinzugefügt werden soll
checkDups	ja	boolean	Steuert bei Zugriff über IIOp, ob das Dokument sofort hinzugefügt wird oder die Änderung erst nur im Cache stattfindet.

Beschreibung

Diese Methode fügt ein angegebenes Dokument zur Dokumentensammlung hinzu.

deleteDocument

Deklaration

void deleteDocument(Document doc) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Zu löschendes Dokument

Beschreibung

Diese Methode löscht ein angegebenes Dokument aus der Dokumentensammlung.

FTSearch

Deklaration

void FTSearch(String query) throws NotesException

void FTSearch(String query, int maxdocs) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Formel für die Suche
maxdocs	nein	int	Maximale Anzahl der auszugebenden Dokumente. Dabei steht 0 für keine Einschränkung. Maximal werden 2.147.483.647 Dokumente geliefert.

Beschreibung

Diese Methode führt eine Volltextsuche über die Dokumente der Dokumentensammlung durch und dezimiert somit die Auswahl an Dokumenten.

getDocument

Deklaration

Document getDocument(Document doc) throws NotesException

Rückgabewert

Document

Wenn vorhanden, eine Referenz auf das gesuchte Dokument in der Dokumentensammlung

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Dokument, das in der Collection gesucht werden soll

Beschreibung

Diese Methode gibt, falls in der Dokumentensammlung vorhanden, das angegebene Dokument zurück. Wenn das Dokument nicht vorhanden ist, wird der Wert null zurückgegeben.

getFirstDocument

Deklaration

Document getFirstDocument() throws NotesException

Rückgabewert

Document

erstes Dokument in der Dokumentensammlung.

Parameter

keine

Beschreibung

Diese Methode liefert eine Referenz auf das erste Dokument in der Dokumentensammlung.

getLastDocument**Deklaration**

Document getLastDocument() throws NotesException

Rückgabewert

Document

letztes Dokument in der Dokumentensammlung

Parameter

keine

Beschreibung

Diese Methode liefert das letzte Dokument aus der Dokumentensammlung zurück.

getNextDocument**Deklaration**

Document getNextDocument() throws NotesException

Document getNextDocument(Document doc) throws NotesException

Rückgabewert

Document

das folgende Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	ja	Document	Das Dokument, dessen nachfolgendes Dokument ermittelt werden soll.

Beschreibung

Diese Methode liefert das dem angegebenen Dokument folgende Dokument in der Dokumentensammlung zurück. Wenn kein Dokument angegeben wird, so wird das Dokument zurückgeliefert, das dem zurückgelieferten Dokument beim letzten Aufruf dieser Methode folgt.

getNthDocument

Deklaration

Document getNthDocument(int n) throws NotesException

Rückgabewert

Document

das Dokument an der angegebenen Position in der Dokumentensammlung

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Die Position des gewünschten Dokuments

Beschreibung

Diese Methode liefert das Dokument zurück, das an der angegebenen Position in der Dokumentensammlung steht.

getPrevDocument

Deklaration

Document getPrevDocument() throws NotesException

Document getPrevDocument(Document doc) throws NotesException

Rückgabewert

Document

das Vorgängerdokument

Parameter

Name	Optional	Typ	Beschreibung
doc	ja	Document	Das Dokument, dessen Vorgängerdokument ermittelt werden soll

Beschreibung

Diese Methode liefert das vor dem angegebenen Dokument stehende Dokument aus der Dokumentensammlung zurück. Wenn kein Dokument angegeben wird, so wird der Vorgänger des beim letzten Aufruf dieser Methode gelieferten Dokuments geliefert.

putAllInFolder

Deklaration

void putAllInFolder(String folderName) throws NotesException

void putAllInFolder(String folderName, boolean createonfail) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
folgerName	nein	String	Name des Ordners
createonfail	nein	boolean	Bestimmt, ob der Ordner angelegt werden soll, falls noch nicht vorhanden

Beschreibung

Diese Methode sorgt dafür, dass alle Dokumente der Dokumentensammlung in den angegebenen Ordner eingefügt werden.

removeAll

Deklaration

`void removeAll(boolean force) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
force	nein	boolean	Bestimmt, ob auch die Dokumente gelöscht werden sollen, die inzwischen von einem anderen Anwender verändert worden sind

Beschreibung

Diese Methode löscht alle Dokumente der Dokumentensammlung aus der Datenbank.

removeAllFromFolder

Deklaration

`void removeAllFromFolder(String foldername) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
foldername	nein	String	Name des Ordners

Beschreibung

Diese Methode entfernt alle Dokumente der Dokumentensammlung aus dem angegebenen Ordner.

stampAll

Deklaration

void stampAll(String itemname, Object value) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
itemname	nein	String	Name des Elements
value	nein	Object	Neuer Wert für das Element

Beschreibung

Diese Methode schreibt den angegebenen Wert in jedes Dokument in das Element mit dem angegebenen Namen. Sollte das Element noch nicht vorhanden sein, so wird es angelegt.

updateAll

Deklaration

void updateAll() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Durch den Aufruf dieser Methode werden alle Dokumente in der Dokumentensammlung als von dem aktuellen Agent bearbeitet markiert.

9.11 Klasse EmbeddedObject

9.11.1 Konstanten

Name	Wert
EMBED_ATTACHMENT	1454
EMBED_OBJECT	1453
EMBED_OBJECTLINK	1452

9.11.2 Eigenschaften

ClassName

Datentyp

String

Deklaration Getter

String getClassName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft lässt sich der Name der Anwendung lesen, mit der das Objekt angelegt wurde. Für Dateianhänge liefert der Aufruf einen Leerstring zurück.

FileSize

Datentyp

integer

Deklaration Getter

int getFileSize() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Größe eines Anhangs in Byte.

Name

Datentyp

Object(String)

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft findet man den Namen des eingebetteten Objekts.

Object

Datentyp

int

Deklaration Getter

int getObject() throws NotesException

Deklaration Setter

keiner

Wertebereich

Keine Einschränkung

Beschreibung

Aus dieser Eigenschaft bekommt man einen Handle für den Zugriff auf das Objekt.

Parent

Datentyp

RichTextItem

Deklaration Getter

RichTextItem getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert – falls vorhanden – eine Referenz auf das RichText-Element, in dem das Objekt gespeichert ist.

Source

Datentyp

String

Deklaration Getter

String getSource() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt dieser Eigenschaft ist der interne Name des Objekts.

Type

Datentyp

int

Deklaration Getter

int getType() throws NotesException

Deklaration Setter

kein

Wertebereich

Wert	Bedeutung
EmbeddedObject.EMBED_ATTACHMENT	Dateianhang
EmbeddedObject.EMBED_OBJECT	Eingebettetes Objekt
EmbeddedObject.EMBED_OBJECTLINK	Verknüpftes Objekt

Beschreibung

Über diese Eigenschaft kann der Typ des Objekts bestimmt werden.

Verbs

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getVerbs() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Liste der möglichen Aktionen für das Objekt.

9.11.3 Methoden

activate

Deklaration

int activate(boolean show) throws NotesException

Rückgabewert

int

Es wird ein Handle für den Zugriff auf das Objekt zurückgegeben oder der Wert 0 als Kennzeichen für einen Fehler.

Parameter

Name	Optional	Typ	Beschreibung
show	nein	boolean	Die Serveranwendung des Objekts wird bei true angezeigt und bei false versteckt.

Beschreibung

Diese Methode öffnet das Objekt mithilfe der Serveranwendung für das Objekt. Die Serveranwendung ist die Anwendung, mit der das Objekt erstellt wurde.

doVerb

Deklaration

void doVerb(String verb) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
verb	nein	String	Aktion, die ausgeführt werden soll

Beschreibung

Die angegebene Aktion wird für das Objekt ausgeführt und ggf. die entsprechende Serveranwendung gestartet. Die Serveranwendung ist meist die Anwendung, mit der das Objekt erstellt wurde.

extractFile

Deklaration

void extractFile(String path) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
path	nein	String	Zieldatei für den Dateianhang

Beschreibung

Diese Methode extrahiert einen Dateianhang und schreibt ihn in die angegebene Datei.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode entfernt das Objekt aus dem Dokument. Danach muss zum Speichern dieser Änderung die Methode `save` aufgerufen werden.

9.12 Klasse Form

9.12.1 Eigenschaften

Aliases

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getAliases() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkungen

Beschreibung

In dieser Eigenschaft befinden sich die Aliasnamen für die Maske.

Fields

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getFields() throws NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich eine Liste aller Felder einer Maske in Form eines Vektors.

FormUsers**Datentyp**

java.util.Vector(String)

Deklaration Getter

java.util.Vector getFormUsers() throws NotesException

Deklaration Setter

void setFormUsers(java.util.Vector users) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt dieser Eigenschaft ist identisch mit dem Inhalt des Felds *\$FormUsers*. Dieses Feld bestimmt, wer mit der Maske Dokumente erstellen darf.

IsProtectReaders**Datentyp**

boolean

Deklaration Getter

boolean isProtectReaders() throws NotesException

Deklaration Setter

void setProtectReaders(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* hat, wird das Überschreiben des Felds *\$Readers* bei der Replikation verhindert. Ansonsten kann der Inhalt des Felds bei der Replikation überschrieben werden.

IsProtectUsers

Datentyp

boolean

Deklaration Getter

boolean isProtectUsers() throws NotesException

Deklaration Setter

void setProtectUsers(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* hat, wird das Überschreiben des Felds *\$Form-Users* bei der Replikation verhindert. Ansonsten kann der Inhalt des Felds bei der Replikation überschrieben werden.

IsSubForm

Datentyp

boolean

Deklaration Getter

boolean isSubForm() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Anhand des Inhalts dieser Eigenschaft lässt sich bestimmen, ob es sich bei der Maske um eine Teilmaske handelt.

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name der Maske.

Parent

Datentyp

Database

Deklaration Getter

Database getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft verweist auf die Datenbank, in der sich die Maske befindet.

Readers

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getReaders() throws NotesException

Deklaration Setter

void setReaders(java.util.Vector readers) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft findet man den Inhalt der Felds *\$Readers* wieder.

9.12.2 Methoden

getFieldType

Deklaration

int getFieldType(String name) throws NotesException

Rückgabewert

int

der Typ des angegebenen Felds als numerischer Wert

Konstante	Feldtyp
Item.ACTIONCD	Actioncd
Item.ASSISTANTINFO	Assistantinfo
Item.ATTACHMENT	Dateianhang
Item.AUTHORS	Autorenfeld
Item.COLLATION	Collation
Item.DATETIMES	Datums- und Zeitwert
Item.EMBEDDEDOBJECT	Eingebettetes Objekt
Item.ERRORITEM	Fehlerhafter Datentyp
Item.FORMULA (Domino formula)	Notes Formel
Item.HTML (HTML source text)	HTML-Text
Item.ICON	Icon
Item.LSOBJECT	LS-Objekt
Item.MIME_PART	Mime_Part
Item.NAMES	Name
Item.NOTELINKS	Notes Link
Item.NOTEREFS	Dokumentenreferenz
Item.NUMBERS	Zahl oder Liste von Zahlen
Item.OTHEROBJECT	Objekt
Item.QUERYCD	Querycd
Item.RICHTEXT	RichText
Item.READERS	Leser
Item.SIGNATURE	Unterschrift
Item.TEXT (text or text list)	Text
Item.UNAVAILABLE	Nicht verfügbar
Item.UNKNOWN	Unbekannt
Item.USERDATA	Benutzerdefiniert
Item.USERID	Benutzer-ID
Item.VIEWMAPDATA	Viewmapdata
Item.VIEWMAPLAYOUT	Viewmaplayout

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des Felds

Beschreibung

Diese Methode liefert den Typ des angegebenen Felds zurück.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode wird die Maske aus der Datenbank entfernt.

9.13 Klasse International

9.13.1 Eigenschaften

AMString

Datentyp

String

Deklaration Getter

String getAMString() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Zeichenkette, die den Nachmittag kennzeichnet.

CurrencyDigits

Datentyp

int

Deklaration Getter

int getCurrencyDigits() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Anzahl der Nachkommastellen für den Dezimalpunkt bei Währungsangaben.

CurrencySymbol

Datentyp

Object(String)

Deklaration Getter

String getCurrencySymbol() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann das Währungssymbol ermittelt werden.

DateSep

Datentyp

String

Deklaration Getter

String getDateSep() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich das Trennzeichen für die Datumsangabe.

DecimalSep

Datentyp

String

Deklaration Getter

String getDecimalSep() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich das Zeichen für den Dezimalpunkt.

IsCurrencySpace

Datentyp

boolean

Deklaration Getter

boolean isCurrencySpace() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Der Inhalt dieser Eigenschaft sagt aus, dass sich zwischen dem Währungssymbol und der Zahl ein Leerzeichen befindet.

IsCurrencySuffix

Datentyp

boolean

Deklaration Getter

boolean isCurrencySuffix() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Der Inhalt diese Eigenschaft bestimmt, ob bei der Währungsangabe das Symbol hinter dem Zahlenwert steht.

IsCurrencyZero

Datentyp

boolean

Deklaration Getter

boolean isCurrencyZero() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Durch den Inhalt dieser Eigenschaft wird bestimmt, ob sich bei der Währungsangabe immer eine führende 0 vor dem Dezimaltrennzeichen befinden muss.

IsDateDMY

Datentyp

boolean

Deklaration Getter

boolean isDateDMY() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Der Inhalt dieser Eigenschaft verrät, ob die Reihenfolge bei der Datumsangabe „Tag, Monat, Jahr“ ist.

IsDateMDY

Datentyp

boolean

Deklaration Getter

boolean isDateMDY() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Der Inhalt dieser Eigenschaft verrät, ob die Reihenfolge bei der Datumsangabe „Monat, Tag, Jahr“ ist.

IsDataYMD

Datentyp

boolean

Deklaration Getter

boolean isDateYMD() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Der Inhalt dieser Eigenschaft verrät, ob die Reihenfolge bei der Datumsangabe „Jahr, Monat, Tag“ ist.

IsDST

Datentyp

boolean

Deklaration Getter

boolean isDST() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft, ob die Sommerzeit berücksichtigt wird.

IsTime24Hour

Datentyp

boolean

Deklaration Getter

boolean isTime24Hour() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob die Uhrzeit im 24-Stunden-, und nicht im 12-Stunden-Format angegeben wird, bestimmt diese Eigenschaft.

Parent

Datentyp

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Referenz auf die Session, in der diese Instanz der Klasse *International* erstellt wurde.

PMString

Datentyp

String

Deklaration Getter

String getPMString() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Zeichenkette, die den Vormittag kennzeichnet.

ThousandsSep

Datentyp

String

Deklaration Getter

String getThousandsSep() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich das Zeichen für den Tausenderpunkt.

TimeSep

Datentyp

String

Deklaration Getter

String getTimeSep() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Separator für die Zeitangabe.

TimeZone

Datentyp

int

Deklaration Getter

int getTimeZone() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt dieser Eigenschaft gibt an, wie viele Stunden zu der GMT-Zeit addiert oder von dieser subtrahiert werden müssen, um die entsprechende Zeitzone zu erhalten.

Today

Datentyp

String

Deklaration Getter

String getToday() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Zeichenkette, die den heutigen Tag in der entsprechenden Sprache bezeichnet.

Tomorrow**Datentyp**

String

Deklaration Getter

String getTomorrow() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Zeichenkette, die den morgigen Tag in der entsprechenden Sprache bezeichnet.

Yesterday**Datentyp**

String

Deklaration Getter

String getYesterday() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Zeichenkette, die den gestrigen Tag in der entsprechenden Sprache bezeichnet.

9.13.2 Methoden

Keine

9.14 Klasse Item

9.14.1 Konstanten

Name	Wert
ACTIONCD	16
ASSISTANTINFO	17
ATTACHMENT	1084
AUTHORS	1076
COLLATION	2
DATETIMES	1024
EMBEDDEDOBJECT	1090
ERRORITEM	256
FORMULA	1536
HTML	21
ICON	6
LSOBJECT	20
MIME_PART	25
NAMES	1074
NOTELINKS	7
NOTEREFS	4
NUMBERS	768
OTHEROBJECT	1085
QUERYCD	15
READERS	1075
RICHTEXT	1
SIGNATURE	8
TEXT	1280

Name	Wert
UNAVAILABLE	512
UNKNOWN	0
USERDATA	14
USERID	1792
VIEWMAPDATA	18
VIEWMAPLAYOUT	19

9.14.2 Eigenschaften

DateTimeValue

Datentyp

DateTime

Deklaration Getter

DateTime getDateTimeValue() throws NotesException

Deklaration Setter

void setDateTimeValue(DateTime value) throws NotesException

Wertebereich

Diese Eigenschaft dient zum Lesen und Schreiben von Elementen in Form eines Datums- und Zeitwerts.

IsAuthors

Datentyp

boolean

Deklaration Getter

boolean isAuthors() throws NotesException

Deklaration Setter

void setAuthors(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft liefert den Wert *true*, wenn es sich um ein Autorenfeld handelt.

IsEncrypted

Datentyp

boolean

Deklaration Getter

boolean isEncrypted() throws NotesException

Deklaration Setter

void setEncrypted(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob ein Element verschlüsselt ist oder nicht.

IsNames

Datentyp

boolean

Deklaration Getter

boolean isNames() throws NotesException

Deklaration Setter

setNames(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob das Element vom Typ Namen ist.

IsProtected

Datentyp

boolean

Deklaration Getter

boolean isProtected() throws NotesException

Deklaration Setter

void setProtected(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob der Anwender Schreibrechte benötigt, um das Element zu ändern oder ob Autorenrechte genügen. Der Wert *true* gibt an, dass das Element geschützt ist und nur mit Schreibrechten geändert werden darf.

IsReaders**Datentyp**

boolean

Deklaration Getter

boolean isReaders() throws NotesException

Deklaration Setter

void setReaders(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft liefert *true*, wenn es sich bei dem Element um eines mit dem Typ *Leser* handelt.

IsSaveToDisk**Datentyp**

boolean

Deklaration Getter

boolean isSaveToDisk() throws NotesException

Deklaration Setter

void setSaveToDisk(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird bestimmt, ob das Element beim Speichern des Dokuments mitgespeichert wird.

IsSigned

Datentyp

boolean

Deklaration Getter

boolean isSigned() throws NotesException

Deklaration Setter

void setSigned(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob das Element eine Unterschrift besitzt.

IsSummary

Datentyp

boolean

Deklaration Getter

boolean isSummary() throws NotesException

Deklaration Setter

void setSummary(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob das Element als Bestandteil der Dokumentenübersicht behandelt werden soll. Nur Elemente in der Dokumentenübersicht können in Ansichten verwendet werden.

LastModified

Datentyp

DateTime

Deklaration Getter

DateTime getLastModified() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet das letzte Änderungsdatum des Elements.

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name des Elements.

Parent

Datentyp

Document

Deklaration Getter

Document getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft gelangt man zu einer Referenz auf das Dokument, in dem sich das Element befindet.

Text

Datentyp

String

Deklaration Getter

String getText() throws NotesException

String getText(int maxlen) throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert den Inhalt des Elements als Zeichenkette.

Type

Datentyp

int

Deklaration Getter

int getType() throws NotesException

Deklaration Setter

kein

Wertebereich

Konstante	Feldtyp
Item.ACTIONCD	Actioncd
Item.ASSISTANTINFO	Assistantinfo
Item.ATTACHMENT	Dateianhang
Item.AUTHORS	Autorenfeld
Item.COLLATION	Kollation
Item.DATETIMES	Datums- und Zeitwert
Item.EMBEDDEDOBJECT	Eingebettetes Objekt
Item.ERRORITEM	Fehlerhafter Datentyp
Item.FORMULA (Domino formula)	Notes Formel
Item.HTML (HTML source text)	HTML-Text

Konstante	Feldtyp
Item.ICON	Icon
Item.LSOBJECT	LS-Objekt
Item.MIME_PART	Mime_Part
Item.NAMES	Name
Item.NOTELINKS	Notes Link
Item.NOTEREFS	Dokumentenreferenz
Item.NUMBERS	Zahl oder Liste von Zahlen
Item.OTHEROBJECT	Objekt
Item.QUERYCD	Querycd
Item.RICHTEXT	RichText
Item.READERS	Leser
Item.SIGNATURE	Unterschrift
Item.TEXT (text or text list)	Text
Item.UNAVAILABLE	Nicht verfügbar
Item.UNKNOWN	Unbekannt
Item.USERDATA	Benutzerdefiniert
Item.USERID	Benutzer-ID
Item.VIEWMAPDATA	Viewmapdata
Item.VIEWMAPLAYOUT	Viewmaplayout

Beschreibung

Diese Eigenschaft verrät den Typ des Elements.

ValueDouble

Datentyp

double

Deklaration Getter

double getValueDouble() throws NotesException

Deklaration Setter

void setValueDouble(double value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann der Inhalt des Elements in Form eines Double-Werts gelesen werden.

ValueInteger

Datentyp

int

Deklaration Getter

int getValueInteger() throws NotesException

Deklaration Setter

void setValueInteger(int value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann der Inhalt des Elements in Form eines int-Werts gelesen werden.

ValueLength

Datentyp

int

Deklaration Getter

int getValueLength() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Speicherbedarf des Elementinhalts in Byte befindet sich in dieser Eigenschaft.

Values

Datentyp

java.util.Vector(String)

java.util.Vector(Strings)

java.util.Vector(Doubles)

java.util.Vector(DateTimes)

Deklaration Getter

java.util.Vector getValues() throws NotesException

Deklaration Setter

void setValues(java.util.Vector values) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Das Schreiben und Lesen des Elementinhalts in Form verschiedener Datentypen ist über diese Eigenschaft möglich.

ValueString

Datentyp

String

Deklaration Getter

String getValueString() throws NotesException

Deklaration Setter

void setValueString(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft erlaubt das Auslesen bzw. Setzen eines Textfelds oder eines Rich-Text-Felds als String.

9.14.3 Methoden

abstractText

Deklaration

String abstractText(int maxlen, boolean dropvowels, boolean userdict) throws NotesException

Rückgabewert

String

der Inhalt in Form eines Texts

Parameter

Name	Optional	Typ	Beschreibung
maxlen	nein	int	Maximale Länge des Texts
dropvowels	nein	boolean	Wenn der Parameter den Wert true hat, werden die Vokale entfernt.
userdict	nein	boolean	Bei dem Wert true wird die Datei NOTEABBR.TXT zur Umsetzung der Abkürzungen verwendet.

Beschreibung

Mit dieser Methode wird der Inhalt des Items in Form eines Texts aufbereitet zurückgegeben.

appendToTextList

Deklaration

`void appendToTextList(String value) throws NotesException`

`void appendToTextList(java.util.Vector values) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
value	nein	String	Wert oder Werte, die an den aktuellen Wert des Items angehängt werden
values		Vector	

Beschreibung

Diese Methode dient dazu, einen oder mehrere Werte an den aktuellen Wert des Items anzuhängen.

containsValue

Deklaration

`boolean containsValue(Object value) throws NotesException`

Rückgabewert

boolean

Es wird true zurück gegeben, wenn das Item den angegebenen Wert beinhaltet, ansonsten false.

Parameter

Name	Optional	Typ	Beschreibung
value	nein	Object	Der Wert, nach dem gesucht werden soll

Beschreibung

Durch diese Methode kann ermittelt werden, ob das Item einen bestimmten Wert beinhaltet.

copyItemToDocument

Deklaration

Item copyItemToDocument(Document doc) throws NotesException

Item copyItemToDocument(Document doc, String newname) throws NotesException

Rückgabewert

Item

Zurückgeliefert wird das neu angelegte Item.

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Dokument, in das das Item hineinkopiert werden soll
newname	ja	String	Name des neuen Items im Zieldokument

Beschreibung

Beim Aufruf dieser Methode wird das Item in das angegebene Dokument kopiert.

getValueCustomData

Deklaration

Object getValueCustomData() throws ClassNotFoundException, NotesException, java.io.IOException

Object getValueCustomData(String datatypename) throws ClassNotFoundException, NotesException, java.io.IOException

Rückgabewert

Object

Zurückgegeben wird der Inhalt des Intems in dem angegebenen Datentyp.

Parameter

Name	Optional	Typ	Beschreibung
datatypename	ja	String	Name des Datentyps

Beschreibung

Diese Methode ist eine Neuerung in der neuen Version von Notes. Mit ihr können aus Items Werte mit benutzerdefiniertem Typ ausgelesen werden.

getValueCustomDataBytes

Deklaration

```
byte[] getValueCustomDataBytes(String datatypeName) throws NotesException,  
    java.io.IOException
```

Rückgabewert

byte[]

Der Inhalt des Items wird mit dem angegebenen Datentyp interpretiert und als Array von Bytes zurückgegeben.

Parameter

Name	Optional	Typ	Beschreibung
datatypeName	nein	String	Name des Datentyps

Beschreibung

Die Handhabung dieser Methode ist eine Neuerung in der neuen Version von Notes. Mit ihr können aus Items Werte mit benutzerdefiniertem Typ ausgelesen werden.

getMIMEEntity

Deklaration

```
MIMEEntity getMIMEEntity() throws NotesException
```

Rückgabewert

MIMEEntity

eine Referenz auf die gelesene MIME-Entity

Parameter

keine

Beschreibung

Die Methode dient zum Lesen von MIME-Entities aus dem Item.

remove

Deklaration

```
void remove() throws NotesException
```

Rückgabewert

kein

Parameter

keine

Beschreibung

Beim Aufruf dieser Methode wird das Item aus dem Dokument entfernt.

setValueCustomData

Deklaration

void setValueCustomData(Object userobj) throws NotesException,
java.io.IOException

void setValueCustomData(String datatypename, Object userobj) throws
NotesException, java.io.IOException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
datatypename	ja	String	Name des Datentyps
userobj	nein	Object	Wert, der gespeichert werden soll

Beschreibung

Diese Methode ist neu und dient zum Speichern von benutzerdefinierten Datentypen.

setValueCustomDataBytes

Deklaration

void setValueCustomData(String datatypename, Byte bytearray) throws
NotesException, java.io.IOException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
datatypename	nein	String	Name des Datentyps
bytearray	nein	Byte	Wert, der gespeichert werden soll

Beschreibung

Diese Methode ist neu und dient zum Speichern von benutzerdefinierten Datentypen.

9.15 Klasse Log

9.15.1 Konstanten

Name	Wert
EV_ALARM	8
EV_COMM	1
EV_MAIL	3
EV_MISC	6
EV_REPLICA	4
EV_RESOURCE	5
EV_SECURITY	2
EV_SERVER	7
EV_UNKNOWN	0
EV_UPDATE	9
SEV_FAILURE	2
SEV_FATAL	1
SEV_NORMAL	5
SEV_UNKNOWN	0
SEV_WARNING1	3
SEV_WARNING2	4

9.15.2 Eigenschaften

IsLogActions

Datentyp

boolean

Deklaration Getter

boolean isLogActions() throws NotesException

Deklaration Setter

void setLogActions(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Über diese Eigenschaft kann gesteuert werden, ob über diese Instanz ein Logging von Nachrichten stattfindet oder nicht. Wenn diese Eigenschaft den Wert *true* hat, erfolgt ein Logging.

IsLogErrors

Datentyp

boolean

Deklaration Getter

boolean isLogErrors() throws NotesException

Deklaration Setter

void setLogErrors(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Über diese Eigenschaft wird gesteuert, ob Fehlermeldungen in der dieser Instanz der *Log*-Klasse geloggt werden. Wenn die Eigenschaft den Wert *true* hat, werden Fehlermeldungen geloggt, bei dem Wert *false* nicht.

IsOverwriteFile

Datentyp

boolean

Deklaration Getter

boolean isOverwriteFile() throws NotesException

Deklaration Setter

void setOverwriteFile(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Über diese Eigenschaft wird bei einem Logging in eine Datei gesteuert, ob die bestehende Datei überschrieben werden soll oder die Einträge an die bestehende Datei angehängt werden sollen.

NumActions

Datentyp

int

Deklaration Getter

int getNumActions() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt dieser Eigenschaft verrät, wie viele Nachrichten bereits geloggt wurden.

NumErrors

Datentyp

int

Deklaration Getter

int getNumErrors() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt dieser Eigenschaft verrät, wie viele Fehlermeldungen bereits geloggt wurden.

Parent

Datentyp

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich eine Referenz auf die Notes Session, in der die *Log*-Instanz erzeugt wurde.

ProgramName

Datentyp

Object(String)

Deklaration Getter

String getProgramName() throws NotesException

Deklaration Setter

void setProgramName(String name) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Programmname, der genutzt werden kann, um die Quelle des Loggings später zu identifizieren. Es kann ein frei gewählter Name angegeben werden.

9.15.3 Methoden

close

Deklaration

void close() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode schließt das Logging. Sollte als Loggingmethode `Mails` gewählt worden sein, so wird durch den Aufruf dieser Methode die Mail abgeschickt.

logAction

Deklaration

```
void logAction(String action) throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
action	nein	String	Nachricht, die protokolliert werden soll

Beschreibung

Diese Methode protokolliert die angegebene Nachricht.

logError

Deklaration

```
void logError(int code, String text) throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
code	nein	int	Fehlercode, der protokolliert werden soll
text	nein	String	Fehlermeldung, die protokolliert werden soll

Beschreibung

Diese Methode protokolliert die angegebene Fehlermeldung.

logEvent

Deklaration

```
void logEvent(String text, String queue, int event, int severity) throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
text	nein	String	Zu protokollierender Text
queue	nein	String	Name der Protokollqueue
event	nein	int	Art des zu protokollierenden Ereignisses: Log.EV_ALARM Log.EV_COMM Log.EV_MAIL Log.EV_MISC Log.EV_REPLICA Log.EV_RESOURCE Log.EV_SECURITY Log.EV_SERVER Log.EV_UNKNOWN Log.EV_UPDATE
severity	nein	int	Gefahrenstufe des zu protokollierenden Ereignisses: Log.SEV_FAILURE Log.SEV_FATAL Log.SEV_NORMAL Log.SEV_WARNING1 Log.SEV_WARNING2 Log.SEV_UNKNOWN

Beschreibung

Diese Methode dient zur Ereignisprotokollierung über das Netzwerk.

openAgentLog

Deklaration

```
void openAgentLog() throws NotesException
```

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode wird ein Agentlog geöffnet.

openFileLog

Deklaration

```
void openFileLog(String filepath) throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
filepath	nein	String	Dateiname der Protokolldatei

Beschreibung

Diese Methode öffnet eine Protokolldatei.

openMailLog

Deklaration

void openMailLog(java.util.Vector recipients, String subject) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
recipients	nein	java.util.Vector	Empfänger der Mail
subject	nein	String	Thema der Mail

Beschreibung

Diese Methode öffnet das Maillogging.

openNotesLog

Deklaration

void openNotesLog(String server, String database) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Servername, auf dem sich die Datenbank befindet
database	nein	String	Protokolldatenbank, in der die Protokollierung stattfinden soll.

Beschreibung

Mit dieser Methode wird das Protokoll in einer Datenbank geöffnet.

9.16 Klasse *MIMEEntity*

Verfügbar ab Version 5.0.2

9.16.1 Konstanten

Name	Wert
ENC_BASE64	1727
ENC_EXTENSION	1731
ENC_IDENTITY_7BIT	1728
ENC_IDENTITY_8BIT	1729
ENC_IDENTITY_BINARY	1730
ENC_NONE	1725
ENC_QUOTED_PRINTABLE	1726
SEARCH_BREADTH	1723
SEARCH_DEPTH	1724

9.16.2 Eigenschaften

BoundaryEnd

Datentyp

String

Deklaration Getter

String getBoundaryEnd() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die Abschlusskennzeichnung des MIME-Elements ausgelesen werden.

BoundaryStart

Datentyp

String

Deklaration Getter

String getBoundaryStart() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die Startkennzeichnung des MIME-Elements ausgelesen werden.

Charset

Datentyp

String

Deklaration Getter

String getCharset() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft verrät den Zeichensatz.

ContentAsText

Datentyp

String

Deklaration Getter

String getContentAsText() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann der Inhalt in Textform ausgelesen werden.

ContentSubType

Datentyp

String

Deklaration Getter

String getContentSubType() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft gibt Auskunft über den Inhaltsuntertyp der MIME-Entität.

ContentType

Datentyp

String

Deklaration Getter

String getContentType() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft gibt Auskunft über den Inhaltstyp der MIME-Entität.

Encoding

Datentyp

int

Deklaration Getter

int getEncoding() throws NotesException

Deklaration Setter

kein

Wertebereich

MIMEEntity.ENC_BASE64

MIMEEntity.ENC_EXTENSION

MIMEEntity.ENC_IDENTITY_7BIT

MIMEEntity.ENC_IDENTITY_8BIT

MIMEEntity.ENC_IDENTITY_BINARY

MIMEEntity.ENC_NONE

MIMEEntity.ENC_QUOTED_PRINTABLE

Beschreibung

Beim Auslesen dieser Eigenschaft erhält man das Encoding.

HeaderObjects

Datentyp

java.util.Vector(MIMEHeader)

Deklaration Getter

java.util.Vector getHeaderObjects() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet einen Vektor mit allen Kopfdaten.

Headers

Datentyp

String

Deklaration Getter

String getHeaders() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Beim Auslesen dieser Eigenschaft werden die Kopfdaten in einer Zeichenkette geliefert.

Preamble

Datentyp

String

Deklaration Getter

String getPreamble() throws NotesException

Deklaration Setter

void setPreamble(String preamble) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Präambel.

9.16.3 Methoden**createChildEntity**

Deklaration

MIMEEntity createChildEntity() throws NotesException

MIMEEntity createChildEntity(MIMEEntity nextSibling) throws NotesException

Rückgabewert

MIMEEntity
der neu erzeugte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
nextSibling	ja	MIMEEntity	Der Eintrag, vor dem der neue Eintrag eingefügt werden soll

Beschreibung

Diese Methode erstellt einen neuen MIME-Kindeintrag.

createHeader

Deklaration

MIMEEntity createHeader(String headerName) throws NotesException

Rückgabewert

MIMEEntity
Referenz auf neu erzeugten Headereintrag

Parameter

Name	Optional	Typ	Beschreibung
headerName	nein	String	Der Name des Headers

Beschreibung

Diese Methode legt einen neuen Headereintrag an.

createParentEntry

Deklaration

MIMEEntity createParentEntry() throws NotesException

Rückgabewert

MIMEEntity
Referenz auf das neu erzeugte Element

Parameter

keine

Beschreibung

Diese Methode erzeugt ein übergeordnetes Element.

decodeContent

Deklaration

`void decodeContent()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode dekodiert die MIME-Entität.

encodeContent

Deklaration

`void encodeContent(int encoding)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
encoding	nein	int	Kodierung MIMEEntity.ENC_BASE64 MIMEEntity.ENC_EXTENSION MIMEEntity.ENC_IDENTITY_7BIT MIMEEntity.ENC_IDENTITY_8BIT MIMEEntity.ENC_IDENTITY_BINARY MIMEEntity.ENC_NONE MIMEEntity.ENC_QUOTED_PRINTABLE

Beschreibung

Diese Methode kodiert die MIME-Entität.

getContentAsBytes

Deklaration

`void getContentAsBytes(Stream stream)` throws `NotesException`

`void getContentAsBytes(Stream stream, boolean decoded)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	Stream	Ausgabestream
decoded	ja	boolean	Dekodierung ja/nein

Beschreibung

Diese Methode schreibt den Inhalt der MIME-Elemente in einen Bytestream.

getContentAsText

Deklaration

`void getContentAsText(Stream stream) throws NotesException`

`void getContentAsText(Stream stream, boolean decoded) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	Stream	Ausgabestream
decode	ja	boolean	Dekodierung ja/nein

Beschreibung

Diese Methode schreibt den Inhalt der MIME-Elemente in einen Textstream.

getEntityAsText

Deklaration

`void getEntityAsText(Stream stream) throws NotesException`

`void getEntityAsText(Stream stream, java.util.Vector headerFilters) throws NotesException`

`void getEntityAsText(Stream stream, java.util.Vector headerFilters, boolean inclusive) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	Stream	Ausgabestream
headerFilters	ja	java.util.Vector	Liste von Headernamen
inclusive	ja	boolean	Wenn der Parameter den Wert true hat, werden die Elemente zurückgeliefert, deren Namen im zweiten Parameter aufgeführt sind. Ansonsten werden die Elemente geliefert, die nicht im zweiten Parameter aufgeführt sind.

Beschreibung

Diese Methode schreibt den Inhalt der MIME-Elemente in einen Textstream.

getFirstChildEntity

Deklaration

MIMEEntity getFirstChildEntity() throws NotesException

Rückgabewert

MIMEEntity
die gefundene MIME-Entität

Parameter

keine

Beschreibung

Diese Methode liefert das erste Element einer Multipart-MIME-Entität.

getNextEntity

Deklaration

MIMEEntity getNextEntity() throws NotesException
MIMEEntity getNextEntity(int search) throws NotesException

Rückgabewert

MIMEEntitiy
die gefundene MIME-Entität

Parameter

Name	Optional	Typ	Beschreibung
search	ja	int	Suchablauf MIMEEntity.SEARCH_DEPTH MIMEEntity.SEARCH_BREADTH

Beschreibung

Die Methode liefert den nächsten Teil eines Multipart-Elements.

getNextSibling

Deklaration

MIMEEntity getNextSibling() throws NotesException

Rückgabewert

MIMEEntity

Parameter

keine

Beschreibung

Liefert das folgende Element einer MIME-Multipart-Entität.

getNthHeader

Deklaration

MIMEHeader getNthHeader(String headerName) throws NotesException

MIMEHeader getNthHeader(String headerName, int instance) throws
NotesException

Rückgabewert

MIMEHeader

der gefundene Header in der MIME-Entität

Parameter

Name	Optional	Typ	Beschreibung
headerName	nein	String	Name des Kopfeintrags
instance	ja	int	Nummer des Kopfeintrags

Beschreibung

Diese Methode dient zum Lesen von Einträgen der MIME-Entität.

getParentEntity

Deklaration

MIMEEntity getParentEntity() throws NotesException

Rückgabewert

MIMEEntity

der gefundene Eintrag der MIME-Entität

Parameter

keine

Beschreibung

Diese Methode liefert den übergeordneten Eintrag.

getPrevEntity

Deklaration

MIMEEntity getPrevEntity() throws *NotesException*

MIMEEntity getPrevEntity(int search) throws *NotesException*

Rückgabewert

MIMEEntity

der gefundene Eintrag der MIME-Entität

Parameter

Name	Optional	Typ	Beschreibung
search	ja	int	Suchreihenfolge <i>MIMEEntity</i> .SEARCH_DEPTH <i>MIMEEntity</i> .SEARCH_BREADTH

Beschreibung

Diese Methode liefert den vorhergehenden Eintrag der MIME-Entität.

getPrevSibling

Deklaration

MIMEEntity getPrevSibling() throws *NotesException*

Rückgabewert

MIMEEntity

der gefundene Eintrag der MIME-Entität

Parameter

keine

Beschreibung

Die Methode liefert den vorherigen Eintrag der MIME-Entität.

getSomeHeaders

Deklaration

String getSomeHeaders() throws NotesException

String getSomeHeaders(java.util.Vector headerFilters) throws NotesException

String getSomeHeaders(java.util.Vector headerFilters, boolean inclusive)
throws NotesException

Rückgabewert

String

der Inhalt der Einträge als Zeichenkette

Parameter

Name	Optional	Typ	Beschreibung
headerFilters	ja	java.util.Vector	Namen der gewünschten oder ungewünschten Einträge
inclusive	ja	boolean	Wenn der Parameter den Wert true hat, werden die Elemente zurückgeliefert, deren Namen im zweiten Parameter aufgeführt sind. Ansonsten werden die Elemente geliefert, die nicht im zweiten Parameter aufgeführt sind.

Beschreibung

Diese Methode liefert den Inhalt der durch die Parameter spezifizierten Elemente der MIME-Entität zurück.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode wird das Element gelöscht.

setContentFromBytes

Deklaration

void setContentFromBytes(Stream stream, String contentType, int encoding)
throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	Stream	Eingabestream
contentType	nein	String	Contenttype
encoding	nein	int	Encoding des MIME-Elements MIMEEntity.ENC_BASE64 MIMEEntity.ENC_EXTENSION MIMEEntity.ENC_IDENTITY_7BIT MIMEEntity.ENC_IDENTITY_8BIT MIMEEntity.ENC_IDENTITY_BINARY MIMEEntity.ENC_NONE MIMEEntity.ENC_QUOTED_PRINTABLE

Beschreibung

Diese Methode schreibt die Daten aus dem Bytestream in das MIME-Element.

setContentFromText

Deklaration

```
void setContentFromText(Stream stream, String contentType, int encoding)  
    throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	Stream	Eingabestream
contentType	nein	String	Contenttype
encoding	nein	int	Encoding des MIME Elements MIMEEntity.ENC_BASE64 MIMEEntity.ENC_EXTENSION MIMEEntity.ENC_IDENTITY_7BIT MIMEEntity.ENC_IDENTITY_8BIT MIMEEntity.ENC_IDENTITY_BINARY MIMEEntity.ENC_NONE MIMEEntity.ENC_QUOTED_PRINTABLE

Beschreibung

Diese Methode schreibt die Daten aus dem Textstream in das MIME-Element.

9.17 Klasse MIMEHeader

Verfügbar ab Version 6.0

9.17.1 Eigenschaften

HeaderName

Datentyp

String

Deklaration Getter

String getHeaderName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Namen des Eintrags.

9.17.2 Methoden

addValText

Deklaration

boolean addValText(String valueText) throws NotesException

boolean addValText(String valueText, String charSet) throws NotesException

Rückgabewert

boolean

liefert true, wenn die Methode erfolgreich ausgeführt wurde

Parameter

Name	Optional	Typ	Beschreibung
valueText	nein	String	Anzuhängender Text
charSet	ja	String	Zeichensatz

Beschreibung

Diese Methode hängt den angegebenen Text an den Inhalt des Eintrags.

getHeaderVal

Deklaration

String getHeaderVal() throws NotesException

String getHeaderVal(boolean folded) throws NotesException

String getHeaderVal(boolean folded, boolean decoded) throws NotesException

Rückgabewert

String

der Inhalt des Elements

Parameter

Name	Optional	Typ	Beschreibung
folded	ja	boolean	True bedeutet, der Inhalt ist im multi-line-Format; false bedeutet single-line-Format.
decoded	ja	boolean	Bei true wird nach RFC-2047 dekodiert.

Beschreibung

Die Methode liefert den Inhalt des Elements.

getHeaderValAndParams

Deklaration

String getHeaderValAndParams() throws NotesException

String getHeaderValAndParams(boolean folded) throws NotesException

String getHeaderValAndParams(boolean folded, boolean decoded) throws
NotesException

Rückgabewert

String

der Inhalt des Elements

Parameter

Name	Optional	Typ	Beschreibung
folded	ja	boolean	True bedeutet, der Inhalt ist im multi-line-Format; false bedeutet single-line-Format.
decoded	ja	boolean	Bei true wird nach RFC-2047 dekodiert.

Beschreibung

Die Methode liefert den Inhalt des Elements.

getParamVal

Deklaration

String getParamVal(String paramName) throws NotesException

String getParamVal(String paramName, boolean folded) throws NotesException

Rückgabewert

String

der Inhalt des Elements

Parameter

Name	Optional	Typ	Beschreibung
folded	ja	boolean	True bedeutet, der Inhalt ist im multi-line-Format; false bedeutet single-line-Format.
decoded	ja	boolean	Bei True wird nach RFC-2047 dekodiert.

Beschreibung

Die Methode liefert den Inhalt des Elements.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Das Element wird gelöscht.

setHeaderVal

Deklaration

boolean setHeaderVal(String headerValue) throws NotesException

Rückgabewert

boolean

liefert true, wenn die Methode erfolgreich ausgeführt werden konnte

Parameter

Name	Optional	Typ	Beschreibung
headerValue	nein	String	Zu schreibender Wert

Beschreibung

Die Methode schreibt den angegebenen Wert in das Element.

setHeaderValAndParams

Deklaration

boolean setHeaderValAndParams(String headerParamValue) throws NotesException

Rückgabewert

boolean

liefert true, wenn die Methode erfolgreich ausgeführt werden konnte

Parameter

Name	Optional	Typ	Beschreibung
headerParamValue	nein	String	Zu schreibender Wert

Beschreibung

Die Methode schreibt den angegebenen Wert in das Element.

setParamVal

Deklaration

boolean setParamVal(String parameterName, String parameterValue)
throws NotesException

Rückgabewert

boolean

liefert true, wenn die Methode erfolgreich ausgeführt werden konnte

Parameter

Name	Optional	Typ	Beschreibung
parameterName	nein	String	Parametername
parameterValue	nein	String	Parameterwert

Beschreibung

Die Methode setzt den Wert des Parameters.

9.18 Klasse Name

9.18.1 Eigenschaften

Abbreviated

Datentyp

String

Deklaration Getter

String getAbbreviated() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft erhält man den Namen in abgekürzter Form.

Addr821

Datentyp

String

Deklaration Getter

String getAddr821() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name in Form des RFC-821-Standards.

Addr821Comment1

Datentyp

String

Deklaration Getter

String getAddr822Comment1() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Kommentar zu dem Namen im RFC-821-Format.

Addr821Comment2

Datentyp

String

Deklaration Getter

String getAddr822Comment2() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Kommentar zu dem Namen im RFC-821-Format.

Addr821Comment3

Datentyp

String

Deklaration Getter

String getAddr822Comment3() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Kommentar zu dem Namen im RFC-821-Format.

Addr821LocalPart

Datentyp

String

Deklaration Getter

String getAddr822LocalPart() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der lokale Bestandteil des Namens im RFC-821-Format.

Addr821Phrase

Datentyp

String

Deklaration Getter

String getAddr822Phrase() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Phrase des Namens im RFC-821-Format.

ADMD

Datentyp

String

Deklaration Getter

String getADMD() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die administrative Domäne des Namens.

Canonical

Datentyp

String

Deklaration Getter

String getCanonical() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name in kanonischer Form.

Common

Datentyp

String

Deklaration Getter

String getCommon() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Hier findet man den allgemeinen Namen als Zeichenkette.

Country

Datentyp

String

Deklaration Getter

String getCountry() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft gibt Auskunft über das Land.

Generation

Datentyp

String

Deklaration Getter

String getGeneration() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Eigenschaft beinhaltet die Generationsbezeichnung für den Namenseintrag.

Given

Datentyp

String

Deklaration Getter

String getGiven() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Vornamen für den Namenseintrag.

Initials

Datentyp

String

Deklaration Getter

String getInitials() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Initialen für den Namenseintrag.

IsHierarchical

Datentyp

boolean

Deklaration Getter

boolean isHierarchical() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob der Eintrag in hierarchischer Form vorliegt.

Keyword

Datentyp

String

Deklaration Getter

String getKeyword() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den vollen Namen.

Language

Datentyp

String

Deklaration Getter

String getLanguage() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft kann die Sprache für den angegebenen Namen stehen. Für den primären Namen eines Anwenders entspricht diese Eigenschaft dem Wert null.

Organization

Datentyp

String

Deklaration Getter

String getOrganization() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die Organisation für den Namen ausgelesen werden.

OrgUnit1

Datentyp

String

Deklaration Getter

String getOrgUnit1() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die erste Organisationseinheit für den Namen ausgelesen werden.

OrgUnit2

Datentyp

String

Deklaration Getter

String getOrgUnit2() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die zweite Organisationseinheit für den Namen ausgelesen werden.

OrgUnit3

Datentyp

String

Deklaration Getter

String getOrgUnit3() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die dritte Organisationseinheit für den Namen ausgelesen werden.

OrgUnit4

Datentyp

String

Deklaration Getter

String getOrgUnit4() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die vierte Organisationseinheit für den Namen ausgelesen werden.

Parent

Datentyp

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann eine Referenz auf die übergeordnete Session ausgelesen werden.

PRMD

Datentyp

String

Deklaration Getter

String getPRMD() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die Administrationsdomäne ausgelesen werden.

Surname

Datentyp

String

Deklaration Getter

String getSurname() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft erhält man den Nachnamen.

9.18.2 Methoden

Keine

9.19 Klasse Newsletter

9.19.1 Eigenschaften

isDoScore

Datentyp

boolean

Deklaration Getter

boolean isDoScore() throws NotesException

Deklaration Setter

void setDoScore(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob die Bewertung des Suchkriteriums im Newsletterdokument gespeichert ist.

IsDoSubject

Datentyp

boolean

Deklaration Getter

boolean isDoSubject() throws NotesException

Deklaration Setter

void setDoSubject(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob das Thema der Dokumente im Newsletterdokument gespeichert ist.

Parent

Datentyp

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Durch diese Eigenschaft erhält man eine Referenz auf die übergeordnete Session.

SubjectItemName

Datentyp

String

Deklaration Getter

String getSubjectItemName() throws NotesException

Deklaration Setter

void setSubjectItemName(String name) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Namen des Items in den Dokumenten, aus dem das Thema des Dokuments ermittelt werden soll.

9.19.2 Methoden

formatDocument

Deklaration

Document formatDocument(Database db, int index) throws NotesException

Rückgabewert

Document

die neu erzeugte Kopie des Dokuments

Parameter

Name	Optional	Typ	Beschreibung
db	nein	Database	Datenbank, in der das Dokument erstellt werden soll
index	nein	int	Die Nummer des Dokuments, das als Basis für das neue Dokument dienen soll

Beschreibung

Mit dieser Methode wird basierend auf das Dokument an der angegebenen Position ein neues Dokument angelegt. Dieser Vorgang ist vergleichbar mit einem Weiterleiten des Dokuments.

formatMsgWithDoclinks

Deklaration

Document formatMsgWithDoclinks(Database db) throws NotesException

Rückgabewert

Document

Parameter

Name	Optional	Typ	Beschreibung
db	nein	Database	Datenbank, in der das Dokument erstellt werden soll

Beschreibung

Mit dieser Methode wird ein Übersichtsdokument erstellt mit jeweils einem Eintrag inklusive Verknüpfung zu jedem Dokument im Newsletter.

9.20 Klasse Outline

9.20.1 Eigenschaften

Alias

Datentyp

String

Deklaration Getter

String getAlias() throws NotesException

Deklaration Setter

void setAlias(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Aliasname für die Gliederung.

Comment

Datentyp

String

Deklaration Getter

String getComment() throws NotesException

Deklaration Setter

void setComment(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft kann ein Kommentar zur Gliederung gespeichert werden.

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft gibt den Namen der Gliederung an.

ParentDatabase

Datentyp

Database

Deklaration Getter

Database getParentDatabase() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft wird eine Referenz auf die Datenbank gespeichert.

9.20.2 Methoden

addEntry

Deklaration

void addEntry(OutlineEntry entry, OutlineEntry referenceEntry)
throws NotesException

void addEntry(OutlineEntry entry, OutlineEntry referenceEntry, boolean after)
throws NotesException

void addEntry(OutlineEntry entry, OutlineEntry referenceEntry, boolean after,
boolean asChild) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der einzufügende Eintrag
referenceEntry	nein	OutlineEntry	Eine Referenz auf einen Eintrag, vor dem oder unter dem ein neuer Eintrag eingefügt werden soll.
after	ja	boolean	Dieser Parameter gibt Auskunft, ob der neue Eintrag hinter dem als referenceEntry übergebenen Eintrag eingefügt werden soll.
asChild	ja	boolean	Dieser Parameter gibt Auskunft, ob der neue Eintrag unter dem als referenceEntry übergebenen Eintrag eingefügt werden soll.

Beschreibung

Mit dieser Methode können neue Einträge in einer Gliederung eingefügt werden. Im Parameter referenceEntry kann ein Eintrag übergeben werden, um die Position für den neuen Eintrag zu bestimmen.

createEntry

Deklaration

OutlineEntry createEntry (OutlineEntry fromEntry) throws NotesException

OutlineEntry createEntry (OutlineEntry fromEntry, OutlineEntry referenceEntry) throws NotesException

OutlineEntry createEntry (OutlineEntry fromEntry, OutlineEntry referenceEntry, boolean after) throws NotesException

OutlineEntry createEntry (OutlineEntry fromEntry, OutlineEntry referenceEntry, boolean after, boolean asChild) throws NotesException

OutlineEntry createEntry (String entryName) throws NotesException

OutlineEntry createEntry (String entryName, OutlineEntry referenceEntry) throws NotesException

OutlineEntry createEntry (String entryName, OutlineEntry referenceEntry, boolean after) throws NotesException

OutlineEntry createEntry (String entryName, OutlineEntry referenceEntry, boolean after, boolean asChild) throws NotesException

Rückgabewert

OutlineEntry

der neue Gliederungseintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	OutlineEntry	Eintrag, der kopiert werden soll
referenceEntry	nein	OutlineEntry	Eine Referenz auf einen Eintrag, vor dem oder unter dem ein Eintrag eingefügt werden soll
after	ja	boolean	Dieser Parameter gibt Auskunft, ob der neue Eintrag hinter dem als referenceEntry übergebenen Eintrag eingefügt werden soll.
asChild	ja	boolean	Dieser Parameter gibt Auskunft, ob der neue Eintrag unter dem als referenceEntry übergebenen Eintrag eingefügt werden soll.
entryName	ja	String	Name des neuen Eintrags

Beschreibung

Diese Methode erstellt einen Gliederungseintrag und fügt diesen dann in die bestehende Gliederung ein.

getChild

Deklaration

OutlineEntry getChild(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry
der untergeordnete Gliederungseintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Eintrag, dessen untergeordnetes Element ermittelt werden soll

Beschreibung

Diese Methode liefert den untergeordneten Gliederungseintrag.

getFirst

Deklaration

OutlineEntry getFirst() throws NotesException

Rückgabewert

OutlineEntry
liefert den ersten Gliederungseintrag

Parameter

keine

Beschreibung

Diese Methode liefert den ersten Gliederungseintrag.

getLast

Deklaration

OutlineEntry getLast() throws NotesException

Rückgabewert

OutlineEntry
der letzte Eintrag in der Gliederung

Parameter

keine

Beschreibung

Diese Methode liefert den letzten Gliederungseintrag.

getNext

Deklaration

OutlineEntry getNext(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry
der Folgeeintrag des angegebenen Eintrags

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, dessen folgender Eintrag ermittelt werden soll

Beschreibung

Diese Methode liefert den nächsten Eintrag aus der Gliederung.

getNextSibling

Deklaration

OutlineEntry getNextSibling(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry

wenn vorhanden, der gefundene Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, dessen Nachfolger in derselben Ebene ermittelt werden soll

Beschreibung

Diese Methode liefert den Eintrag, der dem angegebenen Eintrag auf derselben Ebene folgt.

getParent

Deklaration

OutlineEntry getParent(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry

der übergeordnete Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, dessen übergeordneter Eintrag gefunden werden soll

Beschreibung

Liefert den übergeordneten Eintrag des angegebenen Gliederungseintrags.

getPrev

Deklaration

OutlineEntry getPrev(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry

Der Vorgängereintrag des angegebenen Eintrags

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, dessen Vorgänger ermittelt werden soll

Beschreibung

Diese Methode liefert den Vorgängereintrag des angegebenen Eintrags.

getPrevSibling

Deklaration

OutlineEntry getPrevSibling(OutlineEntry entry) throws NotesException

Rückgabewert

OutlineEntry

wenn vorhanden, der Vorgängereintrag des angegebenen Eintrags

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, dessen Vorgängereintrag ermittelt werden soll

Beschreibung

Diese Methode liefert den Vorgängereintrag des angegebenen Eintrags auf derselben Gliederungsebene.

moveEntry

Deklaration

void moveEntry(OutlineEntry entry, OutlineEntry referenceEntry) throws NotesException

void moveEntry(OutlineEntry entry, OutlineEntry referenceEntry, boolean after) throws NotesException

void moveEntry(OutlineEntry entry, OutlineEntry referenceEntry, boolean after, boolean asChild) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Der Eintrag, der verschoben werden soll
referenceEntry	nein	OutlineEntry	Der Eintrag, der als Orientierung für die neue Position genutzt wird
after	ja	boolean	Bei true wird der Eintrag hinter dem Referenzeintrag eingefügt.
asChild	ja	boolean	Bei true wird der Eintrag unter den Referenzeintrag eingefügt.

Beschreibung

Diese Methode dient zum Verschieben eines Gliederungseintrags.

removeEntry

Deklaration

void removeEntry(OutlineEntry entry) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	OutlineEntry	Eintrag, der gelöscht werden soll

Beschreibung

Diese Methode löscht den angegebenen Eintrag.

save

Deklaration

void save()

Rückgabewert

kein

Parameter

keine

Beschreibung

Durch Aufruf dieser Methode wird die Gliederung gespeichert.

9.21 Klasse OutlineEntry

9.21.1 Konstanten

Name	Wert
OUTLINE_CLASS_DATABASE	2194
OUTLINE_CLASS_DOCUMENT	2190
OUTLINE_CLASS_FOLDER	2197
OUTLINE_CLASS_FORM	2192
OUTLINE_CLASS_FRAMESET	2195
OUTLINE_CLASS_NAVIGATOR	2193
OUTLINE_CLASS_PAGE	2196
OUTLINE_CLASS_UNKNOWN	2189
OUTLINE_CLASS_VIEW	2191
OUTLINE_OTHER_FOLDERS_TYPE	1589
OUTLINE_OTHER_UNKNOWN_TYPE	1591
OUTLINE_OTHER_VIEWS_TYPE	1588
OUTLINE_TYPE_ACTION	2188
OUTLINE_TYPE_NAMEDELEMENT	2187
OUTLINE_TYPE_NOTELINK	2186
OUTLINE_TYPE_URL	2185

9.21.2 Eigenschaften

Alias

Datentyp

String

Deklaration Getter

String getAlias() throws NotesException

Deklaration Setter

void setAlias(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Aliasname für den Eintrag.

Database

Datentyp

Database

Deklaration Getter

Database getDatabase() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft gelangt man zu der Datenbank, in der sich der Gliederungseintrag befindet.

Document

Datentyp

Document

Deklaration Getter

Document getDocument() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Referenz auf das Dokument, in dem Einstellungsdaten für den Eintrag gespeichert sind.

EntryClass

Datentyp

int

Deklaration Getter

int getEntryClass() throws NotesException

Deklaration Setter

kein

Wertebereich

Wert	Beschreibung
OutlineEntry.OUTLINE_CLASS_DATABASE	Datenbankverknüpfung
OutlineEntry.OUTLINE_CLASS_DOCUMENT	Dokumentenverknüpfung
OutlineEntry.OUTLINE_CLASS_FORM	Maskenverknüpfung
OutlineEntry.OUTLINE_CLASS_FOLDER	Ordnerverknüpfung
OutlineEntry.OUTLINE_CLASS_FRAMESET	Rahmengruppenverknüpfung
OutlineEntry.OUTLINE_CLASS_NAVIGATOR	Navigatorverknüpfung
OutlineEntry.OUTLINE_CLASS_PAGE	Seitenverknüpfung
OutlineEntry.OUTLINE_CLASS_UNKNOWN	Unbekannt
OutlineEntry.OUTLINE_CLASS_VIEW	Ansichtsverknüpfung

Beschreibung

In dieser Eigenschaft befindet sich ein numerischer Wert, der Auskunft über die Klasse des Gliderungseintrags gibt.

Formula

Datentyp

String

Deklaration Getter

String getFormula() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn der Gliderungseintrag eine Formularaktion ist, beinhaltet diese Eigenschaft den Formulartext.

FrameText

Datentyp

String

Deklaration Getter

String getFrameText() throws NotesException

Deklaration Setter

void setFrameText(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft steht der Name der Rahmengruppe, in der die Aktion des Gliederungseintrags ausgeführt wird.

HasChildren

Datentyp

boolean

Deklaration Getter

boolean hasChildren() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft liefert den Wert *true*, wenn der Gliederungseintrag untergeordnete Elemente besitzt.

HideFormula

Datentyp

String

Deklaration Getter

String getHideFormula() throws NotesException

Deklaration Setter

void setHideFormula(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Formel, die bestimmt, wann der Eintrag angezeigt und wann versteckt wird.

ImagesText

Datentyp

String

Deklaration Getter

String getImagesText() throws NotesException

Deklaration Setter

void setImagesText(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Namen der Icondatei.

IsHidden

Datentyp

boolean

Deklaration Getter

boolean isHidden() throws NotesException

Deklaration Setter

void setHidden(boolean value) throws NotesException

Wertebereich

true, false

Beschreibung

Durch den Inhalt dieser Eigenschaft kann bestimmt werden, ob der Eintrag angezeigt wird oder nicht.

IsHiddenFromNotes

Datentyp

boolean

Deklaration Getter

boolean isHiddenFromNotes() throws NotesException

Deklaration Setter

void setHiddenFromNotes(boolean value) throws NotesException

Wertebereich

true, false

Beschreibung

Durch den Inhalt dieser Eigenschaft wird bestimmt, ob der Eintrag in einem Notes Client angezeigt wird oder nicht.

IsHiddenFromWeb

Datentyp

boolean

Deklaration Getter

boolean isHiddenFromWeb() throws NotesException

Deklaration Setter

void setHiddenFromWeb(boolean value) throws NotesException

Wertebereich

true, false

Beschreibung

Durch den Inhalt dieser Eigenschaft wird bestimmt, ob der Eintrag im Webbrowser angezeigt wird oder nicht.

IsInThisDB

Datentyp

boolean

Deklaration Getter

boolean isInThisDB() throws NotesException

Deklaration Setter

keinen

Wertebereich

true, false

Beschreibung

Diese Eigenschaft beinhaltet den Wert *true*, wenn das Element, auf das dieser Gliederungseintrag verweist, sich in dieser Datenbank befindet.

IsPrivate

Datentyp

boolean

Deklaration Getter

boolean isPrivate() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Durch den Wert dieser Eigenschaft wird bestimmt, ob es sich bei dem Gliederungseintrag um einen privaten oder öffentlichen Eintrag handelt.

KeepSelectionFocus

Datentyp

boolean

Deklaration Getter

boolean getKeepSelectionFocus() throws NotesException

Deklaration Setter

void setKeepSelectionFocus(boolean value) throws NotesException

Wertebereich

true, false

Beschreibung

Diese Eigenschaft beinhaltet den Wert *true*, wenn der Eintrag den Focus behält.

Label**Datentyp**

String

Deklaration Getter

String getLabel() throws NotesException

Deklaration Setter

void setLabel(String value) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Die Zeichenkette in dieser Eigenschaft entspricht dem Text, der im Gliederungseintrag angezeigt wird.

Level**Datentyp**

int

Deklaration Getter

int getLevel() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der numerische Wert in dieser Eigenschaft bestimmt die Ebene des Gliederungseintrags.

NamedElement**Datentyp**

String

Deklaration Getter

String `getNamedElement()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn der Gliderungseintrag auf ein benanntes Element verweist, beinhaltet diese Eigenschaft den Namen des Elements.

Parent

Datentyp

Outline

Deklaration Getter

Outline `getParent()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Referenz auf die Gliederung, zu der der Eintrag gehört.

Type

Datentyp

int

Deklaration Getter

int `getType()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

Konstante	Beschreibung
OutlineEntry.OUTLINE_OTHER_FOLDERS_TYPE	Ordner
OutlineEntry.OUTLINE_OTHER_UNKNOWN_TYPE	Unbestimmt
OutlineEntry.OUTLINE_OTHER_VIEWS_TYPE	Ansicht
OutlineEntry.OUTLINE_TYPE_ACTION	Aktion
OutlineEntry.OUTLINE_TYPE_NAMEDELEMENT	Benanntes Element
OutlineEntry.OUTLINE_TYPE_NOTELINK	Notes Verknüpfung
OutlineEntry.OUTLINE_TYPE_URL	URL

URL

Datentyp

String

Deklaration Getter

String getURL() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn es sich bei dem Gliederungseintrag um eine URL handelt, liefert die Eigenschaft diese URL zurück.

UseHideFormula

Datentyp

boolean

Deklaration Getter

boolean getUseHideFormula() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Über den Inhalt dieser Eigenschaft kann ermittelt werden, ob die Hidden-Formel berücksichtigt wird. Die Hidden-Formel bestimmt, ob der Eintrag angezeigt wird.

View

Datentyp

View

Deklaration Getter

View getView() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn der Eintrag auf eine Ansicht verweist, beinhaltet diese Eigenschaft den Namen der Ansicht.

9.21.3 Methoden

setAction

Deklaration

boolean setAction(String formula) throws NotesException

Rückgabewert

boolean

Der Rückgabewert liefert true, wenn die Aktion erfolgreich gesetzt werden konnte.

Parameter

Name	Optional	Typ	Beschreibung
formula	nein	String	Formel mit der Aktion für den Eintrag

Beschreibung

Diese Methode setzt die angegebene Formel als Aktion für den Eintrag.

setNamedElement

Deklaration

boolean setNamedElement(Database db, String elementName, int entryClass)
throws NotesException

Rückgabewert

boolean

Die Methode liefert den Wert true, wenn der Verweis auf das benannte Element gesetzt werden konnte.

Parameter

Name	Optional	Typ	Beschreibung
db	nein	Database	Die Datenbank, die das benannte Element beinhaltet
elementName	nein	String	Name des benannten Elements
entryClass	nein	int	Klasse des benannten Elements

Beschreibung

Die Methode dient zum Setzen eines Verweises auf ein benanntes Element.

setNoteLink

Deklaration

boolean setNoteLink(Database db) throws NotesException
boolean setNoteLink(Document doc) throws NotesException
boolean setNoteLink(View view) throws NotesException

Rückgabewert

boolean

Der Wert true wird zurückgeliefert, wenn das Setzen der Verknüpfung erfolgreich war.

Parameter

Name	Optional	Typ	Beschreibung
db	nein	Database	Die Datenbank, auf die verwiesen werden soll
doc	nein	Document	Das Dokument, auf das verwiesen werden soll
view	nein	View	Die Ansicht, auf die verwiesen werden soll

Beschreibung

Diese Methode dient zum Setzen einer Verknüpfung. Mithilfe dieser Methode kann eine Verknüpfung zu einer Datenbank, einem Dokument oder einer Ansicht gesetzt werden.

setURL

Deklaration

`boolean setURL(String url) throws NotesException`

Rückgabewert

`boolean`

Die Methode liefert `true` zurück, wenn das Setzen der URL erfolgreich war.

Parameter

Name	Optional	Typ	Beschreibung
<code>url</code>	nein	<code>String</code>	Die URL, die gesetzt werden soll

Beschreibung

Die Methode dient zum Setzen einer URL, auf die ein Gliderungseintrag verweisen soll.

9.22 Klasse Registration

9.22.1 Konstanten

Name	Wert
<code>ID_CERTIFIER</code>	173
<code>ID_FLAT</code>	171
<code>ID_HIERARCHICAL</code>	172

9.22.2 Eigenschaften

AltOrgUnit

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getAltOrgUnit() throws NotesException`

Deklaration Setter

`void setAltOrgUnit(java.util.Vector names) throws NotesException`

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft wird der alternative Name der Organisation gespeichert, der genutzt wird, wenn neue Registrationen stattfinden.

AltOrgUnitLang

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getAltOrgUnitLang() throws NotesException`

Deklaration Setter

`void setAltOrgUnitLang(java.util.Vector languages) throws NotesException`

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft steht der alternative Name der Organisation, der bei der nächsten Registrierung verwendet wird.

CerifierIDFile

Datentyp

`String`

Deklaration Getter

`String getCertifierIDFile() throws NotesException`

Deklaration Setter

`void setCertifierIDFile(String file) throws NotesException`

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name der Zertifizierungs-ID-Datei.

CreateMailDb

Datentyp

`boolean`

Deklaration Getter

boolean getCreateMailDb() throws NotesException

Deklaration Setter

void setCreateMailDb(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird festgelegt, ob bei der nächsten Registrierung eine Maildatenbank für den Anwender angelegt werden soll.

Expiration**Datentyp**

DateTime

Deklaration Getter

DateTime getExpiration() throws NotesException

Deklaration Setter

void setExpiration(DateTime date) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Das Datum in dieser Eigenschaft gibt das Ablaufdatum für die nächste Zertifizierung an.

IDType**Datentyp**

int

Deklaration Getter

int getIDType() throws NotesException

Deklaration Setter

void setIDType(int type) throws NotesException

Wertebereich

Registration.ID_FLAT
Registration.ID_HIERARCHICAL
Registration.ID_CERTIFIER

Beschreibung

Der Inhalt dieser Eigenschaft bestimmt den Typ der ID-Datei, die erzeugt wird.

IsNorthAmerican**Datentyp**

boolean

Deklaration Getter

boolean isNorthAmerican() throws NotesException

Deklaration Setter

void setNorthAmerican(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn eine ID für die Nutzung in Nordamerika erzeugt werden soll, muss diese Eigenschaft den Wert *true* haben.

MinPasswordLength**Datentyp**

int

Deklaration Getter

int getMinPasswordLength() throws NotesException

Deklaration Setter

void setMinPasswordLength(int len) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Mindestlänge für ein Passwort.

OrgUnit

Datentyp

String

Deklaration Getter

String getOrgUnit() throws NotesException

Deklaration Setter

void setOrgUnit(String unit) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Die Organisationseinheit, die bei der nächsten Registrierung verwendet werden soll.

RegistrationLog

Datentyp

String

Deklaration Getter

String getRegistrationLog() throws NotesException

Deklaration Setter

void setRegistrationLog(String name) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Mit dieser Eigenschaft kann der Name der Protokolldatei angegeben werden, die bei der Registrierung genutzt werden soll.

RegistrationServer

Datentyp

String

Deklaration Getter

String getRegistrationServer() throws NotesException

Deklaration Setter

void setRegistrationServer(String server) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Der Name des Servers, der für die Registrierung verwendet werden soll, steht in dieser Eigenschaft.

StoreIDInAddressBook

Datentyp

boolean

Deklaration Getter

boolean getStoreIDInAddressBook() throws NotesException

Deklaration Setter

void setStoreIDInAddressBook(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dem Inhalt dieser Eigenschaft kann bestimmt werden, ob die ID bei der Registrierung im Adressbuch gespeichert werden soll.

UpdateAddressBook

Datentyp

boolean

Deklaration Getter

boolean getUpdateAddressBook() throws NotesException

Deklaration Setter

void setUpdateAddressBook(boolean flag) throws NotesException

Wertebereich

true, false

Beschreibung

Der Inhalt dieser Eigenschaft bestimmt, ob bei der Registrierung ein Dokument für die neue ID im Domino Adressbuch angelegt werden soll.

9.22.3 Methoden

addCertifierToAddressBook

Deklaration

```
boolean addCertifierToAddressBook(String idfile) throws NotesException  
boolean addCertifierToAddressBook(String idfile, String password, String location,  
    String comment) throws NotesException
```

Rückgabewert

boolean

Es wird true zurückgegeben, wenn die Methode erfolgreich ausgeführt wurde, ansonsten false.

Parameter

Name	Optional	Typ	Beschreibung
idfile	Nein	String	Die Zertifizierungs-ID, die hinzugefügt werden soll
password	Ja	String	Passwort für die Zertifizierungs-ID
location	ja	String	Inhalt für das Feld Standort im neuen Dokument im Domino Adressbuch
comment	ja	String	Inhalt für das Feld Kommentar im neuen Dokument im Domino Adressbuch

Beschreibung

Diese Methode dient zum Eintragen einer neuen Zulassungs-ID im Domino Adressbuch.

addServerToAddressBook

Deklaration

```
boolean addServerToAddressBook(String idfile, String server, String domain)  
    throws NotesException  
boolean addServerToAddressBook(String idfile, String server, String domain,  
    String userpw, String network, String adminname, String title, String location,  
    String comment) throws NotesException
```

Rückgabewert

boolean

Es wird true zurückgegeben, wenn die Methode erfolgreich ausgeführt wurde. Ansonsten bekommt man den Wert false zurück.

Parameter

Name	Optional	Typ	Beschreibung
idfile	nein	String	Der Name der Server-ID-Datei, welche hinzugefügt werden soll
server	nein	String	Der Name des hinzuzufügenden Servers
domain	nein	String	Der Name der Domäne für den hinzuzufügenden Server
userpw	ja	String	Passwort für die ID-Datei des hinzuzufügenden Servers
network	ja	String	Name des Netzwerks für den hinzuzufügenden Server
adminname	ja	String	Name des Administrators des hinzuzufügenden Servers
title	ja	String	Servertitel des hinzuzufügenden Servers
location	ja	String	Standort des hinzuzufügenden Servers
comment	ja	String	Kommentar des hinzuzufügenden Servers

Beschreibung

Diese Methode kann dazu genutzt werden, einen Server im Domino Adressbuch hinzuzufügen.

addUserProfile

Deklaration

`void addUserProfile(String username, String profile) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
username	nein	String	Name der Person
profile	nein	String	Name des Profils

Beschreibung

Diese Methode fügt den Namen des Profils in der Personendokument der angegebenen Person ein.

addUserToAddressBook

Deklaration

`boolean addUserToAddressBook(String idfile, String fullname, String lastn)
throws NotesException`

`boolean addUserToAddressBook(String idfile, String fullname, String lastn,
String userpw, String firstn, String middle, String mailserv, String mailfilepath,
String fwdaddr, String location, String comment) throws NotesException`

Rückgabewert

boolean

Die Methode liefert den Wert true zurück, wenn sie erfolgreich ausgeführt wurde.

Parameter

Name	Optional	Typ	Beschreibung
idfile	nein	String	Name der ID-Datei, die eingefügt werden soll
fullname	nein	String	Name der Person
Lastn	nein	String	Nachname der Person
userpw	ja	String	Passwort der Person
firstn	ja	String	Vorname der Person
middle	ja	String	Namenszusatz der Person
mailserv	ja	String	Mailserver der Person
mailfilepath	ja	String	Pfad der Maildatei für die Person
fwdaddr	ja	String	Forwardingadresse der Person
location	ja	String	Standort der Person
comment	ja	String	Kommentar zu der Person

Beschreibung

Diese Methode dient zum Eintragen von Personen in das Domino Adressbuch.

crossCertify

Deklaration

boolean crossCertify(String idfile) throws NotesException

boolean crossCertify(String idfile, String certpw, String comment) throws NotesException

Rückgabewert

boolean

Die Methode liefert den Wert true zurück, wenn sie erfolgreich ausgeführt werden konnte.

Parameter

Name	Optional	Typ	Beschreibung
idfile	nein	String	Name der Datei, die querzugelassen werden soll
certpw	Ja	String	Passwort der zuzulassenden ID-Datei
comment	ja	String	Kommentar für das Dokument im Domino Verzeichnis

Beschreibung

Diese Methode veranlasst eine Querkzulassung der angegebenen Zulassungs-ID.

deleteIDOnServer

Deklaration

`void deleteIDOnServer(String username, boolean isserverid) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
username	nein	String	Name der Person
isserverid	nein	boolean	Dieser Parameter hat für Server-IDs den Wert true, sonst false.

Beschreibung

Mit dieser Methode wird veranlasst, dass die ID-Datei der Person oder des Servers aus dem Domino Verzeichniss entfernt wird.

getIDFromServer

Deklaration

`void getIDFromServer(String username, String filepath, boolean isserverid) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
username	nein	String	Name der Person
filepath	nein	String	Name der Datei, in der die ID-Datei gespeichert werden soll
isserverid	nein	boolean	Gibt Auskunft darüber, ob es sich um eine Server-ID handelt

Beschreibung

Mit dieser Methode kann eine ID-Datei aus dem Personendokument auf dem Server ausgelesen und in eine Datei kopiert werden.

getUserInfo

Deklaration

void getUserInfo(String username, StringBuffer mailserver, StringBuffer mailfile, StringBuffer maildomain, StringBuffer mailsystem, Vector profile) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
username	nein	String	Name der Person
mailserver	nein	StringBuffer	Liefert den Namen des Mailservers zurück
mailfile	nein	StringBuffer	Liefert den Namen der Maildatei zurück
maildomain	nein	StringBuffer	Liefert den Namen der Maildomäne zurück
mailsystem	nein	StringBuffer	Liefert die Art des Mailsystems zurück
profile	nein	Vector	Liefert eine Liste der Profile zurück

Beschreibung

Mit dieser Methode können verschiedene Information über die Person aus dem Domino Adressbuch ermittelt werden.

recertify

Deklaration

boolean recertify(String idfile) throws NotesException
boolean recertify(String idfile, String certpw, String comment) throws NotesException

Rückgabewert

boolean

Die Methode liefert den Wert true zurück, wenn sie erfolgreich ausgeführt werden konnte.

Parameter

Name	Optional	Typ	Beschreibung
idfile	nein	String	Name der ID-Datei, die rezertifiziert werden soll
certpw	ja	String	Passwort der ID-Datei
comment	ja	String	Kommentar für die Rezertifizierung

Beschreibung

Mit dieser Methode können Rezertifizierungen vorgenommen werden.

registerNewCertifier

Deklaration

`boolean registerNewCertifier(String org, String idfile, String certpw) throws NotesException`

`boolean registerNewCertifier(String org, String idfile, String certpw, String country) throws NotesException`

Rückgabewert

boolean

Parameter

Name	Optional	Typ	Beschreibung
org	nein	String	Name der Organisationseinheit, zu der die neue Zulassungs-ID gehören soll
idfile	nein	String	Name der ID-Datei, die erstellt werden soll
certpw	nein	String	Passwort für die neu zu erstellende Datei
country	ja	String	Das Land für die neue Zulassung

Beschreibung

Mit dieser Methode wird eine neue Zulassungs-ID erstellt.

registerNewServer

Deklaration

`boolean registerNewServer(String server, String idfile, String domain, String serverpw) throws NotesException`

`boolean registerNewServer(String server, String idfile, String domain, String serverpw, String certpw, String location, String comment, String network, String adminname, String title) throws NotesException`

Rückgabewert

boolean

Diese Methode liefert den Wert `true` zurück, wenn die Ausführung erfolgreich war.

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des neuen Servers
idfile	nein	String	Dateiname der zu erzeugenden ID-Datei für den Server
domain	nein	String	Domäne des neuen Servers
serverpw	nein	String	Gewünschtes Passwort für die Server-ID-Datei
certpw	ja	String	Passwort für die Zulassungs-ID
location	ja	String	Standort des neuen Servers
comment	ja	String	Kommentar für den neuen Server
network	ja	String	Netzwerkname für den neuen Server
adminame	ja	String	Name des Administrators für den neuen Server
title	ja	String	Titel des neuen Servers

Beschreibung

Mit dieser Methode kann ein neuer Server angelegt und registriert werden.

registerNewUser

Deklaration

```
boolean registerNewUser(String lastname, String idfile, String server) throws
    NotesException
boolean registerNewUser(String lastname, String idfile, String server, String firstn,
    String middle, String certpw, String location, String comment, String maildbpath,
    String forward, String userpw) throws NotesException
boolean registerNewUser(String lastname, String idfile, String server, String firstn,
    String middle, String certpw, String location, String comment, String maildbpath,
    String forward, String userpw, String altname, String altnamelang) throws
    NotesException
```

Rückgabewert

boolean

Diese Methode liefert den Wert true zurück, wenn die Ausführung erfolgreich war.

Parameter

Name	Optional	Typ	Beschreibung
lastname	nein	String	Nachname der neuen Person
idfile	nein	String	Name der neuen ID-Datei
server	nein	String	Name des Heimservers der neuen Person
firstn	ja	String	Vorname der neuen Person
middle	ja	String	Namenszusatz der neuen Person
certpw	ja	String	Passwort für die Zulassungs-ID

Name	Optional	Typ	Beschreibung
location	ja	String	Standort der neuen Person
comment	ja	String	Kommentar zu der neuen Person
maildbpath	ja	String	Pfad und Name der Maildatei der neuen Person
forward	ja	String	Die Mailforwardingadresse der neuen Person
userpw	ja	String	Passwort für die neue Person
altname	ja	String	Alternativer Name für die Person
altnamelang	ja	String	Alternative Sprache für die Person

Beschreibung

Mit dieser Methode wird eine neue Person registriert und zugelassen.

switchToID

Deklaration

String switchToID(String idfile, String userpw) throws NotesException

Rückgabewert

String

der Name des Besitzers der ID-Datei, zu der umgeschaltet wurde

Parameter

Name	Optional	Typ	Beschreibung
idfile	nein	String	Dateiname der ID-Datei
userpw	nein	String	Passwort der ID-Datei

Beschreibung

Mit dieser Methode kann zu einer anderen ID-Datei umgeschaltet werden.

9.23 Klasse Replication

9.23.1 Konstanten

Name	Wert
CNOTES_REPLCONST_PRIORITYHIGH	1549
CNOTES_REPLCONST_PRIORITYLOW	1547
CNOTES_REPLCONST_PRIORITYMED	1548
CNOTES_REPLCONST_PRIORITYNOTSET	1565
LEVEL_NOACCESS	0

9.23.2 Eigenschaften

CutoffDate

Datentyp

DateTime

Deklaration Getter

DateTime getCutoffDate() throws NotesException

Deklaration Setter

kein

Wertebereich

DateTime

Beschreibung

Diese Eigenschaft beinhaltet das Cutoff-Datum für die Replikation.

CutoffInterval

Datentyp

long

Deklaration Getter

long getCutoffInterval() throws NotesException

Deklaration Setter

void setCutoffInterval(long interval) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Anzahl der Tage, nach dem Dokumente automatisch gelöscht werden.

IsAbstract

Datentyp

boolean

Deklaration Getter

boolean isAbstract() throws NotesException

Deklaration Setter

void setAbstract(boolean bValue) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft gesetzt ist, werden längere Dokumente abgeschnitten und ggf. Anhänge entfernt.

IsCutoffDelete**Datentyp**

boolean

Deklaration Getter

boolean isCutoffDelete() throws NotesException

Deklaration Setter

void setCutoffDelete(boolean bValue) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* besitzt, werden Dokumente, die älter sind als das Cutoff-Intervall, entfernt. Bei dem Wert *false* in dieser Eigenschaft erfolgt keine automatische Löschung.

IsDisabled**Datentyp**

boolean

Deklaration Getter

boolean isDisabled() throws NotesException

Deklaration Setter

void setDisabled(boolean bValue) throws NotesException

Wertebereich

true, false

Beschreibung

Mit dieser Eigenschaft kann die gesamte Replikation ausgeschaltet werden.

IsIgnoreDeletes

Datentyp

boolean

Deklaration Getter

boolean isIgnoreDeletes() throws NotesException

Deklaration Setter

void setIgnoreDeletes(boolean bValue) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *false* hat, werden Dokumentenlöschungen nicht an andere Datenbanken übertragen.

IsIgnoreDestDeletes

Datentyp

boolean

Deklaration Getter

boolean isIgnoreDestDeletes() throws NotesException

Deklaration Setter

void setIgnoreDestDeletes(boolean bValue) throws NotesException

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *false* hat, werden Dokumentenlöschungen aus anderen Datenbanken nicht in die eigene übertragen.

Priority

Datentyp

int

Deklaration Getter

int getPriority() throws NotesException

Deklaration Setter

void setPriority(int priority) throws NotesException

Wertebereich

Replication.CNOTES_REPLCONST_PRIORITYHIGH

Replication.CNOTES_REPLCONST_PRIORITYLOW

Replication.CNOTES_REPLCONST_PRIORITYMED

Replication.CNOTES_REPLCONST_PRIORITYNOTSET

Beschreibung

Diese Eigenschaft beinhaltet die Priorität der Replikation.

9.23.3 Methoden**clearHistory**

Deklaration

int clearHistory() throws NotesException

Rückgabewert

int

Parameter

keine

Beschreibung

Diese Methode sorgt für das Löschen der Replikationshistorie. Nach dem Aufruf dieser Methode muss auch die Methode save zum Festschreiben der Änderung aufgerufen werden.

reset

Deklaration

int reset() throws NotesException

Rückgabewert

int

Parameter

keine

Beschreibung

Diese Methode setzt die Einstellungen für die Replikation auf die zuletzt gespeicherten Daten zurück.

save

Deklaration

int save() throws NotesException

Rückgabewert

int

Parameter

keine

Beschreibung

Durch den Aufruf dieser Methode werden Änderungen in den Replikationseinstellungen gespeichert.

9.24 Klasse RichTextItem

9.24.1 Eigenschaften

EmbeddedObjects

Datentyp

java.util.Vector(EmbeddedObject)

Deklaration Getter

java.util.Vector getEmbeddedObjects() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Über diese Eigenschaft erhält man Zugriff auf die eingebetteten Objekte in diesem RichText-Item.

9.24.2 Methoden

addNewLine

Deklaration

void addNewLine() throws NotesException

void addNewLine(int count) throws NotesException

void addNewLine(int count, boolean newparagraph) throws NotesException

Rückgabewert

kein

Parameter

int count (optional)

boolean newparagraph (optional)

Beschreibung

Durch Aufruf dieser Methode wird ein Zeilenwechsel und ggf. ein Abschnittswechsel in dem RichText-Item eingefügt.

addPageBreak

Deklaration

void addPageBreak() throws NotesException

void addPageBreak(RichTextParagraphStyle pstyle) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
pstyle	ja	RichTextParagraphStyle	Format für den neuen Abschnitt

Beschreibung

Durch diese Methode wird ein Seitenwechsel im RichText-Feld angezeigt. Für die neue Seite kann eine neue Formatierung angegeben werden.

addTab

Deklaration

void addTab() throws NotesException

void addTab(int count) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
count	ja	Int	Anzahl der Tabstops, die eingefügt werden sollen

Beschreibung

Durch Aufruf dieser Methode werden ein oder mehrere Tabstops in das RichText-Item eingefügt.

appendDocLink

Deklaration

void appendDocLink(Document target) throws NotesException

void appendDocLink(Database target) throws NotesException

void appendDocLink(View target) throws NotesException

void appendDocLink(Document target, String comment) throws NotesException

void appendDocLink(Database target, String comment) throws NotesException

void appendDocLink(View target, String comment) throws NotesException

void appendDocLink(Document target, String comment, String hotspottext)
throws NotesException

void appendDocLink(Database target, String comment, String hotspottext)
throws NotesException

void appendDocLink(View target, String comment, String hotspottext)
throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
target	nein	Document Database View	Ziel der Verknüpfung
comment	ja	String	Kommentar für die Verknüpfung
hotspot	ja	String	Anzeigetext für die Verknüpfung

Beschreibung

Diese Methode fügt in das RichText-Feld eine Verknüpfung zu einem Dokument, einer Datenbank oder einer Ansicht ein.

appendParagraphStyle

Deklaration

`void appendParagraphStyle(RichTextParagraphStyle pstyle) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
pstyle	ja	RichTextParagraphStyle	Neues Absatzformat

Beschreibung

Diese Methode veranlasst eine neue Absatzformatierung.

appendRTItem

Deklaration

`void appendRTItem(RichTextItem item) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
Item	nein	RichTextItem	Item, dessen Inhalt kopiert werden soll.

Beschreibung

Mit dieser Methode kann der Inhalt eines RichText-Items an den Inhalt eines anderen RichText-Items angehängt werden.

appendStyle

Deklaration

`void appendStyle(RichTextStyle style) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
Style	nein	RichTextStyle	Neues Zeichenformat

Beschreibung

Mit dieser Methode wird eine Änderung der Zeichenformatierung veranlasst.

appendText

Deklaration

```
void appendText(String text) throws NotesException
```

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
text	nein	String	Anzuhängender Text

Beschreibung

Durch den Aufruf dieser Methode wird der angegebene Text an den Inhalt des Rich-Text-Items angehängt.

embedObject

Deklaration

```
EmbeddedObject embedObject(int type, String class, String source, String name)  
throws NotesException
```

Rückgabewert

EmbeddedObject

Zurückgegeben wird das neu erstellte Objekt.

Parameter

Name	Optional	Typ	Beschreibung
type	nein	Int	Typ des Objekts, entweder EMBED_ATTACHMENT, EMBED_OBJECT oder EMBED_OBJECTLINK
class	nein	String	Name der Anwendung zur Bearbeitung des Objekts
source	nein	String	Entweder Dateiname der Datei, die zur Generierung des Objekts genutzt wurde, oder der Name der verknüpften Datei
name	nein	String	Name des Objekts

Beschreibung

Diese Methode wird verwendet, um ein neues, eingebettetes oder verknüpfted Dokument zu erstellen. Außerdem wird die Methode auch verwendet für das Erstellen von Dateianhängen.

getEmbeddedObject

Deklaration

EmbeddedObject getEmbeddedObject(String name) throws NotesException

Rückgabewert

EmbeddedObject

wenn vorhanden, eine Referenz auf das Objekt

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name des gesuchten Objekts

Beschreibung

Mit dieser Methode erlangt man Zugriff auf bestehende eingebettete oder verknüpfte Objekte sowie auf Dateianhänge.

getFormattedText

Deklaration

String getFormattedText(boolean tabstrip, int linelen, int maxlen) throws NotesException

Rückgabewert

String

Parameter

Parameter Name	Optional	Typ	Beschreibung
tabstrip	nein	boolean	Mit diesem Parameter wird bestimmt, ob die Tabstops bei dem zurückzugebenden Text entfernt werden.
linelen	nein	int	Zeichen, ab dem die Ausgabe beginnt
maxlen	nein	int	Maximale Länge des zu lesenden Texts

Beschreibung

Mit dieser Methode kann der Inhalt eines RichText-Items als normale Zeichenkette ausgelesen werden.

9.25 Klasse RichTextParagraphStyle

9.25.1 Konstanten

Name	Wert
ALIGN_CENTER	3
ALIGN_FULL	2
ALIGN_LEFT	0
ALIGN_NOWRAP	4
ALIGN_RIGHT	1
PAGINATE_BEFORE	1
PAGINATE_DEFAULT	0
PAGINATE_KEEP_TOGETHER	4
PAGINATE_KEEP_WITH_NEXT	2
RULER_ONE_CENTIMETER	567
RULER_ONE_INCH	1440
TAB_CENTER	3
TAB_DECIMAL	2
TAB_LEFT	0
TAB_RIGHT	1

9.25.2 Eigenschaften

Alignment

Datentyp

int

Deklaration Getter

int getAlignment() throws NotesException

Deklaration Setter

void setAlignment(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.ALIGN_CENTER
RichTextParagraphStyle.ALIGN_FULL
RichTextParagraphStyle.ALIGN_LEFT
RichTextParagraphStyle.ALIGN_NOWRAP
RichTextParagraphStyle.ALIGN_RIGHT

Beschreibung

Mit dieser Eigenschaft wird die Ausrichtung bestimmt.

FirstLineLeftMargin

Datentyp

int

Deklaration Getter

int getFirstLineLeftMargin() throws NotesException

Deklaration Setter

void setFirstLineLeftMargin(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.RULER_ONE_CENTIMETER

RichTextParagraphStyle.RULER_ONE_INCH

Beschreibung

Mit dieser Eigenschaft kann der linke Einzug der ersten Zeile des Absatzes bestimmt werden.

InterLineSpacing

Datentyp

int

Deklaration Getter

int getInterLineSpacing() throws NotesException

Deklaration Setter

void setInterLineSpacing(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.SPACING_DOUBLE

RichTextParagraphStyle.SPACING_ONE_POINT_50

RichTextParagraphStyle.SPACING_SINGLE

Beschreibung

Diese Eigenschaft bestimmt den Zeilenabstand innerhalb eines Absatzes.

LeftMargin

Datentyp

int

Deklaration Getter

int getLeftMargin() throws NotesException

Deklaration Setter

void setLeftMargin(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.RULER_ONE_CENTIMETER

RichTextParagraphStyle.RULER_ONE_INCH

Beschreibung

Diese Eigenschaft bestimmt den linken Einzug des Absatzes.

Pagination

Datentyp

int

Deklaration Getter

int getPagination() throws NotesException

Deklaration Setter

void setPagination(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.PAGINATE_BEFORE

RichTextParagraphStyle.PAGINATE_DEFAULT

RichTextParagraphStyle.PAGINATE_KEEP_TOGETHER

RichTextParagraphStyle.PAGINATE_KEEP_WITH_NEXT

Beschreibung

Diese Eigenschaft steuert das Verhalten bei einem Abschnittswechsel bezüglich eines Seitenumbruchs.

RightMargin

Datentyp

int

Deklaration Getter

int getRightMargin() throws NotesException

Deklaration Setter

void setRightMargin(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.RULER_ONE_CENTIMETER
RichTextParagraphStyle.RULER_ONE_INCH

Beschreibung

Diese Eigenschaft bestimmt den rechten Einzug des Absatzes.

SpacingAbove

Datentyp

int

Deklaration Getter

int getSpacingAbove() throws NotesException

Deklaration Setter

void setSpacingAbove(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.SPACING_DOUBLE
RichTextParagraphStyle.SPACING_ONE_POINT_50
RichTextParagraphStyle.SPACING_SINGLE

Beschreibung

In dieser Eigenschaft wird der Abstand zum vorherigen Absatz bestimmt.

SpacingBelow

Datentyp

int

Deklaration Getter

int getSpacingBelow() throws NotesException

Deklaration Setter

void setSpacingBelow(int value) throws NotesException

Wertebereich

RichTextParagraphStyle.SPACING_DOUBLE
RichTextParagraphStyle.SPACING_ONE_POINT_50
RichTextParagraphStyle.SPACING_SINGLE

Beschreibung

In dieser Eigenschaft wird der Abstand zum nächsten Absatz bestimmt.

Tabs

Datentyp

`java.util.Vector(RichTextTab)`

Deklaration Getter

`java.util.Vector getTabs()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Durch das Auslesen dieser Eigenschaft erhält man einen Vektor mit Angaben zu den Tabstops.

9.25.3 Methoden

clearAllTabs

Deklaration

`void clearAllTabs()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Bei Aufruf dieser Methode werden alle Tabstops entfernt.

setTab

Deklaration

`void setTab(int position, int type)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
position	nein	int	Position des Tabstops in twips
type	nein	int	Eine der folgenden Ausrichtungen des Tabstops: RichTextParagraphStyle.TAB_CENTER RichTextParagraphStyle.TAB_DECIMAL RichTextParagraphStyle.TAB_LEFT RichTextParagraphStyle.TAB_RIGHT

Beschreibung

Diese Methode setzt einen Tabstop.

Die Maßeinheit twips wird verwendet, um Abstände für Tabstops anzugeben. Dabei gilt die folgende Umrechnung:

1440 twips = 1 inch

537 twips = 1 cm

Für diese Umrechnung stehen zwei Konstanten zur Verfügung. Die Konstante `RULER_ONE_INCH` hat den Wert 1440 und die Konstante `RULER_ONE_CENTIMETER` den Wert 537.

setTabs

Deklaration

`void setTabs(int count, int startpos, int interval) throws NotesException`
`void setTabs(int count, int startpos, int interval, int type) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
count	nein	int	Anzahl der einzufügenden Tabstops
startpos	nein	int	Startposition der einzufügenden Tabstops
interval	nein	int	Abstand zwischen den einzufügenden Tabstops
type	ja	int	Ausrichtung des einzufügenden Tabstops. Einer der folgenden Werte: RichTextParagraphStyle.TAB_CENTER RichTextParagraphStyle.TAB_DECIMAL RichTextParagraphStyle.TAB_LEFT RichTextParagraphStyle.TAB_RIGHT

Beschreibung

Mit dieser Methode können mehrere Tabulatoren mit einem Aufruf gesetzt werden.

9.26 Klasse RichTextStyle

9.26.1 Konstanten

Name	Wert
COLOR_BLACK	0
COLOR_BLUE	4
COLOR_CYAN	7
COLOR_DARK_BLUE	10
COLOR_DARK_CYAN	13
COLOR_DARK_GREEN	9
COLOR_DARK_MAGENTA	11
COLOR_DARK_RED	8
COLOR_DARK_YELLOW	12
COLOR_GRAY	14
COLOR_GREEN	3
COLOR_LIGHT_GRAY	15
COLOR_MAGENTA	5
COLOR_RED	2
COLOR_WHITE	1
COLOR_YELLOW	6
EFFECTS_EMBOSS	4
EFFECTS_EXTRUDE	5
EFFECTS_NONE	0
EFFECTS_SHADOW	3
EFFECTS_SUBSCRIPT	2
EFFECTS_SUPERSCRIPT	1
FONT_COURIER	4
FONT_HELV	1
FONT_ROMAN	0
STYLE_NO_CHANGE	255
MAYBE	255
NO	0
YES	1

9.26.2 Eigenschaften

Bold

Datentyp

int

Deklaration Getter

int getBold() throws NotesException

Deklaration Setter

void setBold(int value) throws NotesException

Wertebereich

RichTextStyle.YES

RichTextStyle.NO

RichTextStyle.STYLE_NO_CHANGE

RichTextStyle.MAYBE

Beschreibung

Diese Eigenschaft bestimmt bei der Zeichenformatierung den Fettdruck.

Color

Datentyp

int

Deklaration Getter

int getColor() throws NotesException

Deklaration Setter

void setColor(int value) throws NotesException

Wertebereich

RichTextStyle.COLOR_BLACK

RichTextStyle.COLOR_BLUE

RichTextStyle.COLOR_CYAN

RichTextStyle.COLOR_DARK_BLUE

RichTextStyle.COLOR_DARK_CYAN

RichTextStyle.COLOR_DARK_GREEN

RichTextStyle.COLOR_DARK_MAGENTA

RichTextStyle.COLOR_DARK_RED

RichTextStyle.COLOR_DARK_YELLOW

RichTextStyle.COLOR_GRAY

RichTextStyle.COLOR_GREEN
RichTextStyle.COLOR_LIGHT_GRAY
RichTextStyle.COLOR_MAGENTA
RichTextStyle.COLOR_RED
RichTextStyle.COLOR_WHITE
RichTextStyle.COLOR_YELLOW
RichTextStyle.STYLE_NO_CHANGE
RichTextStyle.MAYBE

Beschreibung

Durch diese Eigenschaft wird die Textfarbe bestimmt.

Effects

Datentyp

int

Deklaration Getter

int getEffects() throws NotesException

Deklaration Setter

void setEffects(int value) throws NotesException

Wertebereich

RichTextStyle.EFFECTS_EMBOSS
RichTextStyle.EFFECTS_EXTRUDE
RichTextStyle.EFFECTS_NONE
RichTextStyle.EFFECTS_SHADOW
RichTextStyle.EFFECTS_SUBSCRIPT
RichTextStyle.EFFECTS_SUPERSCRIPT
RichTextStyle.STYLE_NO_CHANGE
RichTextStyle.MAYBE

Beschreibung

Durch diese Eigenschaft können verschiedene Schrifteffekte an- und abgeschaltet werden.

Font

Datentyp

int

Deklaration Getter

int getFont() throws NotesException

Deklaration Setter

void setFont(int value) throws NotesException

Wertebereich

RichTextStyle.FONT_COURIER

RichTextStyle.FONT_HELV

RichTextStyle.FONT_ROMAN

RichTextStyle.STYLE_NO_CHANGE

RichTextStyle.MAYBE

Beschreibung

Diese Eigenschaft bestimmt die Schriftart.

FontSize**Datentyp**

int

Deklaration Getter

int getFontSize() throws NotesException

Deklaration Setter

void setFontSize(int value) throws NotesException

Wertebereich

Integer 1-250

RichTextStyle.STYLE_NO_CHANGE

RichTextStyle.MAYBE

Beschreibung

Diese Eigenschaft bestimmt die Schriftgröße.

Italic**Datentyp**

int

Deklaration Getter

int getItalic() throws NotesException

Deklaration Setter

void setItalic(int value) throws NotesException

Wertebereich

`RichTextStyle.YES`

`RichTextStyle.NO`

`RichTextStyle.STYLE_NO_CHANGE`

`RichTextStyle.MAYBE`

Beschreibung

Mit dieser Eigenschaft kann der Italic-Effekt (kursiv) für die Schrift bestimmt werden.

Parent**Datentyp**

Session

Deklaration Getter

Session getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet eine Referenz auf die übergeordnete Session.

PassThruHTML**Datentyp**

int

Deklaration Getter

int getPassThruHTML() throws NotesException

Deklaration Setter

void setPassThruHTML(int value) throws NotesException

Wertebereich

`RichTextStyle.YES`

`RichTextStyle.NO`

`RichTextStyle.STYLE_NO_CHANGE`

`RichTextStyle.MAYBE`

Beschreibung

Diese Eigenschaft bestimmt das Verhalten bei HTML-Code.

StrikeThrough

Datentyp

int

Deklaration Getter

int getStrikeThrough() throws NotesException

Deklaration Setter

void setStrikeThrough(int value) throws NotesException

Wertebereich

RichTextStyle.YES

RichTextStyle.NO

RichTextStyle.STYLE_NO_CHANGE

RichTextStyle.MAYBE

Beschreibung

Diese Eigenschaft steuert, ob der Text durchgestrichen werden soll.

Underline

Datentyp

int

Deklaration Getter

int getUnderline() throws NotesException

Deklaration Setter

void setUnderline(int value) throws NotesException

Wertebereich

RichTextStyle.YES

RichTextStyle.NO

RichTextStyle.STYLE_NO_CHANGE

RichTextStyle.MAYBE

Beschreibung

Mit dieser Eigenschaft wird gesteuert, ob der Text unterstrichen wird.

9.26.3 Methoden

Keine

9.27 Klasse RichTextTab

9.27.1 Eigenschaften

Position

Datentyp

int

Deklaration Getter

int getPosition() throws NotesException

Deklaration Setter

kein

Wertebereich

RULER_ONE_INCH

RULER_ONE_CENTIMETER

Beschreibung

Diese Eigenschaft gibt die Position des Tabulators in twips an.

Type

Datentyp

int

Deklaration Getter

int getType() throws NotesException

Deklaration Setter

kein

Wertebereich

RichTextParagraphStyle.TAB_CENTER

RichTextParagraphStyle.TAB_DECIMAL

RichTextParagraphStyle.TAB_LEFT

RichTextParagraphStyle.TAB_RIGHT

Beschreibung

Die Eigenschaft gibt die Ausrichtung des Tabulators an.

9.27.2 Methoden

clear

Deklaration

`void clear()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Die Methode dient zum Entfernen des Tabulators.

9.28 Klasse Session

9.28.1 Eigenschaften

AddressBooks

Datentyp

`java.util.Vector(Database)`

Deklaration Getter

`java.util.Vector getAddressBooks()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn diese Eigenschaft von einem Server aus abgefragt wird, so beinhaltet sie alle Domino Adressbücher, die für die aktuelle Sitzung gelten. Bei Aufruf von einer Arbeitsstation oder über einen Remote-Zugriff befinden sich in dieser Eigenschaft alle Domino Adressbücher und die privaten Adressbücher des aktuellen Anwenders.

AgentContext

Datentyp

AgentContext

Deklaration Getter

AgentContext getAgentContext() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Wenn die Abfrage dieser Eigenschaft aus einem Agent heraus erfolgt, liefert die Eigenschaft den AgentContext für den aktuellen Agent zurück. Sollte diese Eigenschaft ansonsten abgefragt werden, so liefert diese Eigenschaft den Wert null.

CommonUserName

Datentyp

String

Deklaration Getter

String getCommonUserName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert den Namen des Anwenders, der die Session erzeugt hat.

International

Datentyp

International

Deklaration Getter

International getInternational() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die aktuelle Sprachumgebung für die Sitzung.

IsConvertMIME

Datentyp

boolean

Deklaration Getter

boolean isConvertMIME() throws NotesException

Deklaration Setter

void setConvertMIME(boolean) throws NotesException

Wertebereich

true, false

Beschreibung

Bei dem Wert *true* in dieser Eigenschaft werden MIME-Elemente in RichText-Elemente konvertiert.

IsOnServer

Datentyp

boolean

Deklaration Getter

boolean isOnServer() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft liefert *true*, wenn die Sitzung auf einem Server läuft, ansonsten *false*.

IsValid

Datentyp

boolean

Deklaration Getter

boolean isValid()

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt bei einer Remote-Sitzung an, ob die Sitzung noch gültig ist.

NotesVersion

Datentyp

String

Deklaration Getter

String getNotesVersion() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Notes Version als Zeichenkette.

OrgDirectoryPath

Datentyp

String

Deklaration Getter

String getOrgDirectoryPath() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Verzeichnisnamen für den aktuellen Anwender.

Plattform

Datentyp

String

Deklaration Getter

String getPlatform() throws NotesException

Deklaration Setter

kein

Wertebereich

Inhalt	Plattform
"Macintosh"	Macintosh
"MS-DOS"	MS-DOS
"Netware"	NetWare
"OS/2v1"	OS/2® 16-bit
"OS/2v2"	OS/2 32-bit
"OS/400"	OS/400
"Windows/16"	Windows 16-bit
"Windows/32"	Windows 32-bit
"UNIX"	UNIX (Sun, SCO, HP-UX, AIX®)

Beschreibung

Diese Eigenschaft beinhaltet eine Beschreibung für die Plattform, auf der das Notes System läuft.

ServerName

Datentyp

String

Deklaration Getter

String getServerName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Falls die Eigenschaft auf einem Server abgefragt wird, liefert sie den Namen des Servers, andernfalls den Wert null.

UserGroupNameList**Datentyp**

java.util.Vector(Name)

Deklaration Getter

java.util.Vector getUserGroupNameList() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Liste der Gruppen, zu denen der aktuelle Anwender gehört.

UserName**Datentyp**

String

Deklaration Getter

String getUsername() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name des aktuellen Anwenders.

UserNameList

Datentyp

java.util.Vector(Name)

Deklaration Getter

java.util.Vector getUsernameList() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Liste aller Namen inklusive der Alternativnamen des aktuellen Anwenders oder Servers.

UserNameObject

Datentyp

Name

Deklaration Getter

Name getUsernameObject() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name des aktuellen Anwenders in Form einer Referenz auf die Klasse *Name*.

9.28.2 Methoden

createDataRange

Deklaration

`DateRange createDataRange()` throws `NotesException`

`DateRange createDataRange(DateTime startt, DateTime endt)` throws `NotesException`

`DateRange createDataRange(java.util.Date startt, java.util.Date endt)` throws `NotesException`

Rückgabewert

`DateRange`

eine Referenz auf den erstellten Datumsbereich

Parameter

Name	Optional	Typ	Beschreibung
startt	ja	<code>DateTime</code> <code>java.utl.Date</code>	Anfangsdatum oder Anfangszeit
endt	ja	<code>DateTime</code> <code>java.util.Date</code>	Enddatum oder Endzeit

Beschreibung

Diese Methode erstellt eine neue Instanz der Klasse *DateRange*.

createDateTime

Deklaration

`DateTime createDateTime(String date)` throws `NotesException`

`DateTime createDateTime(java.util.Date date)` throws `NotesException`

`DateTime createDateTime(java.util.Calendar date)` throws `NotesException`

Rückgabewert

`DateTime`

eine Referenz auf die neu erzeugte Instanz der Klasse `DateTime`

Parameter

Name	Optional	Typ	Beschreibung
date	nein	<code>String</code> <code>java.util.Date</code> <code>java.util.Calendar</code>	Datum

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *DateTime*.

createLog

Deklaration

Log createLog(String name) throws NotesException

Rückgabewert

Log

Referenz auf die neu erzeugte Log-Instanz

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Namen der neuen Log-Instanz

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *Log* mit dem angegebenen Namen.

createName

Deklaration

Name createName(String name) throws NotesException

Name createName(String name, String lang) throws NotesException

Rückgabewert

Name

Referenz auf die neu erzeugte Instanz der Klasse Name

Parameter

Name	Optional	Typ	Beschreibung
name	nein	String	Name
lang	nein	String	Sprache

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *Name* für den angegebenen Namen in der angegebenen Sprache.

createNewsletter

Deklaration

Newsletter createNewsletter(DocumentCollection collection) throws NotesException

Rückgabewert

Newsletter

eine Referenz auf die neu erzeugte Instanz der Klasse Newsletter

Parameter

Name	Optional	Typ	Beschreibung
collection	nein	DocumentCollection	Dokumentensammlung

Beschreibung

Diese Methode erstellt eine neue Instanz der Klasse *Newsletter* basierend auf der angegebenen Dokumentensammlung.

createRegistration

Deklaration

Registration createRegistration() throws NotesException

Rückgabewert

Registration

die Referenz auf die neu erzeugte Instanz der Klasse Registration

Parameter

keine

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *Registration*.

createRichTextParagraphStyle

Deklaration

RichTextParagraphStyle createRichTextParagraphStyle() throws NotesException

Rückgabewert

RichTextParagraphStyle

eine Referenz auf die neu erzeugte Instanz der Klasse RichTextParagraphStyle

Parameter

keine

Beschreibung

Diese Methode erstellt eine neue Instanz der Klasse *RichTextParagraphStyle*.

createRichTextStyle**Deklaration**

`RichTextStyle createRichTextStyle()` throws `NotesException`

Rückgabewert

`RichTextStyle`

eine Referenz auf die neu erzeugte Instanz der Klasse `RichTextStyle`

Parameter

keine

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *RichTextStyle*.

createStream**Deklaration**

`Stream createStream()` throws `NotesException`

Rückgabewert

`Stream`

eine Referenz auf die neu erzeugte Instanz der Klasse `Stream`

Parameter

keine

Beschreibung

Diese Methode erzeugt eine neue Instanz der Klasse *Stream*.

evaluate**Deklaration**

`java.util.Vector evaluate(String formula)` throws `NotesException`

`java.util.Vector evaluate(String formula, Document doc)` throws `NotesException`

Rückgabewert

`java.util.Vector`

das Ergebnis oder die Ergebnisse der Berechnung

Parameter

Name	Optional	Typ	Beschreibung
formual	nein	String	Die zu berechnende Formel
doc	ja	Document	Referenz auf das Dokument

Beschreibung

Diese Methode berechnet die angegebene Formel und nutzt dazu die Feldinhalte aus dem angegebenen Dokument.

freeTimeSearch

Deklaration

```
java.util.Vector freeTimeSearch(DateRange window, int duration, Object names,  
    boolean firstfit) throws NotesException
```

Rückgabewert

java.util.Vector
der oder die passenden Termine

Parameter

Name	Optional	Typ	Beschreibung
window	nein	DateRange	Zeitraum, in dem gesucht wird
duration	nein	int	Terminlänge in Minuten
names	nein	Object	Namen der zu dem Termin eingeladenen Personen oder Gruppen
firstfit	nein	boolean	Wenn der Wert true angegeben wird, wird der erste passende Termin zurückgegeben. Bei dem Wert false werden alle passenden Termine zurückgegeben.

Beschreibung

Diese Methode sucht in dem angegebenen Zeitraum eine Zeit, zu der alle eingeladenen Personen oder Gruppen keine anderen Termine in ihren Kalendern haben.

getDatabase

Deklaration

```
Database getDatabase(String server, String db) throws NotesException  
Database getDatabase(String server, String db, boolean createonfail)  
    throws NotesException
```

Rückgabewert

Database
Referenz auf die neue Instanz der geforderten Datenbank

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des Servers, auf dem die Datenbank geöffnet werden soll
db	nein	String	Dateiname der zu öffnenden Datenbank
createonfail	ja	boolean	Falls der Wert true angegeben wurde und die Datenbank noch nicht existiert, wird sie angelegt. Bei dem Wert false wird sie nicht angelegt.

Beschreibung

Diese Methode öffnet oder ggf. die angegebene Datenbank an.

getDbDirectory

Deklaration

DbDirectory getDbDirectory(String server) throws NotesException

Rückgabewert

DbDirectory

eine Referenz auf die neu angelegte Instanz der Klasse DbDirectory

Parameter

Name	Optional	Typ	Beschreibung
server	nein	String	Name des Servers, für den die neue Instanz angelegt werden soll

Beschreibung

Diese Methode erzeugt für den angegebenen Server eine neue Instanz der Klasse *DbDirectory*.

getEnvironmentString

Deklaration

String getEnvironmentString(String vname) throws NotesException

String getEnvironmentString(String vname, boolean issystem) throws NotesException

Rückgabewert

String

der Inhalt der Umgebungsvariable

Parameter

Name	Optional	Typ	Beschreibung
vname	nein	String	Name der Umgebungsvariablen
issystem	ja	boolean	Bei dem Wert true wird der angegebene Variablenname verwendet, bei false wird ein „\$“ vor den Namen gestellt.

Beschreibung

Diese Methode liefert den geforderten Inhalt der Umgebungsvariablen zurück

getEnvironmentValue

Deklaration

Object getEnvironmentValue(String vname) throws NotesException
Object getEnvironmentValue(String vname, boolean issystem) throws NotesException

Rückgabewert

Object
Inhalt der angegebenen Umgebungsvariablen

Parameter

Name	Optional	Typ	Beschreibung
vname	nein	String	Name der Umgebungsvariablen
issystem	ja	boolean	Bei dem Wert true wird der angegebene Variablenname verwendet, bei false wird ein „\$“ vor den Namen gestellt.

Beschreibung

Diese Methode liefert den geforderten Inhalt der Umgebungsvariablen zurück

getSessionToken

Deklaration

String getSessionToken() throws NotesException
String getSessionToken(String serverName) throws NotesException

Rückgabewert

String
ein Sessiontoken z. B. für eine laufende WebSphere-Session

Parameter

Name	Optional	Typ	Beschreibung
serverName	nein	String	Name des Servers, der für das SingleSignOn konfiguriert wurde

Beschreibung

Diese Methode dient zur Realisierung eines SingleSignOn-Verfahrens.

getURLDatabase

Deklaration

Database getURLDatabase() throws NotesException

Rückgabewert

Database

eine Referenz auf die WebNavigator-Datenbank

Parameter

keine

Beschreibung

Diese Methode liefert eine Referenz auf die WebNavigator-Datenbank.

resolve

Deklaration

Base resolve(String url) throws NotesException

Rückgabewert

Base

das adressierte Notes Objekt

Parameter

Name	Optional	Typ	Beschreibung
url	nein	String	URL-Adresse des zu öffnenden Notes Objekts

Beschreibung

Die Methode dient dazu, ein Notes Objekt über die URL-Adresse zu öffnen.

Dabei sind folgende Adressformen möglich:

protocol://host/database?OpenDatabase

protocol://host/database /view?OpenView

protocol://host/database /form?OpenForm

protocol://host/database/document?OpenDocument
 protocol://host/database /agent?OpenAgent

Für lokale Aufrufe ist der Name des Protokolls „notes“; ein Servername wird nicht angegeben.

setEnvironmentVar

Deklaration

void setEnvironmentVar(String vname, Object value) throws NotesException
 void setEnvironmentVar(String vname, Object value, boolean issystem)
 throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
vname	nein	String	Name der Umgebungsvariablen
value	nein	Object	Neuer Inhalt für die Umgebungsvariablen
issystem	ja	boolean	Bei dem Wert true wird der angegebene Variablenname verwendet, bei false wird ein „\$“ vor den Namen gestellt.

Beschreibung

Diese Methode dient zum Setzen von Umgebungsvariablen.

9.29 Klasse Stream

Verfügbar ab der Version 6.0

9.29.1 Konstanten

Name	Wert
EOL_ANY	4
EOL_CR	2
EOL_CRLF	0
EOL_LF	1
EOL_NONE	5
EOL_PLATFORM	3
STMREAD_LINE	0

9.29.2 Eigenschaften

Bytes

Datentyp

int

Deklaration Getter

int getBytes() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Größe des Streams in Byte.

Charset

Datentyp

String

Deklaration Getter

String getCharset() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet den Zeichensatz.

IsEOS

Datentyp

boolean

Deklaration Getter

boolean isEOS() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Über diese Eigenschaft erfährt man, ob das Ende des Streams erreicht wurde.

IsReadOnly

Datentyp

boolean

Deklaration Getter

boolean isReadOnly() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft, ob der File, auf dem der Stream basiert, schreibgeschützt ist.

Position

Datentyp

int

Deklaration Getter

int getPosition() throws NotesException

Deklaration Setter

void setPosition(int position) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Position im Stream.

9.29.3 Methoden

close

Deklaration

`void close()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Durch diese Methode wird der Stream geschlossen.

getContents

Deklaration

`void getContents(java.io.OutputStream stream)` throws `NotesException`

`void getContents(java.io.Writer stream)` throws `NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	java.io.OutputStream java.io.Writer	Stream, in den der Inhalt kopiert werden soll

Beschreibung

Die Methode kopiert den Inhalt des Streams in den als Parameter übergebenen Stream.

open

Deklaration

`boolean open(String pathname)` throws `NotesException`

`boolean open(String pathname, String charset)` throws `NotesException`

Rückgabewert

boolean

Parameter

Name	Optional	Typ	Beschreibung
pathname	nein	String	Dateiname
charset	ja	String	Zeichensatz

Beschreibung

Diese Methode öffnet die Datei und bindet den Stream an die Datei.

read

Deklaration

byte[] read() throws NotesException

byte[] read(int length) throws NotesException

Rückgabewert

byte[]

die gelsenen Daten

Parameter

Name	Optional	Typ	Beschreibung
length	ja	int	Die Anzahl der Bytes, die gelesen werden sollen

Beschreibung

Liest die angegebene Anzahl von Bytes und liefert diese zurück.

readText

Deklaration

String readText() throws NotesException

String readText(int constant1) throws NotesException

String readText(int constant1, int constant2) throws NotesException

Rückgabewert

String

Parameter

Name	Optional	Typ	Beschreibung
constant1	ja	int	Stream.STMREAD_LINE bedeutet, dass eine Zeile gelesen wird. Standardmäßig werden alle Zeilen gelesen.
constant2	ja	int	Dieser Parameter bestimmt, was als Zeilenende angesehen wird.

Beschreibung

Diese Methode liest Textzeilen aus dem Stream.

Wert des zweiten Parameters	Bedeutung
Stream.EOL_ANY	CR, CR+LF, LF
Stream.EOL_CR	CR
Stream.EOL_CRLF	CR + LF
Stream.EOL_LF	LF
Stream.EOL_NONE	Kein Zeichen für Zeilenende
Stream.EOL_PLATFORM	Plattformspezifische Zeichen

setContents

Deklaration

void setContents(java.io.InputStream stream) throws NotesException

void setContents(java.io.Reader stream) throws NotesException

Rückgabewert

keinen

Parameter

Name	Optional	Typ	Beschreibung
stream	nein	java.io.InputStream java.io.Reader	Stream aus dem die Daten kopiert werden sollen.

Beschreibung

Diese Methode kopiert den Inhalt des als Parameter übergebenen Streams in den Stream.

truncate

Deklaration

void truncate() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode löscht den Inhalt des Streams.

write

Deklaration

`int write(byte[] buffer) throws NotesException`

Rückgabewert

`int`

die Anzahl der Bytes, die geschrieben worden sind

Parameter

Name	Optional	Typ	Beschreibung
buffer	nein	byte[]	Die zu schreibenden Daten

Beschreibung

Diese Methode schreibt die übergebenen Daten in den Stream.

writeText

Deklaration

`int writeText(String text) throws NotesException`

`int writeText(String text, int endOfLine) throws NotesException`

Rückgabewert

`int`

die Anzahl der Bytes, die geschrieben worden sind

Parameter

Name	Optional	Typ	Beschreibung
text	nein	String	Der Text, der geschrieben werden soll
endOfLine	ja	int	Dieser Parameter bestimmt, was als Zeilenende angesehen wird.

Beschreibung

Diese Methode schreibt den Inhalt des ersten Parameters in den Stream.

Wert des zweiten Parameters	Bedeutung
Stream.EOL_ANY	CR, CR+LF, LF
Stream.EOL_CR	CR
Stream.EOL_CRLF	CR + LF
Stream.EOL_LF	LF
Stream.EOL_NONE	Kein Zeichen für Zeilenende
Stream.EOL_PLATFORM	Plattformspezifische Zeichen

9.30 Klasse View

9.30.1 Konstanten

Name	Wert
SPACING_DOUBLE	4
SPACING_ONE_POINT_25	1
SPACING_ONE_POINT_50	2
SPACING_ONE_POINT_75	3
SPACING_SINGLE	0

9.30.2 Eigenschaften

Aliases

Datentyp

`java.util.Vector(String)`

Deklaration Getter

`java.util.Vector getAliases()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befinden sich die alternativen Namen der Ansicht.

AllEntries

Datentyp

`ViewEntryCollection`

Deklaration Getter

`ViewEntryCollection getAllEntries()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Sammlung der in der Ansicht enthaltenen Dokumente in Form einer Instanz der Klasse *ViewEntryCollection*.

BackgroundColor

Datentyp

int

Deklaration Getter

int getBackgroundColor() throws NotesException

Deklaration Setter

kein

Wertebereich

RichTextStyle.COLOR_BLACK
RichTextStyle.COLOR_WHITE
RichTextStyle.COLOR_RED
RichTextStyle.COLOR_GREEN
RichTextStyle.COLOR_BLUE
RichTextStyle.COLOR_MAGENTA
RichTextStyle.COLOR_YELLOW
RichTextStyle.COLOR_CYAN
RichTextStyle.COLOR_DARK_RED
RichTextStyle.COLOR_DARK_GREEN
RichTextStyle.COLOR_DARK_BLUE
RichTextStyle.COLOR_DARK_MAGENTA
RichTextStyle.COLOR_DARK_YELLOW
RichTextStyle.COLOR_DARK_CYAN
RichTextStyle.COLOR_GRAY
RichTextStyle.COLOR_LIGHT_GRAY

Beschreibung

Diese Eigenschaft beinhaltet die Hintergrundfarbe.

ColumnCount

Datentyp

int

Deklaration Getter

int getColumnCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Aus dieser Eigenschaft kann die Anzahl der Spalten ermittelt werden.

ColumnNames

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getColumnNames() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Vektor mit den Spaltennamen der Ansicht.

Columns

Datentyp

java.util.Vector(ViewColumn)

Deklaration Getter

java.util.Vector getColumns() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich ein Vektor mit den Spalten der Ansicht als Instanzen der Klasse *ViewColumn*.

Created

Datentyp

DateTime

Deklaration Getter

DateTime getCreated() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Erstelldatum der Ansicht befindet sich in dieser Eigenschaft.

HeaderLines

Datentyp

int

Deklaration Getter

int getHeaderLines() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Zeilenanzahl in der Kopfzeile der Ansicht kann dieser Eigenschaft entnommen werden.

IsAutoUpdate

Datentyp

boolean

Deklaration Getter

boolean isAutoUpdate() throws NotesException

Deklaration Setter

`void setAutoUpdate(boolean flag) throws NotesException`

Wertebereich

true, false

Beschreibung

Durch diese Eigenschaft wird gesteuert, ob die Ansicht automatisch aktualisiert wird.

IsCalendar

Datentyp

boolean

Deklaration Getter

`boolean isCalendar() throws NotesException`

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob die Ansicht im Kalenderformat ist, verrät diese Eigenschaft.

IsCategorized

Datentyp

boolean

Deklaration Getter

`boolean isCategorized() throws NotesException`

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft, ob die Ansicht kategorisiert ist.

IsConflict

Datentyp

boolean

Deklaration Getter

boolean isConflict() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft, ob die Ansicht für Konflikteüberprüfung geeignet ist.

IsDefaultView

Datentyp

boolean

Deklaration Getter

boolean isDefaultView() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

In jeder Datenbank existiert eine Standardansicht. Ob diese Ansicht die Standardansicht ist, verrät diese Eigenschaft.

IsFolder

Datentyp

boolean

Deklaration Getter

boolean isFolder() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft dient zum Kennzeichen von Ordnern.

IsHierarchical

Datentyp

boolean

Deklaration Getter

boolean isHierarchical() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob die Ansicht Antwortdokumente hierarchisch anzeigt.

IsModified

Datentyp

boolean

Deklaration Getter

boolean isModified() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob an den Daten in der Ansicht eine Änderung stattgefunden hat und die Ansicht somit aktualisiert werden muss.

IsPrivate

Datentyp

boolean

Deklaration Getter

boolean isPrivate() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Eine private Ansicht liefert in dieser Eigenschaft den Wert *true* zurück.

IsProtectReaders

Datentyp

boolean

Deklaration Getter

boolean isProtectReaders() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* hat, werden bei der Replikation Änderungen im Feld *\$Reader* der Ansicht nicht übernommen.

LastModified

Datentyp

DateTime

Deklaration Getter

DateTime getLastModified() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Hier bekommt man das Datum der letzten Änderung zurück.

Name

Datentyp

String

Deklaration Getter

String getName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Name der Ansicht befindet sich in dieser Eigenschaft.

Parent

Datentyp

Database

Deklaration Getter

Database getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Eine Referenz auf die Datenbank befindet sich in dieser Eigenschaft.

Readers

Datentyp

java.util.Vector(String)

Deklaration Getter

java.util.Vector getReaders() throws NotesException

Deklaration Setter

void setReaders(java.util.Vector reader) throws NotesException

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft findet man den Inhalt des Felds *\$Readers* der Ansicht wieder.

RowLines

Datentyp

int

Deklaration Getter

int getRowLines() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Anzahl der Zeilen für einen Eintrag in der Ansicht wird durch den Inhalt dieser Eigenschaft angegeben.

Spacing

Datentyp

int

Deklaration Getter

int getSpacing() throws NotesException

Deklaration Setter

kein

Wertebereich

View.SPACING_DOUBLE

View.SPACING_ONE_POINT_25

View.SPACING_ONE_POINT_50

View.SPACING_ONE_POINT_75

View.SPACING_SINGLE

Beschreibung

Der Abstand zwischen den Zeilen in der Ansicht wird über diese Eigenschaft bestimmt.

TopLevelEntryCount

Datentyp

int

Deklaration Getter

int getTopLevelEntryCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Anzahl der Top-Level-Einträge befindet sich in dieser Eigenschaft.

UniversalID

Datentyp

String

Deklaration Getter

String getUniversalID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Universal-ID der Ansicht.

9.30.3 Methoden

clear

Deklaration

`void clear()` throws `NotesException`

Rückgabewert

kein

Parameter

keine

Beschreibung

Diese Methode löscht den Eintrag für die Volltextsuche der Ansicht.

createViewNav

Deklaration

`ViewNavigator createViewNav()` throws `NotesException`

`ViewNavigator createViewNav(int cacheSize)` throws `NotesException`

Rückgabewert

`ViewNavigator`

der neu erstellte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt eine Ansichtsnavigation.

createViewNavFrom

Deklaration

`ViewNavigator createViewNavFrom(Object entry)` throws `NotesException`

`ViewNavigator createViewNavFrom(Object entry, int cacheSize)` throws `NotesException`

Rückgabewert

`ViewNavigator`

der neu erzeugte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	Object	Der Eintrag, bei dem begonnen werden soll
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt einen neuen Ansichtsnavigator und beginnt bei dem angegebenen Eintrag.

createViewNavFromCategory

Deklaration

ViewNavigator createViewNavFromCategory(String categoryName) throws
NotesException

ViewNavigator createViewNavFromCategory(String categoryName, int cacheSize)
throws NotesException

Rückgabewert

ViewNavigator
der neu erzeugte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
categoryName	nein	String	Kategorienbezeichnung
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt einen Ansichtsnavigator für alle Einträge in der angegebenen Kategorie.

createViewNavFromChildren

Deklaration

ViewNavigator createViewNavFromChildren(Object entry) throws NotesException
ViewNavigator createViewNavFromChildren(Object entry, int cacheSize) throws
NotesException

Rückgabewert

ViewNavigator
der neu erzeugte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	Object	Der Eintrag, dessen untergeordnete Einträge aufgenommen werden sollen
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt einen neuen Ansichtsnavigator für alle Einträge, die dem als Parameter angegebenen Eintrag direkt untergeordnet sind.

createViewNavFromDescendants

Deklaration

ViewNavigator createViewNavFromDescendants(Object entry) throws
NotesException

ViewNavigator createViewNavFromDescendants(Object entry, int cacheSize)
throws NotesException

Rückgabewert

ViewNavigator
der neu erzeugte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	Object	Der Eintrag, dessen untergeordnete Einträge aufgenommen werden sollen
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt einen neuen Ansichtsnavigator für alle Einträge, die dem als Parameter angegebenen Eintrag untergeordnet sind.

createViewNavMaxLevel

Deklaration

ViewNavigator createViewNavMaxLevel(int level) throws NotesException
ViewNavigator createViewNavMaxLevel(int level, int cacheSize) throws
NotesException

Rückgabewert

ViewNavigator
der neu erzeugte Ansichtsnavigator

Parameter

Name	Optional	Typ	Beschreibung
level	nein	int	Alle Dokumente bis zu der hier angegebenen Ebene werden im Ansichtsnavigator berücksichtigt.
cacheSize	ja	int	Die Größe des Caches

Beschreibung

Diese Methode erzeugt einen Ansichtsnavigator, der alle Dokumente bis zu der angegebenen Ebene beinhaltet.

FTSearch

Deklaration

int FTSearch(String query) throws NotesException

int FTSearch(String query, int maxdocs) throws NotesException

Rückgabewert

int

die Anzahl der gefundenen Dokumente

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Suchformel
maxdocs	ja	int	Die maximale Anzahl der Dokumente, die zurückgeliefert werden sollen

Beschreibung

Diese Methode führt eine Volltextsuche aus und wirft alle nicht passenden Dokumente aus der Ansicht.

getAllDocumentsByKey

Deklaration

DocumentCollection getAllDocumentsByKey(java.util.Vector keys) throws NotesException

DocumentCollection getAllDocumentsByKey(Object key) throws NotesException

DocumentCollection getAllDocumentsByKey(java.util.Vector keys, boolean exact) throws NotesException

DocumentCollection getAllDocumentsByKey(Object key, boolean exact) throws NotesException

Rückgabewert

DocumentCollection

Dokumentensammlung mit allen Dokumenten, die dem Suchkriterium entsprechen

Parameter

Name	Optional	Typ	Beschreibung
keys	nein	java.util.Vector	Liste mit Werten, nach denen in den Spalten gesucht werden soll
key	nein	Object	Wert, nach dem in der ersten Spalte gesucht werden soll
exact	ja	boolean	Dieser Parameter bestimmt, ob eine vollständige Übereinstimmung notwendig ist, oder ob es reicht, wenn der angegebene Wert teilweise in der entsprechenden Spalte vorhanden ist.

Beschreibung

Diese Methode vergleicht den oder die angegebenen Daten mit den Einträgen in den entsprechenden Spalten.

getAllEntriesByKey

Deklaration

ViewEntryCollection getAllEntriesByKey(java.util.Vector keys) throws
NotesException

ViewEntryCollection getAllEntriesByKey(Object key) throws NotesException

ViewEntryCollection getAllEntriesByKey(java.util.Vector keys, boolean exact)
throws NotesException

ViewEntryCollection getAllEntriesByKey(Object key, boolean exact) throws
NotesException

Rückgabewert

ViewEntryCollection

Die Sammlung beinhaltet alle Einträge, die dem Suchkriterium entsprechen.

Parameter

Name	Optional	Typ	Beschreibung
keys	nein	java.util.Vector	Liste mit Werten, nach denen in den Spalten gesucht werden soll
key	nein	Object	Wert, nach dem in der ersten Spalte gesucht werden soll
exact	ja	boolean	Dieser Parameter bestimmt, ob eine vollständige Übereinstimmung notwendig ist, oder ob es reicht, wenn der angegebene Wert teilweise in der entsprechenden Spalte vorhanden ist.

Beschreibung

Diese Methode vergleicht den oder die angegebenen Daten mit den Einträgen in den entsprechenden Spalten und liefert die Dokumente, die dem Kriterium entsprechen.

getChild

Deklaration

Document getChild(Document doc) throws NotesException

Rückgabewert

Document

das erste dem angegebenen Dokument untergeordnete Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Dokument, dessen untergeordnetes Dokument gesucht werden soll

Beschreibung

Diese Methode liefert das erste dem angegebenen Dokument untergeordnete Dokument.

getColumn

Deklaration

ViewColumn getColumn(int columnNumber) throws NotesException

Rückgabewert

ViewColumn

die entsprechende Spalte

Parameter

Name	Optional	Typ	Beschreibung
columnNumber	nein	int	Spaltennummer

Beschreibung

Die Methode liefert die Spalte der angegebenen Position zurück.

getDocumentByKey

Deklaration

Document getDocumentByKey(java.util.Vector keys) throws NotesException

Document getDocumentByKey(Object key) throws NotesException

Document getDocumentByKey(java.util.Vector keys, boolean exact) throws
NotesException

Document getDocumentByKey(Object key, boolean exact) throws NotesException

Rückgabewert

Document

das erste Dokument, das dem Suchkriterium entspricht

Parameter

Name	Optional	Typ	Beschreibung
keys	nein	java.util.Vector	Liste mit Werten, nach denen in den Spalten gesucht werden soll
key	nein	Object	Wert, nach dem in der ersten Spalte gesucht werden soll
exact	ja	boolean	Dieser Parameter bestimmt, ob eine vollständige Übereinstimmung notwendig ist, oder ob es reicht, wenn der angegebene Wert teilweise in der entsprechenden Spalte vorhanden ist.

Beschreibung

Diese Methode vergleicht den oder die angegebenen Daten mit den Einträgen in den entsprechenden Spalten und liefert das erste Dokument, das dem Kriterium entspricht.

getEntryByKey

Deklaration

ViewEntry getEntryByKey(java.util.Vector keys) throws NotesException

ViewEntry getEntryByKey(Object key) throws NotesException

ViewEntry getEntryByKey(java.util.Vector keys, boolean exact) throws
NotesException

ViewEntry getEntryByKey(Object key, boolean exact) throws NotesException

Rückgabewert

ViewEntry

der erste Eintrag, der dem Suchkriterium entspricht

Parameter

Name	Optional	Typ	Beschreibung
keys	nein	java.util.Vector	Liste mit Werten, nach denen in den Spalten gesucht werden soll
key	nein	Object	Wert, nach dem in der ersten Spalte gesucht werden soll
exact	ja	boolean	Dieser Parameter bestimmt, ob eine vollständige Übereinstimmung notwendig ist, oder ob es reicht, wenn der angegebene Wert teilweise in der entsprechenden Spalte vorhanden ist.

Beschreibung

Diese Methode vergleicht den oder die angegebenen Daten mit den Einträgen in den entsprechenden Spalten und liefert den ersten Eintrag, der dem Kriterium entspricht.

getFirstDocument

Deklaration

Document getFirstDocument() throws NotesException

Rückgabewert

Document
das erste Dokument der Ansicht

Parameter

keine

Beschreibung

Diese Eigenschaft liefert das erste Dokument der Ansicht zurück.

getLastDocument

Deklaration

Document getLastDocument() throws NotesException

Rückgabewert

Document
das letzte Dokument der Ansicht

Parameter

keine

Beschreibung

Diese Methode liefert das letzte Dokument der Ansicht zurück.

getNextDocument

Deklaration

Document getNextDocument(Document doc) throws NotesException

Rückgabewert

Document

das nächste Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Das Dokument, dessen Nachfolger gesucht wird

Beschreibung

Die Methode sucht das dem angegebenen Dokument folgende Dokument.

getNextSibling

Deklaration

Document getNextSibling(Document doc) throws NotesException

Rückgabewert

Document

das nächste Dokument auf derselben Ebene

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Das Dokument, dessen Nachfolger gesucht wird

Beschreibung

Diese Methode sucht das dem angegebenen Dokument folgende Dokument in derselben Ebene.

getNthDocument

Deklaration

Document getNthDocument(int n) throws NotesException

Rückgabewert

Document

gefundenes Dokument

Parameter

Name	Optional	Typ	Beschreibung
N	nein	int	Position des gesuchten Dokuments

Beschreibung

Diese Methode liefert das Dokument zurück, das sich an der angegebenen Position befindet.

getParentDocument

Deklaration

Document getParentDocument(Document doc) throws NotesException

Rückgabewert

Document
das gefundene Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Das Dokument, dessen übergeordnetes Dokument ermittelt werden soll

Beschreibung

Diese Methode liefert das Dokument, das dem angegebenen Dokument übergeordnet ist.

getPrevDocument

Deklaration

Document getPrevDocument(Document doc) throws NotesException

Rückgabewert

Document
das gefundene Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Das Dokument, dessen Vorgänger gesucht werden soll

Beschreibung

Diese Methode liefert das Vorgängerdokument des angegebenen Dokuments.

getPrevSibling

Deklaration

Document getPrevSibling(Document doc) throws NotesException

Rückgabewert

Document
das gefundene Dokument

Parameter

Name	Optional	Typ	Beschreibung
doc	nein	Document	Das Dokument, dessen Vorgänger gesucht wird

Beschreibung

Diese Methode sucht das Vorgängerdokument des angegebenen Dokuments auf derselben Ebene.

refresh

Deklaration

void refresh() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Der Aufruf dieser Methode aktualisiert den Inhalt in der Ansicht.

remove

Deklaration

void remove() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Mit dieser Methode wird die Ansicht entfernt.

9.31 Klasse ViewColumn

9.31.1 Konstanten

Name	Wert
ALIGN_CENTER	2
ALIGN_LEFT	0
ALIGN_RIGHT	1
ATTR_PARENS	2
ATTR_PERCENT	4
ATTR_PUNCTUATED	1
FMT_MD	2
FMT_YM	3
FMT_YMD	0
FMT_Y4M	6
FONT_BOLD	1
FONT_ITALIC	2
FONT_STRIKEOUT	8
FONT_UNDERLINE	4
FMT_CURRENCY	3
FMT_FIXED	1
FMT_GENERAL	0
FMT_SCIENTIFIC	2
SEP_COMMA	2
SEP_NEWLINE	4
SEP_SEMICOLON	3
SEP_SPACE	1
FMT_DATE	0
FMT_DATETIME	2
FMT_TIME	1
FMT_TODAYTIME	3
FMT_HM	0
FMT_HMS	1
FMT_ALWAYS	2
FMT_NEVER	0
FMT_SOMETIMES	1

9.31.2 Eigenschaften

Alignment

Datentyp

int

Deklaration Getter

int getAlignment() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.ALIGN_CENTER

ViewColumn.ALIGN_LEFT

ViewColumn.ALIGN_RIGHT

Beschreibung

Diese Eigenschaft gibt die Ausrichtung des Inhalts in der Spalte wieder.

DateFmt

Datentyp

int

Deklaration Getter

int getDateFmt() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.FMT_MD

ViewColumn.FMT_YM

ViewColumn.FMT_Y4M

ViewColumn.FMT_YMD

Beschreibung

Durch diese Eigenschaft wird das Datumsformat des Inhalts der Spalte definiert.

FontColor

Datentyp

int

Deklaration Getter

int getFontColor() throws NotesException

Deklaration Setter

kein

Wertebereich

RichTextStyle.COLOR_BLACK

RichTextStyle.COLOR_BLUE

RichTextStyle.COLOR_CYAN

RichTextStyle.COLOR_DARK_BLUE

RichTextStyle.COLOR_DARK_CYAN

RichTextStyle.COLOR_DARK_GREEN

RichTextStyle.COLOR_DARK_MAGENTA

RichTextStyle.COLOR_DARK_RED

RichTextStyle.COLOR_DARK_YELLOW

RichTextStyle.COLOR_GRAY

RichTextStyle.COLOR_GREEN

RichTextStyle.COLOR_LIGHT_GRAY

RichTextStyle.COLOR_MAGENTA

RichTextStyle.COLOR_RED

RichTextStyle.COLOR_WHITE

RichTextStyle.COLOR_YELLOW

Beschreibung

Die Schriftfarbe befindet sich in dieser Eigenschaft.

FontFace

Datentyp

String

Deklaration Getter

String getFontFace() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Schriftart der in der Ansicht präsentierten Daten.

FontPointSize

Datentyp

int

Deklaration Getter

int getFontPointSize() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Schriftgröße für die Anzeige der Daten befindet sich in dieser Eigenschaft.

FontStyle

Datentyp

int

Deklaration Getter

int getFontStyle() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.FONT_BOLD

ViewColumn.FONT_ITALIC

ViewColumn.FONT_UNDERLINE

ViewColumn.FONT_STRIKEOUT

Beschreibung

Der Schriftstil der in der Ansicht gezeigten Daten wird über diese Eigenschaft bestimmt.

Formula

Datentyp

String

Deklaration Getter

String getFormula() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Hier befindet sich, wenn vorhanden, die Formel zur Berechnung des Spaltenwerts.

HeaderAlignment

Datentyp

int

Deklaration Getter

int getHeaderAlignment() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.ALIGN_CENTER

ViewColumn.ALIGN_LEFT

ViewColumn.ALIGN_RIGHT

Beschreibung

Die Ausrichtung des Spaltentitels verrät diese Eigenschaft.

IsAccentSensitiveSort

Datentyp

boolean

Deklaration Getter

boolean isAccentSensitiveSort() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob bei der Sortierung dieser Spalte die Akzente berücksichtigt werden.

IsCaseSensitiveSort

Datentyp

boolean

Deklaration Getter

boolean isCaseSensitiveSort() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob bei der Sortierung die Groß-/Kleinschreibung relevant ist.

isCategory

Datentyp

boolean

Deklaration Getter

boolean isCategory() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob eine Spalte kategorisiert ist.

IsField

Datentyp

boolean

Deklaration Getter

boolean isField() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft steht auf *true*, wenn die Spalte auf einem Feld basiert.

IsFormula

Datentyp

boolean

Deklaration Getter

boolean isFormula() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Dass es sich um eine Formel zur Berechnung der Spalte handelt, verrät diese Eigenschaft.

IsHidden

Datentyp

boolean

Deklaration Getter

boolean isHidden() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob es sich hier um eine versteckte Spalte handelt, verrät diese Eigenschaft.

IsHideDetail

Datentyp

boolean

Deklaration Getter

boolean isHideDetail() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob bei der Detailanzeige der Inhalt in der Spalte angezeigt werden soll.

IsIcon

Datentyp

boolean

Deklaration Getter

boolean isIcon() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob der Inhalt der Spalte in Form eines Symbols dargestellt wird.

IsResize

Datentyp

boolean

Deklaration Getter

boolean isResize() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob die Größe einer Spalte verändert werden kann, verrät diese Eigenschaft.

IsRessortAscending

Datentyp

boolean

Deklaration Getter

boolean isResortAscending() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft zeigt, ob die Suchrichtung aufsteigend ist.

IsResortDescending

Datentyp

boolean

Deklaration Getter

boolean isResortDescending() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft zeigt eine absteigende Suchrichtung an.

IsResortToView

Datentyp

boolean

Deklaration Getter

boolean isResortToView() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob die Suchreihenfolge der Spalte manuell von Anwender verändert werden kann.

IsResponse

Datentyp

boolean

Deklaration Getter

boolean isResponse() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft bestimmt, ob die Spalte nur Antwortdokumente beinhaltet.

IsSecondaryResort

Datentyp

boolean

Deklaration Getter

boolean isSecondaryResort() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob die Spalte das zweite Sortierkriterium ist.

IsSecondaryResortDescending

Datentyp

boolean

Deklaration Getter

boolean isSecondaryResortDescending() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft liefert *true* zurück, wenn die Sortierreihenfolge in der zweiten sortierten Spalte abwärts ist.

IsShowTwistie

Datentyp

boolean

Deklaration Getter

boolean isShowTwistie() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob ein Symbol zum Auf- und Zuklappen einer Ansicht angezeigt wird, kann durch diese Eigenschaft in Erfahrung gebracht werden.

IsSortDescending

Datentyp

boolean

Deklaration Getter

boolean isSortDescending() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Methode verrät, ob die Spalte absteigend sortiert ist.

IsSorted

Datentyp

boolean

Deklaration Getter

boolean isSorted() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Ob diese Spalte überhaupt sortiert ist, bestimmt diese Methode.

itemName

Datentyp

String

Deklaration Getter

String getItemName() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich der Name der Spalte.

ListSep

Datentyp

int

Deklaration Getter

int getListSep() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.SEP_COMMA

ViewColumn.SEP_NEWLINE

ViewColumn.SEP_SEMICOLON

ViewColumn.SEP_SPACE

Beschreibung

Diese Eigenschaft verrät, mit welchem Zeichen die Einträge einer Liste separiert werden.

NumberAttrib

Datentyp

int

Deklaration Getter

`int getNumberAttrib()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

`ViewColumn.ATTR_PARENS`

`ViewColumn.ATTR_PERCENT`

`ViewColumn.ATTR_PUNCTUATED`

Beschreibung

Diese Eigenschaft liefert Informationen über die Formatierung von numerischen Ausgaben in der Spalte.

NumberDigits**Datentyp**

`int`

Deklaration Getter

`int getNumberDigits()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Anzahl der Nachkommastellen.

NumberFormat**Datentyp**

`int`

Deklaration Getter

`int getNumberFormat()` throws `NotesException`

Deklaration Setter

kein

Wertebereich

ViewColumn.FMT_CURRENCY
ViewColumn.FMT_FIXED
ViewColumn.FMT_GENERAL
ViewColumn.FMT_SCIENTIFIC

Beschreibung

Diese Eigenschaft steuert die Anzeige der numerischen Daten.

Parent**Datentyp**

View

Deklaration Getter

View getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert eine Referenz auf die übergeordnete Ansicht zurück.

Position**Datentyp**

int

Deklaration Getter

int getPosition() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Durch diese Eigenschaft kann ermittelt werden, an welcher Position sich eine Spalte befindet.

TimeDateFmt

Datentyp

int

Deklaration Getter

int getTimeDateFmt() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.FMT_DATE

ViewColumn.FMT_DATETIME

ViewColumn.FMT_TIME

ViewColumn.FMT_TODAYTIME

Beschreibung

Diese Eigenschaft definiert das Aussehen von Datumswerten in der Spalte.

TimeFmt

Datentyp

int

Deklaration Getter

int getTimeFmt() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.FMT_HM

ViewColumn.FMT_HMS

Beschreibung

Diese Eigenschaft beeinflusst das Aussehen von Zeitangaben in der Spalte.

TimeZoneFmt

Datentyp

int

Deklaration Getter

int getTimeZoneFmt() throws NotesException

Deklaration Setter

kein

Wertebereich

ViewColumn.FMT_ALWAYS

ViewColumn.FMT_NEVER

ViewColumn.FMT_SOMETIMES

Beschreibung

Durch diese Eigenschaft wird definiert, welches Format die Angabe der Zeitzone aufweisen soll.

Title**Datentyp**

String

Deklaration Getter

String getTitle() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

In dieser Eigenschaft befindet sich die Titelzeile.

Width**Datentyp**

int

Deklaration Getter

int getWidth() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Breite einer Spalte.

9.31.3 Methoden

Keine

9.32 Klasse ViewEntry

9.32.1 Konstanten

Name	Wert
TYPE_CATEGORY	2
TYPE_DOCUMENT	1
TYPE_TOTAL	3

9.32.2 Eigenschaften

ChildCount

Datentyp

int

Deklaration Getter

int getChildCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Anzahl der direkt untergeordneten Einträge.

ColumnIndentLevel

Datentyp

int

Deklaration Getter

int getColumnIndentLevel() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Ebene des aktuellen Eintrags.

ColumnValues

Datentyp

java.util.Vector(Double,DateTime,String)

Deklaration Getter

java.util.Vector getColumnValues() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der Inhalt der Spalten für diesen Eintrag wird durch diese Eigenschaft bestimmt.

DescendantCount

Datentyp

int

Deklaration Getter

int getDescendantCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Anzahl der Einträge, die direkt oder indirekt dem Eintrag untergeordnet sind, wird hier angegeben.

Document

Datentyp

Document

Deklaration Getter

Document getDocument() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Das Dokument, auf das sich der Eintrag bezieht, wird mit dieser Eigenschaft bestimmt.

FTSearchScore

Datentyp

int

Deklaration Getter

int getFTSearchScore() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Relevanz, die dieser Eintrag bezüglich der letzten Volltextsuche bekommen hat.

IndentLevel

Datentyp

int

Deklaration Getter

int getIndentLevel() throws NotesException

Deklaration Setter

kein

Wertebereich

int

Beschreibung

Die hierarchische Position des Eintrags beinhaltet diese Eigenschaft.

IsCategory

Datentyp

boolean

Deklaration Getter

boolean isCategory() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft gibt Auskunft darüber, ob die Spalte eine Kategorie ist.

isConflict

Datentyp

boolean

Deklaration Getter

boolean isConflict() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Wenn diese Eigenschaft den Wert *true* besitzt, besteht ein Replikationskonflikt bei diesem Dokument.

IsDocument**Datentyp**

boolean

Deklaration Getter

boolean isDocument() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob es sich bei dem Eintrag in der Ansicht um ein Dokument handelt.

IsTotal**Datentyp**

boolean

Deklaration Getter

boolean isTotal() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft enthält den Wert *true*, wenn die Spalte zur Summenbildung genutzt wird.

IsValid

Datentyp

boolean

Deklaration Getter

boolean isValid() throws NotesException

Deklaration Setter

kein

Wertebereich

true, false

Beschreibung

Diese Eigenschaft verrät, ob es sich bei dem Eintrag in der Ansicht um ein real existierendes und nicht um ein gelöscht Dokument handelt.

NotesID

Datentyp

String

Deklaration Getter

String getNoteID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Note-ID des entsprechenden Dokuments.

Parent

Datentyp

Object

Deklaration Getter

Object getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Liefert das übergeordnete Objekt. Dieses kann eine Instanz der Klassen *View*, *ViewNavigator* oder *ViewEntryCollection* sein.

SiblingCount

Datentyp

int

Deklaration Getter

int getSiblingCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft beinhaltet die Anzahl der Dokumente auf derselben Ebene.

UniversalID

Datentyp

String

Deklaration Getter

String getUniversalID() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Diese Eigenschaft liefert die Universal-ID des Dokuments, auf dem der Eintrag basiert.

9.32.3 Methoden

GetPosition

Deklaration

String getPosition(char separator) throws NotesException

Rückgabewert

String

die hierachische numerische Position des Eintrags

Parameter

Name	Optional	Typ	Beschreibung
separator	nein	char	Trennzeichen bei der Angabe

Beschreibung

Diese Methode liefert die hierachische numerische Position des Eintrags in einer Zeichenkette.

9.33 Klasse ViewEntryCollection

9.33.1 Eigenschaften

Count

Datentyp

int

Deklaration Getter

int getCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Eigenschaft gibt Auskunft über die Anzahl der Einträge in der Sammlung.

Parent

Datentyp

View

Deklaration Getter

View getParent() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Eigenschaft beinhaltet die übergeordnete Ansicht.

Query

Datentyp

String

Deklaration Getter

String getQuery() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Eigenschaft enthält die Suchformel, mit der die Sammlung erzeugt wurde.

9.33.2 Methoden**addEntry**

Deklaration

void addEntry(Object obj) throws NotesException

void addEntry(Object obj, boolean checkDups) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
obj	nein	Object	Einzufügendes Element
checkDups	ja	boolean	Bei true werden die Änderungen sofort ausgeführt und nicht gecached.

Beschreibung

Diese Methode fügt Einträge der Sammlung hinzu.

deleteEntry

Deklaration

void deleteEntry(ViewEntry entry) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
entry	nein	ViewEntry	Der zu löschende Eintrag

Beschreibung

Durch den Aufruf dieser Methode wird der Eintrag gelöscht.

FTSearch

Deklaration

void FTSearch(String query) throws NotesException

void FTSearch(String query, int maxdocs) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
query	nein	String	Suchformel
maxdocs	ja	int	Die maximale Anzahl der Dokumente, die zurückgeliefert werden sollen

Beschreibung

Diese Methode durchsucht die Sammlung und wirft die nicht dem Suchkriterium entsprechenden Dokumente aus der Sammlung.

getEntry**Deklaration**

`ViewEntry getEntry(Object entry) throws NotesException`

Rückgabewert

`ViewEntry`
das gefundene Element

Parameter

Name	Optional	Typ	Beschreibung
Entry	nein	<code>ViewEntry</code> <code>Document</code>	Zu suchendes Objekt

Beschreibung

Diese Methode liefert das Element, auf dem das angegebene Objekt basiert.

getFirstEntry**Deklaration**

`ViewEntry getFirstEntry() throws NotesException`

Rückgabewert

`ViewEntry`
das erste Element in der Sammlung

Parameter

keine

Beschreibung

Diese Methode liefert den ersten Eintrag aus der Sammlung.

getLastEntry**Deklaration**

`ViewEntry getLastEntry() throws NotesException`

Rückgabewert

`ViewEntry`
das letzte Element der Sammlung

Parameter

keine

Beschreibung

Diese Methode liefert das letzte Element der Sammlung.

getNextEntry

Deklaration

ViewEntry getNextEntry() throws NotesException

ViewEntry getNextEntry(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry

das nachfolgende Element

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen Nachfolger gesucht wird

Beschreibung

Diese Methode liefert entweder den Nachfolger des aktuellen oder des angegebenen Dokuments.

getNthEntry

Deklaration

ViewEntry getNthEntry(int n) throws NotesException

Rückgabewert

ViewEntry

der gefundene Eintrag

Parameter

Name	Optional	Typ	Beschreibung
n	nein	Int	Position des gesuchten Dokuments

Beschreibung

Diese Methode liefert das Dokument, welches an der angegebenen Stelle steht.

getPrevEntry

Deklaration

ViewEntry getPrevEntry() throws NotesException

ViewEntry getPrevEntry(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry
der gefundene Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen Vorgänger gesucht wird.

Beschreibung

Diese Methode liefert entweder den Vorgänger des aktuellen oder des angegebenen Dokuments.

putAllInFolder

Deklaration

void putAllInFolder(String folderName) throws NotesException

void putAllInFolder(String folderName, boolean createOnFail) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
folderName	nein	String	Name des Ordners
createOnFail	ja	boolean	Im Fehlerfall Erstellung eines Ordners

Beschreibung

Diese Methode schiebt alle Einträge in den angegebenen Ordner. Sollte der Ordner nicht existieren, wird er ggf. erstellt.

removeAll

Deklaration

void removeAll(boolean force) throws NotesException

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
force	nein	boolean	Erzwingen des Löschens, auch wenn das Dokument gerade geöffnet ist

Beschreibung

Mit dieser Methode werden alle Dokumente der Sammlung aus der Datenbank entfernt.

removeAllFromFolder

Deklaration

`void removeAllFromFolder(String folderName) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
folderName	nein	String	Ordnername

Beschreibung

Diese Methode entfernt alle Dokumente der Sammlung aus dem angegebenen Ordner.

stampAll

Deklaration

`void stampAll(String itemName, Object value) throws NotesException`

Rückgabewert

kein

Parameter

Name	Optional	Typ	Beschreibung
itemName	nein	String	Name des Felds
value	nein	Object	Wert, der geschrieben werden soll

Beschreibung

Diese Methode schreibt bei allen Dokumenten der Sammlung den angegebenen Wert in das angegebene Feld.

updateAll**Deklaration**

void updateAll() throws NotesException

Rückgabewert

kein

Parameter

keine

Beschreibung

Durch diese Methode werden alle Dokumente der Sammlung als geändert markiert.

9.34 Klasse ViewNavigator

9.34.1 Eigenschaften

CacheSize**Datentyp**

int

Deklaration Getter

int getCacheSize() throws NotesException

Deklaration Setter

void setCacheSize(int numEntries) throws NotesException

Wertebereich

0 (no cache) - 128 (default)

Beschreibung

In dieser Eigenschaft befindet sich die Größe des Caches.

Count

Datentyp

int

Deklaration Getter

int getCount() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Der hier gefundene Wert ist die Anzahl der Einträge, die vom Navigator verwaltet werden.

MaxLevel

Datentyp

int

Deklaration Getter

int getMaxLevel() throws NotesException

Deklaration Setter

void setMaxLevel(int mLevel) throws NotesException

Wertebereich

0 (top level) - 30 (default)

Beschreibung

Der maximale Level des Navigators befindet sich in dieser Eigenschaft.

ParentView

Datentyp

View

Deklaration Getter

View getParentView() throws NotesException

Deklaration Setter

kein

Wertebereich

keine Einschränkung

Beschreibung

Die Eigenschaft gibt die Ansicht an, von der der Navigator abstammt.

9.34.2 Methoden

getChild

Deklaration

ViewEntry getChild() throws NotesException

ViewEntry getChild(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry

Der gesuchte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen erster untergeordneter Eintrag ermittelt werden soll

Beschreibung

Diese Methode liefert den ersten untergeordneten Eintrag des aktuellen oder angegebenen Eintrags zurück.

getCurrent

Deklaration

ViewEntry getCurrent() throws NotesException

Rückgabewert

ViewEntry

der aktuelle Eintrag

Parameter

keine

Beschreibung

Diese Methode gibt den aktuellen Eintrag zurück.

getFirst**Deklaration**

ViewEntry getFirst() throws NotesException

Rückgabewert

ViewEntry
der erste Eintrag

Parameter

keine

Beschreibung

Diese Methode liefert den ersten Eintrag im Navigator.

getFirstDocument**Deklaration**

ViewEntry getFirstDocument() throws NotesException

Rückgabewert

ViewEntry
das erste Dokument

Parameter

keine

Beschreibung

Diese Methode liefert das erste Dokument zurück.

getLast**Deklaration**

ViewEntry getLast() throws NotesException

Rückgabewert

ViewEntry
der letzte Eintrag

Parameter

keine

Beschreibung

Diese Methode liefert den letzten Eintrag zurück.

getLastDocument

Deklaration

ViewEntry getLastDocument() throws NotesException

Rückgabewert

ViewEntry
das letzte Dokument

Parameter

keine

Beschreibung

Diese Methode liefert das letzte Dokument zurück.

getNext

Deklaration

ViewEntry getNext() throws NotesException

ViewEntry getNext(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry
der gesuchte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen Nachfolger ermittelt werden soll

Beschreibung

Diese Methode liefert den Eintrag, der dem aktuellen oder dem angegebenen Eintrag folgt.

getNextCategory

Deklaration

ViewEntry getNextCategory() throws NotesException

Rückgabewert

ViewEntry
der nächste Kategorieeintrag

Parameter

keine

Beschreibung

Diese Methode liefert den nächsten Kategorieeintrag zurück.

getNextDocument

Deklaration

ViewEntry getNextDocument() throws NotesException

Rückgabewert

ViewEntry
das nächste Dokument

Parameter

keine

Beschreibung

Diese Methode liefert das nächste dem aktuellen Eintrag folgende Dokument zurück.

getNextSibling

Deklaration

ViewEntry getNextSibling() throws NotesException
ViewEntry getNextSibling(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry
der gesuchte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen nachfolgender Eintrag ermittelt werden soll

Beschreibung

Diese Methode liefert den Eintrag, der dem aktuellen oder dem angegebenen Eintrag folgt und sich auf derselben Ebene befindet.

getNth

Deklaration

ViewEntry getNth(int n) throws NotesException

Rückgabewert

ViewEntry

Parameter

Name	Optional	Typ	Beschreibung
n	nein	int	Position des Eintrags, der ermittelt werden soll

Beschreibung

Liefert den Eintrag zurück, der an der angegebenen Position steht.

getParent

Deklaration

ViewEntry getParent() throws NotesException

ViewEntry getParent(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry
der übergeordnete Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen übergeordneter Eintrag ermittelt werden soll

Beschreibung

Diese Methode liefert den dem aktuellen oder dem angegebenen Eintrag übergeordneten Eintrag zurück.

getPos

Deklaration

ViewEntry getPos(String pos, char separator) throws NotesException

Rückgabewert

ViewEntry

Parameter

Name	Optional	Typ	Beschreibung
post	nein	String	Die hierarchische Position des gesuchten Eintrags
separator	nein	char	Das Trennzeichen in der Positionsangabe

Beschreibung

Diese Methode liefert das Dokument, welches an der angegebenen hierarchischen Position steht.

getPrev

Deklaration

ViewEntry getPrev() throws NotesException

ViewEntry getPrev(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry

der gesuchte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
Entry	ja	ViewEntry	Der Eintrag, dessen Vorgänger ermittelt werden soll

Beschreibung

Diese Methode liefert den Vorgänger des aktuellen oder des angegebenen Eintrags zurück.

getPrevCategory

Deklaration

ViewEntry getPrevCategory() throws NotesException

Rückgabewert

ViewEntry
der vorherige Kategorieeintrag

Parameter

keine

Beschreibung

Diese Methode liefert den vorhergehenden Kategorieeintrag zurück.

getPrevDocument

Deklaration

ViewEntry getPrevDocument() throws NotesException

Rückgabewert

ViewEntry
das gesuchte Dokument

Parameter

keine

Beschreibung

Diese Methode liefert das Vorgängerdokument des aktuellen Dokuments zurück.

getPrevSibling

Deklaration

ViewEntry getPrevSibling() throws NotesException
ViewEntry getPrevSibling(ViewEntry entry) throws NotesException

Rückgabewert

ViewEntry
der gesuchte Eintrag

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Der Eintrag, dessen Vorgänger ermittelt werden soll

Beschreibung

Diese Methode liefert den Eintrag, der dem aktuellen oder dem angegebenen Eintrag vorausgeht.

gotoChild

Deklaration

boolean gotoChild() throws NotesException

boolean gotoChild(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem geforderten Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen erstem untergeordneten Eintrag gesprungen werden soll

Beschreibung

Diese Methode springt zu dem ersten untergeordneten Eintrag des aktuellen oder des angegebenen Eintrags.

gotoEntry

Deklaration

boolean gotoEntry(Object entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem geforderten Eintrag gesprungen werden konnte

Parameter

Object entry

Beschreibung

Mit dieser Methode bewegt man sich zu dem gewünschten Eintrag.

gotoFirst

Deklaration

boolean gotoFirst() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Nach dem Aufruf dieser Methode ist der erste Eintrag des Navigators der neue aktuelle Eintrag.

gotoFirstDocument

Deklaration

boolean gotoFirstDocument() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Mit dieser Methode springt man zu dem ersten Dokument im Navigator.

gotoLast

Deklaration

boolean gotoLast() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Durch diese Methode wird zum letzten Eintrag im Navigator gesprungen.

gotoLastDocument

Deklaration

boolean gotoLastDocument() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Diese Methode springt zum letzten Dokument im Navigator.

gotoNext

Deklaration

boolean gotoNext() throws NotesException

boolean gotoNext(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen Nachfolger gesprungen werden soll

Beschreibung

Diese Methode springt zum Nachfolger des aktuellen oder des angegebenen Eintrags.

gotoNextCategory

Deklaration

boolean gotoNextCategory() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Diese Methode springt zum nächsten Kategorieeintrag.

gotoNextDocument

Deklaration

boolean gotoNextDocument() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Diese Methode springt zum nächsten Dokument.

gotoNextSibling

Deklaration

boolean gotoNextSibling() throws NotesException

boolean gotoNextSibling(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen Nachfolger gesprungen werden soll

Beschreibung

Diese Methode springt zum Nachfolger des aktuellen oder des angegebenen Eintrags auf derselben Ebene.

gotoParent

Deklaration

boolean gotoParent() throws NotesException

boolean gotoParent(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen übergeordnetem Eintrag gesprungen werden soll

Beschreibung

Mit dieser Methode wird zu dem übergeordneten Eintrag des aktuellen oder des angegebenen Eintrags gesprungen.

gotoPos

Deklaration

boolean gotoPos(String pos, char separator) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
pos	nein	String	Position, zu der gesprungen werden soll
separator	nein	char	Trennzeichen bei der Positionsangabe

Beschreibung

Mit dieser Methode wird an die angegebene Position gesprungen.

gotoPrev

Deklaration

boolean gotoPrev() throws NotesException

boolean gotoPrev(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen Vorgänger gesprungen werden soll

Beschreibung

Diese Methode springt zu dem Vorgängereintrag des aktuellen oder des angegebenen Eintrags.

gotoPrevCategory

Deklaration

boolean gotoPrevCategory() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Diese Methode springt zum vorhergehenden Kategorieeintrag.

gotoPrevDocument

Deklaration

boolean gotoPrevDocument() throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

keine

Beschreibung

Diese Methode springt zu dem vorhergehenden Dokument.

gotoPrevSibling

Deklaration

boolean gotoPrevSibling() throws NotesException

boolean gotoPrevSibling(ViewEntry entry) throws NotesException

Rückgabewert

boolean

liefert true, wenn zu dem angegebenen Eintrag gesprungen werden konnte

Parameter

Name	Optional	Typ	Beschreibung
entry	ja	ViewEntry	Eintrag, zu dessen Vorgänger gesprungen werden soll

Beschreibung

Diese Methode springt zum Vorgängereintrag auf gleicher Ebene des aktuellen oder des angegebenen Eintrags.

10 Referenz JSP

10.1 Referenz der JSP Domino Tags

10.1.1 action

```
<domino:action
  name="action name"
  enabled="true|false"
  showdisabled="true|false"
  text="link text"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML attributes listed below
>
```

10.1.2 attachmentlist

```
<domino:attachmentlist
  format="formatting keywords"
  enableddelete="true|false"
  enableconfirm="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML attributes listed below>
```

10.1.3 attachments

```
<domino:attachments
  name="control name"
  bgcolor="color of the control"
  width="control width"
  height="control height"
  maxsize="maximum kb size for files"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.4 authors

```
<domino:authors
  id="name of object variable"
  format="formatting keywords"
  locale="<languageCode>_<countryCode>"
>
```

```

        debug="true|false"
        onfailure="inline|exception"
        time="true|false"
    >

```

10.1.5 checkbox

```

<domino:checkbox
    name="name of the control"
    itemname="name of the item in the document"
    datatype="text|number|date|datetime|readers|authors|names"
    multivalue="true|false"
    default="formula or simple value"
    label="text to display on control"
    labelkey="resource string"
    labelbefore="true|false"
    bundle="name of .properties file bundle"
    locale="<languageCode>_<countryCode>"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    value="HTML attribute that represents the checked state of a checkbox"
    The standard HTML tag attributes listed below
>

```

10.1.6 created

```

<domino:created
    id="name of object variable"
    format="formatting keywords"
    locale="<languageCode>_<countryCode>"
    timezone="time zone"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>

```

10.1.7 db

```

<domino:db
    id="name of object variable"
    dbname="database name"
    dbserver="server name"
    user="login information"
    password="password associated with user"
    host="CORBA server name"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>

```


10.1.8 dbfill

```
<domino:dbfill
  target="input control to be filled with value"
  dbfield="field in document that value is extracted from"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.9 dbselect

```
<domino:dbselect
  name="web control name"
  itemname="name of the item in the document"
  view="database view to retrieve data from"
  query="search query string"
  key="lookup value to search for in a sorted column in the view"
  max="number"
  dbserver="server database resides on"
  dbname="database name"
  user="login information"
  password="password associated with user"
  category="category name"
  displayitem="item to use to display an option tag"
  valueitem="item to use to represent the value of an option tag"
  host="CORBA server name"
  format="options display format"
  formula="Notes formula"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
>
```

10.1.10 default

```
<domino:default
  scope="application|session|request|page"
  dbname="database name"
  dbserver="server name"
  user="log-on information"
  password="password associated with user"
  viewname="name of the view"
  unid="universal ID of the document"
  responseto="UNID of the parent document"
  host="remote server name"
  bundle="name of .properties file bundle"
  timezone="time zone"
  locale="<languageCode>_<countryCode>"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.11 deleteattachment

```
<domino:deleteattachment
    text="button label"
    showdisabled="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.12 deletedoc

```
<domino:deletedoc
    text="button label"
    showdisabled="true|false"
    validhref="link to go to"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.13 docloop

```
<domino:docloop
    id="name of object variable"
    count="number"
    start="number"
    direction="ASC|DSC|Ascending|Descending"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.14 docnavig

```
<domino:docnavig
    type="S|N|SC|D|DA|E|P|W"
    showdisabled="true|false"
    alttext="text string"
    enabled="image file"
    disabled="image file"
    asresponse="true|false"
    border="number"
    validhref="url"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.15 document

```
<domino:document
  id="name of object variable"
  dbname="database name"
  dbserver="server name"
  host="CORBA server name"
  user="login information"
  password="password associated with user"
  unid="universal id for document to edit"
  responseto="unid of parent document"
  schema="Domino form name for new document schema"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.16 editdoc

```
<domino:editdoc
  text="button label"
  showdisabled="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
>
```

10.1.17 fill

```
<domino:fill
  target="input control to be filled with value"
  value="value to fill in input target"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.18 form

```
<domino:form
  id="name of object variable"
  name="name of the form"
  clientvalidate="true|false"
  counter="parameter name"
  unid="unique id of the document to be edited/displayed/responded to"
  editmode="display|edit|delete"
  dbname="name of the database to use"
  dbserver="name of the server that contains dbname"
  user="login information"
  password="password associated with user"
  viewname="name of the view to use for navigation"
  validhref="link to go to if the form is validated and document is saved"
```

```

    responseto="true|false"
    schema="Domino form name"
    host="CORBA server name"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below

```

>

10.1.19 formlink

```

<domino:formlink
    href="name of page with form or document tag"
    asresponse="true|false"
    text="hypertext link content"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below

```

>

10.1.20 formula

```

<domino:formula
    id="object variable name"
    format="formatting keywords"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"

```

>

10.1.21 ftsearch

```

<domino:ftsearch
    id="name of object variable"
    dbname="database name"
    dbserver="server name"
    user="session information"
    password="password associated with user"
    query="search string"
    max="maximum number of documents to return"
    sort="dateascending|datedescending|score"
    options="database|filestytem|fuzzy|stems"
    host="CORBA server name"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"

```

>

10.1.22 ifauthor

```
<domino:ifauthor
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.23 ifcapability

```
<domino:ifcapability
  is="true|false"
  tag="name of the tag to test"
  taglibversion="tag library version number to test"
  dominoversion="domino version number to test"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.24 ifcategoryentry

```
<domino:ifcategoryentry
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.25 ifconflictenry

```
<domino:ifconflictenry
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.26 ifdbrole

```
<domino:ifdbrole
  is="true|false"
  name="role name"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.27 ifdepositor

```
<domino:ifdepositor
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.28 ifdesigner

```
<domino:ifdesigner
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.29 ifdisplaymode

```
<domino:ifdisplaymode
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.30 ifdocauthor

```
<domino:ifdocauthor
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.31 ifdocreader

```
<domino:ifdocreader
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.32 ifdocumententry

```
<domino:ifdocumententry
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.33 ifeditmode

```
<domino:ifeditmode
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.34 ifeditor

```
<domino:ifeditor
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.35 ifmanager

```
<domino:ifmanager
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.36 ifnoaccess

```
<domino:ifnoaccess
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.37 ifreader

```
<domino:ifreader
  is="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.38 ifserverevent

```
<domino:ifserverevent
  id="name of object variable"
  event="QuerySave|QueryClose|OnAction"
  for="name of the action initiator"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.39 iftotalentry

```
<domino:iftotalentry
    is="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.40 indent

```
<domino:indent
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.41 input

```
<domino:input
    name="control name"
    format="format for displaying"
    editformat="format for editing"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    itemname="Domino item name"
    datatype="text|number|date|datetime|readers|authors|names"
    clientvalidate="onblur|onsubmit|none"
    defaultformula="formula determining default value"
    defaultvalue="simple value to use as default value"
    multivalue="true|false"
    mveditseparator="blank|comma|colon|semicolon|newline"
    mvdisplayseparator="blank|comma|semicolon|newline"
    value=""
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.42 item

```
<domino:item
    id="object variable name"
    name="item name"
    format="formatting keywords"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```


10.1.43 lastaccessed

```
<domino:lastaccessed
  id="name of object variable"
  format="formatting keywords"
  timezone="time zone"
  locale="<languageCode>_<countryCode>"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.44 lastmodified

```
<domino:lastmodified
  id="name of object variable"
  format="formatting keywords"
  timezone="time zone"
  locale="<languageCode>_<countryCode>"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.45 mailto

```
<domino:mailto
  to="names"
  cc="names"
  bcc="names"
  subject="subject of message"
  returnreceipt="true|false"
  importance="low|normal|high"
  priority="low|normal|high"
  deliveryreport="none|onlyonfailure|confirm|tracepath"
  encoding="text/html|text/plain"
  saveonsend="true|false"
  dbname="database name"
  dbserver="server name"
  host="CORBA server name"
  user="login information"
  password="password associated with user"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.46 msg

```
<domino:msg
  id="name of object variable"
  msgkey="key id in the bundle"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
>
```

```

    arg0="1st replacement argument"
    arg1="2nd replacement argument"
    arg2="3rd replacement argument"
    arg3="4th replacement argument"
    arg4="5th replacement argument"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>

```

10.1.47 newdoc

```

<domino:newdoc
    showdisabled="true|false"
    asresponse="true|false"
    text="button label"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>

```

10.1.48 nextdoc

```

<domino:nextdoc
    text="button label"
    showdisabled="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>

```

10.1.49 nodocument

```

<domino:nodocument
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>

```

10.1.50 novalues

```

<domino:novales
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>

```

10.1.51 option

```

<domino:option
    debug="true|false"

```

```
onfailure="inline|exception"
time="true|false"
The standard HTML tag attributes listed below
```

>

10.1.52 page

```
<domino:page
  id="name of object variable"
  start="page number to start at"
  rows="number of entries rendered per page"
  direction="ASC|DSC|Ascending|Descending"
  max="maximum number of pages to display"
  evaluateonnorows="true|false"
  viewasdocuments="true|false"
  name="name of page parameter storing current page number"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
```

>

10.1.53 pagebar

```
<domino:pagebar
  pages="number of hotspots"
  navstyle="None|AllBlocks|NextPrev|OnlyBlocks|OnlyPages"
  nextblocking="image file"
  prevblocking="image file"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
```

>

10.1.54 pagefirst

```
<domino:pagefirst
  text="hypertext link content"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
```

>

10.1.55 pagelast

```
<domino:pagelast
  text="hypertext link content"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
```

>

10.1.56 pagenext

```
<domino:pagenext
  showdisabled="true|false"
  text="hypertext link content"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
>
```

10.1.57 pageprev

```
<domino:pageprev
  showdisabled="true|false"
  text="hypertext link content"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
>
```

10.1.58 parentlink

```
<domino:parentlink
  href="name of page with parent form or document tag"
  showdisabled="true|false"
  text="hypertext link content"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The standard HTML tag attributes listed below
>
```

10.1.59 preserve

```
<domino:preserve
  name="page parameter"
  value="value to override the specified page parameter value with"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.60 preset

```
<domino:preset
  scope="application|session|request|page"
  dbname="database name"
  dbserver="server name"
  user="login information"
  password="password associated with user"
  viewname="name of the view"
```

```

    unid="Universal ID of the document"
    responseto="parent document's unid"
    host="name of CORBA server"
    bundle="name of .properties file bundle"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    onfailure="inline|exception"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"

```

>

10.1.61 prevdoc

```

<domino:prevdoc
    text="button label"
    showdisabled="true|false"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below

```

>

10.1.62 profiledocument

```

<domino:profiledocument
    profilename="name"
    profileuser="user name"
    id="id number"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"

```

>

10.1.63 radio

```

<domino:radio
    name="control name"
    itemname="item name in document"
    datatype="text|number|date|datetime|readers|authors|names"
    default="true|false"
    labelkey="resource string"
    labelbefore="true|false"
    clientvalidate="onblur|onsubmit|none"
    bundle="name of .properties file bundle"
    locale="<languageCode>_<countryCode>"
    value="HTML attribute indicating value of a selected radio button"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below

```

>

10.1.64 response

```
<domino:responses
    id="name of object variable"
    max="number"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.65 richtext

```
<domino:richtext
    name="name of the Web control"
    itemname="name of the item in a document"
    control="ie5|applet|check"
    locale="<languageCode>_<countryCode>"
    clientvalidate="onblur|onsubmit|none"
    width="percentage of window height"
    height="pixels"
    rows="height if running remotely"
    cols="width if running remotely"
    codepath="path to the editing files"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.66 runagent

```
<domino:runagent
    name="agent name"
    dbname="database name"
    dbserver="server name"
    user="login information"
    password="password for user"
    host="name of CORBA server"
    unid="universal id of document to pass to agent"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.67 saveclosedoc

```
<domino:saveclosedoc
    showdisabled="true|false"
    text="button label"
    validhref="uri"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>
```

10.1.68 savedoc

```
<domino:savedoc
    showdisabled="true|false"
    text="button label"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>
```

10.1.69 savenow

```
<domino:savenow
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.70 select

```
<domino:select
    name="control name"
    itemname="item name"
    format="display format of options"
    defaultformula="item default value derived from a formula"
    defaultvalue="item default value derived from a simple value"
    multivalued="true|false"
    mvdiseparator="blank|comma|colon|semicolon|newline"
    mveditseparator="blank|comma|colon|semicolon|newline"
    datatype=""
    locale="<languageCode>_<countryCode>"
    timezone="time zone"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```

10.1.71 selectedloop

```
<domino:selectedloop
    id="name of object variable"
    dbname="database name"
    dbserver="name of server hosting the database"
    user="log-in information"
    password="password associated with specified user"
    host="CORBA server name"
    viewname="view containing selected list of documents"
    start="object to begin display at"
    count="number of objects to display"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.72 selectentry

```
<domino:selectentry
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.73 session

```
<domino:session
    id="name of object variable"
    user="user name"
    password="user password"
    host="CORBA server name"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.74 setitem

```
<domino:setitem
    name="name of the item in the document"
    value="value to assign to item"
    datatype="text|number|date|datetime|richtext|readers|authors|names"
    mvseparator="blank|comma|colon|semicolon|newline|,|;| |:"
    locale="<languageCode>_<countryCode>"
    timezone="time zone"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.75 textarea

```
<domino:textarea
    name="name of the Web control"
    itemname="name of the item in a document"
    datatype="text|number|date|datetime|readers|authors|names|richtext"
    defaultformula="item default value derived from a formula"
    defaultvalue="item default value derived from a simple value"
    editformat="format for editing"
    format="format for displaying"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    mulivalue="true|false"
    mveditseparator="blank|comma|colon|semicolon|newline"
    mvdisplayseparator="blank|comma|colon|semicolon|newline"
    clientvalidate="onblur|onsubmit|none"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The standard HTML tag attributes listed below
>
```


10.1.76 **unid**

```
<domino:unid
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.77 **user**

```
<domino:user
  id="name of object variable"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.1.78 **validatecc**

```
<domino:validatecc
  name="input tag name"
  message="message to display if validation fails"
  msgkey="message bundle key"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
  alert="true|false"
  onalert="alert dialog box message"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The HTML tag attributes listed below
>
```

10.1.79 **validatedate**

```
<domino:validatedate
  name="input tag name"
  max="maximum date"
  min="minimum date"
  type="DATE|DATETIME"
  message="message to display if validation fails"
  msgkey="message bundle key"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
  alert="true|false"
  onalert="alert dialog box message"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The HTML tag attributes listed below
>
```

10.1.80 validateLength

```
<domino:validateLength
    name="input tag name"
    max="maximum length"
    min="minimum length"
    message="message to display if validation fails"
    msgkey="message bundle key"
    bundle="name of .properties file bundle"
    locale="<languageCode>_<countryCode>"
    alert="true|false"
    onalert="alert dialog box message"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>
```

10.1.81 validatePattern

```
<domino:validatePattern
    name="input tag name"
    pattern="valid pattern for matching values"
    message="message to display if validation fails"
    msgkey="message bundle key"
    bundle="name of .properties file bundle"
    locale="<languageCode>_<countryCode>"
    alert="true|false"
    onalert="alert dialog box message"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>
```

10.1.82 validateRange

```
<domino:validateRange
    name="input tag name"
    max="maximum value"
    min="minimum value"
    type="string|number"
    message="message to display if validation fails"
    msgkey="message bundle key"
    bundle="name of .properties file bundle"
    locale="<languageCode>_<countryCode>"
    alert="true|false"
    onalert="alert box message"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
    The HTML tag attributes listed below
>
```

10.1.83 validaterequired

```
<domino:validaterequired
  name="input tag name"
  message="message to display if validation fails"
  msgkey="message bundle key"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
  alert="true|false"
  onalert="alert dialog box message"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The HTML tag attributes listed below
>
```

10.1.84 validatesummary

```
<domino:validatesummary
  caption="error header"
  captionkey="key id for message"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
  format="formatting keywords"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The HTML tag attributes listed below
>
```

10.1.85 validatewith

```
<domino:validatewith
  name="input tag name"
  value="comparison value"
  otherfield="name of the other input tag to compare to"
  operation="EQ|NE|LT|GT|LE|GE|=|<|>|<=|>="
  type="string|number"
  casesensitive="true|false"
  message="message to display if validation fails"
  msgkey="message bundle key"
  bundle="name of .properties file bundle"
  locale="<languageCode>_<countryCode>"
  alert="true|false"
  onalert="alert dialog box message"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
  The HTML tag attributes listed below
>
```

10.1.86 view

```
<domino:view
    id="name of object variable"
    dbname="database name"
    dbserver="database server name"
    viewname="view name"
    defaultname="default view name"
    key="key"
    keyseparator="blank|comma|colon|semicolon|newline| |,:|;"
    keytype="string|number|date|daterange"
    keyexact="true|false"
    ftsearch="search string"
    max="maximum number of entries to return"
    toponly="true|false"
    depth="number of level to process to"
    category="category that all documents returned must fall under"
    user="login information for the session"
    password="password associated with user name"
    enableselect="true|false"
    host="CORBA server name"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.87 viewitem

```
<domino:viewitem
    id="name of object variable to store"
    format="formatting keywords"
    col="number of column containing item"
    name="name of item to display"
    timezone="time zone"
    locale="<languageCode>_<countryCode>"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.1.88 viewloop

```
<domino:viewloop
    id="name of object variable"
    count="total to include in loop"
    start="offset of first element"
    direction="ASC|DSC|Ascending|Descending"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.2 Referenz der JSP Domutil

10.2.1 browsertag

```
<domutil:browsertag
  id="page variable name"
  useragent="HTTP user agent string"
  supports="capability"
  value="value of passed capability"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.2 case

```
<domutil:case
  value="string to evaluate"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.3 condition

```
<domutil:condition
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.4 default

```
<domutil:default
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.5 else

```
<domutil:else
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.6 elseif

```
<domutil:elseif
    condition="true|false|0|1|yes|no"
    isblank="parameter name | * for all parameters"
    isnull="parameter name"
    browser="Explorer|Netscape"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.2.7 format

```
<domutil:format
    id="variable name"
    format="formatting style to apply to string"
    locale="<languageCode>_<countryCode>"
    timezone="time zone"
    value="value to format"
    type="string|date|double|integer|javadata"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.2.8 if

```
<domutil:if
    condition="true|false|0|1|yes|no"
    isblank="parameter name | * for all parameters"
    isnull="parameter name"
    inrole="role name"
    trueaction="exclude|include"
    browser="Explorer|Netscape"
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.2.9 nocache

```
<domutil:nocache
    debug="true|false"
    onfailure="inline|exception"
    time="true|false"
>
```

10.2.10 switch

```
<domutil:switch
  value="string being matched"
  ignorecase="true|false"
  autotrim="true|false"
  partialmatch="true|false"
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.11 switchvalue

```
<domutil:switchvalue
  debug="true|false"
  onfailure="inline|exception"
  time="true|false"
>
```

10.2.12 urlconverter

```
<domutil:urlconverter
  id="variable name"
  value="URL string to decode"
  charset="character set to use for decoding"
  onfailure="inline|exception"
  time="true|false"
>
```


11

CGI-Variablen

Feldname	Rückgabewerte
Auth_Type	Wenn der Server die Benutzer-Authentifizierung unterstützt und das Script geschützt ist, ist dies die protokollspezifische Authentifizierungsmethode zur Überprüfung des Benutzers.
Content_Length	Die vom Client vorgegebene Länge des Inhalts.
Content_Type	Für Abfragen mit angehängten Informationen, wie etwa HTTP POST und PUT, ist dies der Inhaltstyp der Daten.
Gateway_Interface	Die Version der CGI-Spezifikation, nach der der Server sich richtet.
HTTP_Accept	Die MIME-Typen, die der Client akzeptiert, wie im HTTP-Header angegeben.
HTTP_Accept_language	Die Sprachen, die der Client akzeptiert, wie im HTTP-Header angegeben.
HTTP_REFERER	Die URL der Seite, die der Benutzer verwendet, um an diese Stelle zu gelangen.
HTTPS	Gibt an, ob der SSL-Modus für den Server aktiviert ist.
HTTPS_CLIENT_CERT_COMMON_NAME	Der allgemeine Name im X.509-Zertifikat.
HTTPS_CLIENT_CERT_ISSUER_COMMON_NAME	Der Aussteller des X.509-Zertifikats.
HTTPS_KEYSIZE	Der Sitzungsschlüssel während einer SSL-Sitzung, z. B. 40-Bit, 128-Bit.
HTTP_User_Agent	Der Browser, den der Client verwendet, um die Anfrage zu senden.
Path_Info	Die zusätzlichen Pfad-Informationen (aus dem HTML-Stammverzeichnis des Servers), wie vom Client vorgegeben. Das heißt, auf Scripts kann über deren virtuellen Pfadnamen zugegriffen werden, gefolgt von zusätzlichen Informationen, die als PATH_INFO gesendet werden.

Feldname	Rückgabewerte
Path_Info_Decoded	Gibt dieselben Informationen wie Path_Info zurück, dekodiert die Zeichenfolge jedoch. Wenn sich beispielsweise eine URL auf einen Ansichtsnamen bezieht, der Zeichen enthält, die in einer URL nicht erlaubt sind, wird der Name kodiert. Diese CGI-Variable dekodiert die Zeichenfolge. Path_Info_Decoded ist nur für Domino Anwendungen verfügbar.
Path_Translated	Der Server stellt eine übersetzte Version von PATH_INFO zur Verfügung, die den virtuellen Angaben physische Pfaddaten zuordnet.
Query_String	Die Informationen nach dem Fragezeichen (?) in der URL, die auf dieses Script verweist. Hinweis: Wenn der Domino Server so konfiguriert ist, dass er es Suchmaschinen erlaubt, Ihre Website zu durchsuchen, generiert Domino URLs mit einem Ausrufezeichen (!) anstelle eines Fragezeichens (?). Wenn dies der Fall ist, enthält die CGI-Variable "Query_String" die Informationen hinter dem Ausrufezeichen (!). Domino erkennt immer sowohl das Fragezeichen (?) als auch das Ausrufezeichen (!), generiert aber nur URLs beim Ausrufezeichen (!), wenn Ihre Website für Web-Suchmaschinen zur Verfügung steht. Durch das Generieren von URLs mit einem Ausrufezeichen (!) kann die Website effizienter durchsucht werden.
Query_String_Decoded	Gibt dieselben Informationen wie Query_String zurück, dekodiert die Zeichenfolge jedoch. Wenn beispielsweise eine URL auf einen Ansichtsnamen verweist, der Zeichen enthält, die in einer URL nicht erlaubt sind, wird der Name kodiert. Diese CGI-Variable dekodiert diese Zeichenfolge. Path_Info_Decoded ist nur für Domino Anwendungen verfügbar.
Remote_Addr	Die IP-Adresse des Remote-Hosts, der die Anfrage stellt.
Remote_Host	Der Name des Hosts, der die Anfrage stellt.
Remote_Ident	Diese Variable wird für den Namen des Remote-Benutzers vergeben, der vom Server abgerufen wurde. Verwenden Sie diese Variable nur zum Protokollieren.

Feldname	Rückgabewerte
Remote_User	Authentifizierungsmethode, die den authentifizierten Benutzernamen zurückgibt.
Request_Content	Nur für Agenten unterstützt. Enthält die Daten, die mit einer HTTP POST-Anforderung gesendet werden. Die Daten sind in der Regel "URLencoded" (URL-verschlüsselt) und bestehen aus "Name=Wert"-Paaren, die durch Et-Zeichen aneinander gehängt werden. Beispiel: Vorname=Julia&Nachname=Meier
Request_Method	Die für die Anfrage verwendete Methode. Für HTTP sind dies "GET", "HEAD", "POST" usw.
Script_Name	Ein virtueller Pfad auf das ausgeführte Script; wird für URLs, die auf sich selbst verweisen, verwendet.
Server_Name	Der Host-Name des Servers, der DNS-Aliasname oder die IP-Adresse, wie sie in URLs, die auf sich selbst verweisen, erscheinen würden.
Server_Protocol	Der Name und die Überarbeitung des Informationsprotokolls, das zu dieser Anfrage gehört.
Server_Port	Der Anschluss, an den die Anfrage gesendet wurde.
Server_Software	Der Name und die Version der Informationsserver-Software, die das CGI-Programm ausführt.
Server_URL_Gateway_Interface	Die Version der CGI-Spezifikation, nach der der Server sich richtet.

Listingverzeichnis

Listing 2.1:	Beispiel Aufbau eines Javaprogramms	19
Listing 2.2:	Beispiel Kommentarzeilen	19
Listing 2.3:	Beispiel Kommentare	20
Listing 2.4:	Beispiel für die if-Anweisung	23
Listing 2.5:	Beispiel für die switch-Anweisung	24
Listing 2.6:	Beispiel für die for-Anweisung	24
Listing 2.7:	Beispiel für die do-Anweisung	25
Listing 2.8:	Beispiel für die while-Anweisung	25
Listing 2.9:	Beispiel einer Zeichenkette	26
Listing 2.10:	Beispiel für eine Klasse	29
Listing 2.11:	Beispiel Konstruktoraufruf	30
Listing 2.12:	Beispiel Kapselung	31
Listing 2.13:	Beispiel Sichtbarkeit	31
Listing 2.14:	Beispiel Vererbung	32
Listing 2.15:	Beispiel Überladung	33
Listing 2.16:	Beispiel Abstrakte Klassen und Methoden	33
Listing 2.17:	Beispiel Statische Variablen	34
Listing 2.18:	Beispiel Statische Methoden	34
Listing 2.19:	Beispiel Schlüsselwort final	35
Listing 2.20:	Beispiel Schnittstelle	36
Listing 2.21:	Beispiel Ausnahme	37
Listing 2.22:	Beispiel finally	38
Listing 2.23:	Beispiel für das Werfen einer Exception	38
Listing 4.1:	Beispiel Java-Agent	46
Listing 4.2:	Beispiel für die Programmierung der Agent-Eigenschaften	47
Listing 4.3:	Beispiel Agent-Parameterdokument	48
Listing 4.4:	Beispiel runOnServer	48
Listing 4.5:	Beispiel Agent-Datenspeicherung	49
Listing 4.6:	Beispiel Standalone-Anwendung NotesThread	50
Listing 4.7:	Beispiel Standalone-Anwendung	51
Listing 4.8:	Beispiel Standalone-Anwendung CORBA	53
Listing 4.9:	Beispiel Standard-Applet	54
Listing 4.10:	Beispiel DbDirectory.getFirstDatabase und DbDirectory.getNextDatabase	58
Listing 4.11:	Beispiel Database.open	59
Listing 4.12:	Beispiel Session.getDatabase	60
Listing 4.13:	Beispiel DbDirectory.openDatabase	61
Listing 4.14:	Beispiel DbDirectory.openDatabaseByReplicaID	61
Listing 4.15:	Beispiel DBDirectory.openDatabaseifModified	62
Listing 4.16:	Beispiel AgentContext.getCurrentDatabase	63
Listing 4.17:	Beispiel Session.getAddressBooks	64
Listing 4.18:	Beispiel Database.isPublicAddressBook und Database.isPrivateAddressBook	64
Listing 4.19:	Beispiel DbDirectory.openMailDatabase	65

Listing 4.20:	Beispiel Session.getURLDatabase	66
Listing 4.21:	Beispiel Session.resolve	67
Listing 4.22:	Beispiel Database.isOpen	68
Listing 4.23:	Beispiel DbDirectory.createDatabase	69
Listing 4.24:	Beispiel Database.createFromTemplate	69
Listing 4.25:	Beispiel Database.createCopy	70
Listing 4.26:	Beispiel Database.replicate	71
Listing 4.27:	Beispiel Database.compact	72
Listing 4.28:	Beispiel Database.remove	72
Listing 4.29:	Beispiel Database.getServer	73
Listing 4.30:	Beispiel Database.getFileName und Database.getFilePath	74
Listing 4.31:	Beispiel Database.Title	75
Listing 4.32:	Beispiel Database.Created	76
Listing 4.33:	Beispiel Database.LastModified	76
Listing 4.34:	Beispiel Database.ReplicaID	77
Listing 4.35:	Beispiel Replication.save	78
Listing 4.36:	Beispiel Replication.IgnoreDestDeletes	80
Listing 4.37:	Beispiel Replication.Priority	81
Listing 4.38:	Beispiel Replication.disabled	82
Listing 4.39:	Beispiel Replikation.clearHistory	83
Listing 4.40:	Beispiel Replication.CutoffInterval	84
Listing 4.41:	Beispiel Replication.CutoffDatum	84
Listing 4.42:	Beispiel Replication.CutoffDelete	85
Listing 4.43:	Beispiel Replication.Abstract	86
Listing 4.44:	Beispiel Database.TemplateName	87
Listing 4.45:	Beispiel Database.DesignTemplateName	88
Listing 4.46:	Beispiel Database.Categories	89
Listing 4.47:	Beispiel Database.size	90
Listing 4.48:	Beispiel Database.sizeQuota	91
Listing 4.49:	Beispiel Database.MaxSize	92
Listing 4.50:	Beispiel Database.PercentUsed	93
Listing 4.51:	Beispiel Database.LastFTIndexed und Database.FTIndexed	94
Listing 4.52:	Beispiel Database.MultiDbSearch	95
Listing 4.53:	Beispiel Database.DelayUpdates	96
Listing 4.54:	Beispiel Database.FolderReferencesEnabled	97
Listing 4.55:	Beispiel Database.Managers	98
Listing 4.56:	Beispiel Database.getDocumentByID	99
Listing 4.57:	Beispiel Document.getNoteID	100
Listing 4.58:	Beispiel Database.getDocumentByUNID	101
Listing 4.59:	Beispiel Document.getUniversalID	101
Listing 4.60:	Beispiel Database.AllDocuments	102
Listing 4.61:	Beispiel DocumentCollection.nextDocument	103
Listing 4.62:	Beispiel DocumentCollection.getPrevDocument	104
Listing 4.63:	Beispiel DocumentCollection.getNthDocument	105
Listing 4.64:	Beispiel DocumentCollection.deleteDocument	106
Listing 4.65:	Beispiel DocumentCollection.addDocument	107
Listing 4.66:	Beispiel DocumentCollection.removeAll	107
Listing 4.67:	Beispiel DocumentCollection.putAllInFolder	108

Listing 4.68:	Beispiel DocumentCollection.removeAllFromFolder	109
Listing 4.69:	Beispiel DocumentCollection.stampAll	110
Listing 4.70:	Beispiel DocumentCollection.updateAll	110
Listing 4.71:	Beispiel Database.search	111
Listing 4.72:	Beispiel Database.FTSearch und DocumentCollection.FTSearch	112
Listing 4.73:	Beispiel Database.unprocessedSearch	113
Listing 4.74:	Beispiel Database.createDocument	114
Listing 4.75:	Beispiel Document.save	115
Listing 4.76:	Beispiel Document.remove	116
Listing 4.77:	Beispiel Document.copyToDatabase	117
Listing 4.78:	Beispiel Document.send	118
Listing 4.79:	Beispiel Document.SaveMessageOnSend	119
Listing 4.80:	Beispiel Database.getProfileDocument	120
Listing 4.81:	Beispiel Database.getProfileDocCollection	121
Listing 4.82:	Beispiel Database.getACL	121
Listing 4.83:	Beispiel ACL.getFirstEntry und ACL.getNextEntry	123
Listing 4.84:	Beispiel ACL.getEntry	123
Listing 4.85:	Beispiel ACL.createACLEntry	124
Listing 4.86:	Beispiel ACL.removeACLEntry	125
Listing 4.87:	Beispiel ACLEntry.remove	126
Listing 4.88:	Beispiel ACLEntry.Group, ACLEntry.Server, ACLEntry.Person	128
Listing 4.89:	Beispiel ACLEntry.AdminServer	129
Listing 4.90:	Beispiel ACLEntryt.Level	130
Listing 4.91:	Beispiel ACLEntry.Name	130
Listing 4.92:	Beispiel ACLEntry.CanCreateDocuments	132
Listing 4.93:	Beispiel ACL.getRoles	133
Listing 4.94:	Beispiel ACL.addRole	133
Listing 4.95:	Beispiel ACL.deleteRole	134
Listing 4.96:	Beispiel ACLEntry.getRoles	135
Listing 4.97:	Beispiel ACLEntry.isRoleEnabled	136
Listing 4.98:	Beispiel ACLEntry.enableRole	136
Listing 4.99:	Beispiel ACLEntry.disableRole	137
Listing 4.100:	Beispiel ACL.UniformAccess	138
Listing 4.101:	Beispiel ACL.InternetLevel	139
Listing 4.102:	Beispiel Document.Itmes	140
Listing 4.103:	Beispiel Document.getFirstItem	141
Listing 4.104:	Beispiel Document.hasItem	142
Listing 4.105:	Beispiel Item.DateValue	143
Listing 4.106:	Beispiel Item.getValueInteger	144
Listing 4.107:	Beispiel Item.getValueString	144
Listing 4.108:	Beispiel Item.containsValue	146
Listing 4.109:	Beispiel Document.getItem (RichTextItem)	149
Listing 4.110:	Beispiel Document.createRichTextItem	151
Listing 4.111:	Beispiel RichTextItem.getFormattedText	152
Listing 4.112:	Beispiel RichTextItem.appendStyle	154
Listing 4.113:	Beispiel RichTextStyle	156

Listing 4.114:	Beispiel RichTextParagraphStyle	157
Listing 4.115:	Beispiel RichTextParagraphStyle.setTab	158
Listing 4.116:	Beispiel RichTextParagraphStyle.setTabs	159
Listing 4.117:	Beispiel Session.createDateTime	160
Listing 4.118:	Beispiel DateTime.getTimeOnly und DateTime.getTimeOnly	161
Listing 4.119:	Beispiel DateTime.setNow	162
Listing 4.120:	Beispiel DateTime.adjustSecond bis DateTime.adjustYear	164
Listing 4.121:	Beispiel DateTime.timeDifference	164
Listing 4.122:	Beispiel Session.createDateRange	165
Listing 4.123:	Beispiel DateRange.StartDateTime und DateRange.EndDateTime	166
Listing 4.124:	Beispiel RichTextItem.embedObject	167
Listing 4.125:	Beispiel RichTextItem.getEmbeddedObject	168
Listing 4.126:	Beispiel EmbeddedObject.doVerb	169
Listing 4.127:	Beispiel RichTextItem.embedObject	170
Listing 4.128:	Beispiel Log.openNotesLog	174
Listing 4.129:	Beispiel Log.openFileLog	175
Listing 4.130:	Beispiel Log.openMailLog	176
Listing 4.131:	Beispiel Log.logAction	177
Listing 4.132:	Beispiel Log.LogActions	177
Listing 4.133:	Beispiel Log.logError	178
Listing 4.134:	Beispiel Log.LogErrors	179
Listing 4.135:	Beispiel Database.createOutline	180
Listing 4.136:	Beispiel Database.getOutlinet	180
Listing 4.137:	Beispiel Outline.getFirst und Outline.getNextSibling	181
Listing 4.138:	Beispiel Name.Common	183
Listing 4.139:	Beispiel Session.createNewsletter	185
Listing 4.140:	Beispiel Newsletter.formatMsgWithDoclinks	187
Listing 4.141:	Beispiel Newsletter.formatDocument	188
Listing 5.1:	Beispiel JDBC – Schritt1	191
Listing 5.2:	Beispiel JDBC – Schritt2	192
Listing 5.3:	Beispiel JDBC – Schritt3	193
Listing 5.4:	Beispiel JDBC – Schritt4	193
Listing 5.5:	Beispiel JDBC – Schritt5	194
Listing 5.6:	Beispiel JDBC-Anweisung	195
Listing 7.1:	Beispiel Servlet	202
Listing 7.2:	Beispiel Bean	203
Listing 7.3:	Beispiel Servlet – Schritt1	211
Listing 7.4:	Beispiel Servlet – Schritt2	211
Listing 7.5:	Beispiel Servlet – Schritt3	212
Listing 7.6:	Beispiel Servlet – Schritt4	212
Listing 7.7:	Beispiel Servlet – Schritt5	213
Listing 7.8:	Beispiel Servlet – Schritt6	214
Listing 7.9:	Beispiel Servlet – Schritt7	215
Listing 7.10:	Beispiel Servlet – Schritt8	216
Listing 8.1:	Beispiel Bean	222

Stichwortverzeichnis

► A

- abstract 86
- Abstrakte Klassen 33
- Abstrakte Methoden 33
- ACL
 - addRole 133
 - deleteRole 134
 - getInternetLevel 138
 - isUniformAccess 137
 - setInternetLevel 138
 - setUniformAccess 137
- ACL.createACLEntry 124
- ACL.getEntry 123
- ACL.getFirstEntry 122
- ACL.getNextEntry 122
- ACL.getRoles 132
- ACL.removeACLEntry 125
- ACLEntry
 - disableRole 137
 - enableRole 136
 - getLevel 129
 - getName 130
 - getRoles 134
 - getUserType 126
 - isAdminServer 128
 - isCanCreateDocuments 131
 - isCanCreateLSOrJavaAgent 131
 - isCanCreatePersonalAgent 131
 - isCanCreatePersonalFolder 131
 - isCanDeleteDocuments 131
 - isGroup 127
 - isPerson 127
 - isPublicReader 131
 - isPublicWriter 131
 - isRoleEnabled 135
 - isServer 127
 - remove 126
 - setAdminServer 128
 - setCanCreateDocuments 131
 - setCanCreateLSOrJavaAgent 131
 - setCanCreatePersonalAgent 131
 - setCanCreatePersonalFolder 131
 - setCanDeleteDocuments 131
 - setGroup 127
 - setLevel 129
 - setName 130
 - setPerson 127
 - setPublicReader 131
 - setPublicWriter 131
 - setServer 127
 - setUserType 126
- Adressdatenbank 63
- Agent 41, 46
 - getCommonOwner 46
 - getName 46
 - getOwner 46
 - getParameterDocID 47
 - getSavedData 49
 - isEnabled 46
 - isNotesAgent 46
 - isPublic 46
 - isWebAgent 46
 - runOnServer 48
- AgentBase 46
 - run 47
- AgentContext
 - getCurrentDatabase 63
- Agentprogrammierung 41
- Applet 53
 - init 54
 - paint 54
 - start 54
 - stop 54
- AppletBase
 - notesAppletDestroy 55
 - notesAppletInit 55
 - notesAppletStart 55
 - notesAppletStop 55
- Arrays 21
- Ausnahmebehandlung 36

► B

- boolean 21
- byte 21

► C

- Casting 35
- catch 36
- Categories 89
- char 21
- clearHistory 82
- compact 71

- Cookie 205–206, 210
 - setMaxAge 210
 - setValue 211
- CORBA 51
- createCopy 70
- Created 75
- createDatabase 68
- createFromTemplate 69
- createReplica 70
- CurrentDatabase 62
- Cutoff-Datum 84
- CutoffDelete 85
- Cutoff-Mechanismus 85
- cutoffInterval 83–84
- **D**
- Database 57
 - compact 71
 - createCopy 70
 - createDocument 114
 - createFromTemplate 69
 - createOutline 179
 - createReplica 70
 - FTSearch 112
 - getACL 121
 - getAllDocuments 102
 - getCategories 89
 - getCreated 75
 - getDesignTemplateName 88
 - getDocumentByID 99
 - getDocumentByUNID 100
 - getFileName 74
 - getFilePath 74
 - getFolderReferencesEnabled 96
 - getLastFTIndexed 93
 - getLastModified 76
 - getManagers 97
 - getMaxSize 91
 - getOutline 180
 - getParent 98
 - getPercentUsed 92
 - getProfileDocCollection 120
 - getProfileDocument 119
 - getReplicaID 77
 - getReplicationInfo 78
 - getServer 73
 - getSize 90
 - getSizeQuota 90
 - getTemplateName 87
 - getTitle 74
 - getUniversalID 100
 - isDelayUpdates 95
 - isFTIndexed 93
 - isMultiDbSearch 94
 - isOpen 67
 - isPrivateAddressBook 64
 - isPublicAddressBook 64
 - open 59
 - remove 72
 - replicate 71
 - search 111
 - setCategories 89
 - setDelayUpdates 95
 - setFolderReferencesEnabled 96
 - setSizeQuota 90
 - setTitle 74
 - setUniversalID 100
 - unprocessedSearch 113
- Dateianhänge 166, 170
- Datenbankkategorie 88
- Datenbankschablone 87
- Datenbankvorlage 87
- Datentypen 20
- DateTime
 - adjustDay 163
 - adjustHour 163
 - adjustMinute 163
 - adjustMonth 163
 - adjustSecond 163
 - adjustYear 163
 - getDateOnly 161
 - getEndDateTime 166
 - getStartDateTime 166
 - getTimeOnly 161
 - setAnyDate 161
 - setAnyTime 162
 - setEndDateTime 166
 - setNow 162
 - setStartDateTime 166
 - timeDifference 164
 - timeDifferenceDouble 164
- Datums- und Zeithandling 160
- Datumsbereich 165
- Datumsdifferenz 164
- DbDirctory
 - openMailDatabase 65
- DbDirectory
 - createDatabase 68
 - getFirstDatabase 58
 - getNextDatabase 58
 - openDatabase 60
 - openDatabaseByReplicaID 61
 - openDatabaseSelfModified 62
- DelayUpdates 95
- DesignTemplateName 88
- disabled 81

Document

- appendText 150
- copyToDatabase 116
- createRichTextItem 149
- getFirstItem 140
- getItems 140
- getNoteID 99
- hasItem 141
- isSaveMessageOnSend 118
- remove 116
- save 115
- send 117

DocumentCollection

- addDocument 106
- deleteDocuzment 105
- FTSearch 112
- getFirstDocument 103
- getLastDocument 104
- getNextDocument 103
- getNthDocument 105
- getPrevDocument 104
- putAllInFolder 108
- removeAll 107
- removeAllFromFolder 109
- stampAll 109
- unprocessedFTSearch 113
- updateAll 110

Dokumente 98

Dokumentensammlung 102

Dokument-ID 98

do-Schleife 24

double 21

► E

Eingebettete Objekte 167

Eingebundene Objekte 166

Elementardatentypen 21

Elementaroperatoren 22

EmbeddedObject

- doVerb 169
- extractFile 171
- remove 171

Exception 36

► F

filename 73

filepath 73

final 21, 35

finally 37

float 21

FolderRefsEnabled 96

for 24

FTIndexed 93

► G

Garbage Collector 22

getAbstract 86

getAddressBooks 63

getCategories 89

getCreated 75

getCurrentDatabase 62

getCutoffDate 84

getCutoffInterval 83

getDatabase 59

getDbDirectory 57

getDesignTemplateName 88

getFileName 74

getFilePath 74

getFirstDatabase 57

getFolderRefsEnabled 96

getLastFTIndexed 93

getLastModified 76

getManagers 97

getMaxSize 91

getNextDatabase 57

getPercentUsed 92

getPriority 80

getReplicaID 77

getReplicationInfo 78

getServer 73

getSize 89

getSizeQuota 90

getTemplateName 87

getTitle 74

getURLDatabase 65

Gliederung 179

► H

HTML 197

HttpResponse 206

HttpServletRequest 204

getCookies 205

getMethode 205

getParameter 204

getParameterNames 204

getParameterValues 204

getPathInfo 205

getRequestedSessionID 205

getSession 204

isRequestedSessionIdValid 205

HttpServletResponse

addCookie 206

getWriter 206

sendRedirect 206

setStatus 207

HttpSession 208
 getCreationTime 209
 getId 209
 getLastAccessedTime 209
 getRemoteAddr 208
 getRemoteHost 209
 getServerName 209
 getServerPort 209
 getValue 208
 invalidate 210
 isNew 210
 putValue 208
 removeValue 208
HTTP-Status 206

► I

if-then-else 23
IgnoreDeletes 79
implements 36
INSERT 190
int 21
Interface 35
Internationalisierung 171
Internetzugriffsrechte 138
isCutoffDelete 85
isDelayUpdates 95
isDisabled 81
isFTIndex 93
isIgnoreDeletes 79
isIgnoreDestDeletes 79
isMultiDbSearch 94
isOpen 67
isPrivateAddressBook 64
isPublicAddressBook 64
Item 139
 containsValue 145
 getDateTimeValue 142
 getItemValueDouble 148
 getItemValueInteger 148
 getItemValueString 148
 getValueDouble 143
 getValueInteger 143
 getValues 145
 getValueString 144
 isAuthors 146
 isEncrypted 147
 isNames 146
 isProtected 147
 isReaders 146
 isSaveToDisk 147
 isSigned 147
 isSummary 147

 replaceItemValue 148
 replaceItemValueCustomData 148
 setAuthors 146
 setDateTimeValue 142
 setEncrypted 147
 setNames 146
 setProtected 147
 setReaders 146
 setSaveToDisk 147
 setSigned 147
 setSummary 147
 setValueDouble 143
 setValueInteger 143
 setValues 145
 setValueString 144

► J

Java Beans 203, 222
Java Virtual Machine 17
JDBC 189
JDK 39
JSP 217
JVM 51

► K

Kapselung 30
Klassen 28
Kommentare 19
Komprimieren 68
Konstanten 21

► L

LastFTIndexes 93
LastModified 76
Log
 createLog 173
 getNumActions 179
 getNumErrors 179
 isLogAction 177
 isLogErrors 178
 logAction 176
 logError 178
 openFileLog 175
 openMailLog 176
 setLogAction 177
 setLogErrors 178
long 21

► M

Maildatenbank 65
managers 97
maxSize 91

►N

Name

- Common 183
- getAbbreviated 182
- getADMD 184
- getCanonical() 183
- getCommon 184
- getCountry 184
- getGeneration 184
- getGiven 184
- getInitials 184
- getKeyword 184
- getLanguage 185
- getOrganization 184
- getOrgUnit1 184
- getOrgUnit2 184
- getOrgUnit3 184
- getOrgUnit4 184
- getSurname 184
- isHierarchical 183

Namenshandling 182

Newsletter 185

- formatDocument 187
- formatMsgWithDoclinks 186
- getSubjectItemName 186
- isDoScore 186
- isDoSubject 186
- setDoScore 186
- setDoSubject 186
- setSubjectItemName 186

Nicht druckbare Sonderzeichen 27

notesAppletDestroy 55

notesAppletInit 55

notesAppletStart 55

notesAppletStop 55

►O

Objekte 27

Objektinstanzen 27

Objektorientierte

- Softwareentwicklung 27

ODBC 191

open 58

openDatabase 60

openDatabaseByReplicaID 61

openDatabaseIfModified 62

openMailDatabase 65

Outline

- getFirst 181
- getFirstSibling 181
- getNext 181
- getNextSibling 181

►P

Packages 25

ParentDatabase 67

private 31

Profildokument 119

protected 31

Protokolldatei 174

Protokolldatenbank 173

Protokollierung 173

Protokollmail 175

Protokollmedium 173

public 31

►R

remove 72

replicate 71

Replication

- clearHistory 82
- getCutoffDate 84
- getCutoffInterval 83
- getPriority 80
- isAbstract 86
- isCutoffDelete 85
- isDisabled 81
- isIgnoreDeletes 79
- isIgnoreDestDeletes 79
- reset 78
- save 78
- setAbstract 86
- setCutoffDelete 85
- setCutoffInterval 83
- setDisabled 81
- setIgnoreDeletes 79
- setIgnoreDestDeletes 79
- setPriority 80

Replikationshistorie 82

Replikations-ID 77

Replikationspriorität 80

Replizierung 70, 77

reset 78

resolve 66

RichText 148

RichTextItem

- addNewLine 150
- addPageBreak 150
- appendDocLink 151
- appendParagraphStyle 156
- appendRTItem 151
- appendStyle 153
- EmbeddedObject 168
- embedObject 167, 170–171
- getEmbeddedObject 168
- getFormattedText 152

RichTextParagraphStyle

clearAllTabs 160
setTab 158
setTabs 159

►S

Schleifen 24
Schnittstelle 36
SELECT 189
server 73
Servlets 201
Session
 createDateTime 160
 createDateRange 165
 createName 182
 createNewsletter 185
 createRichTextStyle 153, 156
 getAddressBooks 63
 getDatabase 60
 getDbDirectory 57
 getURLDatabase 65
 getUserNameObject 182
 resolve 66
setAbstract 86
setCategories 89
setCutoffDelete 85
setCutoffInterval 83
setDelayUpdate 95
setDisabled 81
setFolderReferencesEnabled 96
setIgnoreDeletes 79
setPriority 80
setSizeQuota 90
setTitle 74
short 21
Sichtbarkeit 31
SizeQuota 90
Sonderzeichen 27
SQL 189
Standalone 49
static 21
Statische Methoden 33
Statische Variablen 33
switch-case 23

►T

Tabulatoren 158
Tag Bibliothek 224
TagLib 221, 224
TemplateName 87
throw 38
try 36

►U

Überladung 32
Universal-ID 100
UPDATE 190

►V

Variablen 20
Vererbung 32
Verknüpfte Objekte 169
Verzögertes Schreiben 95
Verzweigungen 23
virtual 32
Virtual Machine 17
Visual Age 39
Volltextindex 93
Volltextsuche 112

►W

Währungsangaben 171
Webnavigator-Datenbank 65
while-Schleife 25

►Z

Zeichenketten 26
Zeitbereich 164–165
Zeitdifferenz 164
Zeithandling 160
Zeitspanne 165
Zugriffskontrollliste 121
Zugriffslevel 124
Zugriffsrollen 132
Zugriffstypen 127



Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt.

Dieses eBook stellen wir lediglich als **Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks und zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Plazierung auf anderen Websites, der Veränderung und der Veröffentlichung bedarf der schriftlichen Genehmigung des Verlags.

Bei Fragen zu diesem Thema wenden Sie sich bitte an:

<mailto:info@pearson.de>

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf der Website ist eine freiwillige Leistung des Verlags. Der Rechtsweg ist ausgeschlossen.

Hinweis

Dieses und andere eBooks können Sie rund um die Uhr
und legal auf unserer Website



(<http://www.informit.de>)
herunterladen