

Auf einen Blick

1	Initialisierung und Setup	27
2	Package Management	67
3	Logging und Debugging	103
4	Konfiguration und Internationalisierung	131
5	Dateisystem, Streams und Events	165
6	Datenformate	211
7	Persistenz	285
8	Webanwendungen und Webservices	333
9	Sockets und Messaging	391
10	Testing und TypeScript	449
11	Skalierung, Performance und Sicherheit	481
12	Native Module	543
13	Publishing, Deployment und Microservices	587

Inhalt

Materialien zum Buch	19
Geleitwort des Fachgutachters	21
Vorwort	23
1 Initialisierung und Setup	27
<hr/>	
1.1 Rezept 1: Node.js installieren	27
1.1.1 Arten der Installation	27
1.1.2 Lösung: Installation über Installationsdatei unter macOS	30
1.1.3 Lösung: Installation über Installationsdatei unter Windows	32
1.1.4 Lösung: Installation über Binärpaket unter macOS	33
1.1.5 Lösung: Installation über Binärpaket unter Windows	33
1.1.6 Lösung: Installation über Binärpaket unter Linux	34
1.1.7 Lösung: Installation über Paketmanager	34
1.1.8 Ausblick	35
1.2 Rezept 2: Mehrere Node.js-Versionen parallel betreiben	35
1.2.1 Lösung	35
1.2.2 Alternativen	39
1.2.3 Ausblick	40
1.3 Rezept 3: Ein neues Node.js-Package manuell erstellen	41
1.3.1 Lösung	41
1.3.2 Modulsyntax	42
1.3.3 Weitere Dateien	43
1.3.4 Best Practices für Node.js-Packages	43
1.3.5 Ausblick	44
1.4 Rezept 4: Ein neues Node.js-Package automatisch erstellen	45
1.4.1 Möglichkeiten zur Initialisierung von Packages	45
1.4.2 Lösung: Packages über den Kommandozeilenwizard erstellen	45
1.4.3 Lösung: Packages über Starter-Kits erstellen	48
1.4.4 Lösung: Packages über Code-Generatoren erstellen	48
1.4.5 Ausblick	49
1.5 Rezept 5: Den Kommandozeilenwizard von npm anpassen	49
1.5.1 Lösung: Standardwerte anpassen	49
1.5.2 Lösung: Eine individuelle Package-Initialisierung einrichten	51
1.5.3 Ausblick	54

- 1.6 Rezept 6: Abhängigkeiten richtig installieren und verwalten** 54
 - 1.6.1 Die verschiedenen Typen von Abhängigkeiten verstehen 54
 - 1.6.2 Lösung: Installieren von Laufzeitabhängigkeiten 56
 - 1.6.3 Lösung: Installieren von Entwicklungsabhängigkeiten 57
 - 1.6.4 Lösung: Installieren von globalen Packages 57
 - 1.6.5 Lösung: Installieren von globalen Packages ohne sudo-Rechte 58
 - 1.6.6 Ausblick 59
- 1.7 Rezept 7: Packages in Mono-Repositorys organisieren** 59
 - 1.7.1 Lösung 59
 - 1.7.2 Lerna 60
 - 1.7.3 Aufbau von Mono-Repositorys 60
 - 1.7.4 Packages innerhalb eines Mono-Repositorys anlegen 62
 - 1.7.5 Projekt-globale Abhängigkeiten 62
 - 1.7.6 Voraussetzung: Git 63
 - 1.7.7 Workflow 64
 - 1.7.8 Ausblick 66
- 1.8 Zusammenfassung** 66

2 Package Management 67

- 2.1 Rezept 8: Semantische Versionierung richtig einsetzen** 67
 - 2.1.1 Exkurs: semantische Versionierung 67
 - 2.1.2 Lösung: variable Versionsnummern verwenden 68
 - 2.1.3 Lösung: exakte Versionsnummern verwenden 70
 - 2.1.4 Die Datei package-lock.json 71
 - 2.1.5 Die Datei npm-shrinkwrap.json 72
 - 2.1.6 Ausblick 73
- 2.2 Rezept 9: Den alternativen Package Manager »Yarn« verwenden** 73
 - 2.2.1 Lösung 73
 - 2.2.2 Einführung und Vergleich zu npm 74
 - 2.2.3 Installation 74
 - 2.2.4 Verwendung 75
 - 2.2.5 Lock-Datei 77
 - 2.2.6 Ausblick 78
- 2.3 Rezept 10: Den alternativen Package Manager »pnpm« verwenden** 78
 - 2.3.1 Lösung 78
 - 2.3.2 Einführung und Vergleich zu npm 78

2.3.3	Installation	80
2.3.4	Verwendung	81
2.3.5	Ausblick	84
2.4	Rezept 11: Lokale Abhängigkeiten für die Entwicklung verlinken	84
2.4.1	Lösung	85
2.4.2	Alternativen	87
2.5	Rezept 12: Informationen zu verwendeten Abhängigkeiten abrufen	88
2.5.1	Lösung: allgemeine Informationen für ein Package ermitteln	88
2.5.2	Lösung: Downloadstatistik eines Packages ermitteln	90
2.5.3	Lösung: den Abhängigkeitsbaum eines Packages ermitteln	92
2.5.4	Ausblick	93
2.6	Rezept 13: Lizenzen der verwendeten Abhängigkeiten ermitteln	94
2.6.1	Lösung	94
2.6.2	Alternativen	95
2.6.3	Ausblick	96
2.7	Rezept 14: Nicht verwendete oder fehlende Abhängigkeiten ermitteln	97
2.7.1	Lösung	97
2.7.2	Weitere Features	98
2.7.3	Ausblick	98
2.8	Rezept 15: Veraltete Abhängigkeiten ermitteln	99
2.8.1	Lösung	99
2.8.2	Alternativen	100
2.9	Zusammenfassung	101
3	Logging und Debugging	103
3.1	Rezept 16: Logging für Node.js-Packages einrichten	103
3.1.1	Exkurs: Logging	103
3.1.2	Lösung: das »debug«-Package	106
3.1.3	Ausblick	108
3.2	Rezept 17: Logging für Node.js-Applikationen einrichten	109
3.2.1	Lösung: Logging mit »winston«	109
3.2.2	Lösung: Logging mit »bunyan«	111
3.2.3	Lösung: Logging mit »log4js-node«	112
3.2.4	Ausblick	113

3.3	Rezept 18: Logging über Adapter-Packages einrichten	114
3.3.1	Lösung	114
3.3.2	Ausblick	117
3.4	Rezept 19: Applikationen mit Chrome Developer Tools debuggen	118
3.4.1	Vorbereitung	118
3.4.2	Lösung	119
3.4.3	Ausblick	122
3.5	Rezept 20: Applikationen mit Visual Studio Code debuggen	122
3.5.1	Lösung	122
3.5.2	Ausblick	124
3.6	Rezept 21: Applikationen über die Kommandozeile debuggen	125
3.6.1	Lösung	125
3.6.2	Ausblick	129
3.7	Zusammenfassung	129
4	Konfiguration und Internationalisierung	131
<hr/>		
4.1	Rezept 22: Applikationen konfigurieren über Umgebungsvariablen	132
4.1.1	Exkurs: Konfiguration von Applikationen	132
4.1.2	Lösung: Umgebungsvariablen mit der Node.js-API auslesen	132
4.1.3	Lösung: Umgebungsvariablen über .env-Datei definieren	134
4.1.4	Ausblick	137
4.2	Rezept 23: Applikationen konfigurieren über Konfigurationsdateien	137
4.2.1	Lösung: Konfigurationsdateien mit JSON	137
4.2.2	Lösung: Konfigurationsdateien mit JavaScript	139
4.2.3	Ausblick	141
4.3	Rezept 24: Applikationen konfigurieren über Kommandozeilenargumente	141
4.3.1	Lösung: auf Kommandozeilenargumente zugreifen über die Standard-Node.js-API	141
4.3.2	Lösung: auf Kommandozeilenargumente zugreifen über »yargs«	144
4.3.3	Lösung: auf Kommandozeilenargumente zugreifen über »commander.js«	145
4.3.4	Ausblick	147
4.4	Rezept 25: Applikationen optimal konfigurierbar machen	147
4.4.1	Lösung	147
4.4.2	Ausblick	150

4.5	Rezept 26: Mehrsprachige Applikationen erstellen	151
4.5.1	Exkurs: i18n	151
4.5.2	Die Internationalization API	152
4.5.3	Lösung: Vergleich von Zeichenketten	153
4.5.4	Lösung: Formatierung von Datums- und Zeitangaben	156
4.5.5	Lösung: Formatierung von Zahlenwerten	158
4.5.6	Ausblick	161
4.6	Rezept 27: Sprachdateien verwenden	161
4.6.1	Verwenden von Sprachdateien mit »i18n«	161
4.6.2	Ausblick	163
4.7	Zusammenfassung	163
5	Dateisystem, Streams und Events	165
<hr/>		
5.1	Rezept 28: Mit Dateien und Verzeichnissen arbeiten	165
5.1.1	Lösung	166
5.1.2	Dateien lesen	167
5.1.3	Dateien schreiben	169
5.1.4	Weitere Methoden	170
5.1.5	Über Verzeichnisse iterieren	171
5.1.6	Ausblick	172
5.2	Rezept 29: Dateien und Verzeichnisse überwachen	173
5.2.1	Einführung	173
5.2.2	Lösung: Dateien und Verzeichnisse überwachen mit dem »fs«-Modul	174
5.2.3	Lösung: Dateien und Verzeichnisse überwachen mit »chokidar«	175
5.2.4	Ausblick	178
5.3	Rezept 30: Daten mit Streams lesen	178
5.3.1	Exkurs: Streams	178
5.3.2	Lösung	180
5.3.3	Ausblick	182
5.4	Rezept 31: Daten mit Streams schreiben	182
5.4.1	Lösung	182
5.4.2	Ausblick	184

5.5	Rezept 32: Mehrere Streams über Piping kombinieren	184
5.5.1	Lösung	185
5.5.2	Piping im Produktionsbetrieb	188
5.5.3	Ausblick	190
5.6	Rezept 33: Eigene Streams implementieren	190
5.6.1	Lösung	190
5.6.2	Lösung: Einen Readable Stream implementieren	191
5.6.3	Lösung: Einen Writable Stream implementieren	193
5.6.4	Lösung: Einen Duplex Stream implementieren	194
5.6.5	Lösung: Einen Transform Stream implementieren	197
5.6.6	Ausblick	199
5.7	Rezept 34: Events versenden und empfangen	199
5.7.1	Lösung	200
5.7.2	Ausblick	205
5.8	Rezept 35: Erweiterte Features beim Event-Handling verwenden	206
5.8.1	Lösung	206
5.8.2	Ausblick	208
5.9	Zusammenfassung	209
6	Datenformate	211
<hr/>		
6.1	Rezept 36: XML verarbeiten	211
6.1.1	Einführung: XML-Verarbeitung	212
6.1.2	Lösung 1: XML mit einem DOM-Parser verarbeiten	213
6.1.3	Lösung 2: XML mit einem SAX-Parser verarbeiten	215
6.1.4	Ausblick	218
6.2	Rezept 37: XML generieren	218
6.2.1	Lösung 1: XML generieren mit »xmlbuilder«	218
6.2.2	Lösung 2: XML generieren mit Template Strings	221
6.2.3	Ausblick	223
6.3	Rezept 38: RSS und Atom generieren und verarbeiten	224
6.3.1	Lösung: RSS und Atom generieren	224
6.3.2	Lösung: RSS und Atom verarbeiten	227
6.3.3	Ausblick	229
6.4	Rezept 39: CSV verarbeiten	229
6.4.1	Lösung	229
6.4.2	Ausblick	232

6.5	Rezept 40: HTML mit Template-Engines generieren	233
6.5.1	Einführung	233
6.5.2	Lösung 1: HTML generieren mit »Pug«	234
6.5.3	Lösung 2: HTML generieren mit »EJS«	237
6.5.4	Ausblick	239
6.6	Rezept 41: HTML mit der DOM-API generieren	239
6.6.1	Exkurs: das Document Object Model	239
6.6.2	Lösung	241
6.6.3	Ausblick	244
6.7	Rezept 42: YAML verarbeiten und generieren	244
6.7.1	Exkurs: das YAML-Format	244
6.7.2	Lösung: YAML lesen	245
6.7.3	Lösung: YAML generieren	251
6.7.4	Ausblick	252
6.8	Rezept 43: TOML verarbeiten	253
6.8.1	Lösung	253
6.8.2	Ausblick	259
6.9	Rezept 44: INI verarbeiten und generieren	259
6.9.1	Lösung: INI verarbeiten	259
6.9.2	Lösung: INI generieren	263
6.9.3	Ausblick	263
6.10	Rezept 45: JSON validieren	264
6.10.1	Einführung	264
6.10.2	Lösung	266
6.10.3	Ausblick	269
6.11	Rezept 46: JavaScript verarbeiten und generieren	270
6.11.1	Lösung: JavaScript verarbeiten	270
6.11.2	Lösung: JavaScript generieren	275
6.11.3	Ausblick	276
6.12	Rezept 47: CSS verarbeiten und generieren	277
6.12.1	Lösung: CSS mit »PostCSS« verarbeiten	277
6.12.2	Lösung: eigene Plugins für »PostCSS« schreiben	282
6.12.3	Ausblick	284
6.13	Zusammenfassung	284

7	Persistenz	285
<hr/>		
7.1	Rezept 48: Auf eine MySQL-Datenbank zugreifen	285
7.1.1	Lösung	285
7.1.2	Ausblick	291
7.2	Rezept 49: Auf eine PostgreSQL-Datenbank zugreifen	292
7.2.1	Lösung	292
7.2.2	Ausblick	299
7.3	Rezept 50: Objektrelationale Mappings definieren	299
7.3.1	Exkurs: Objektrelationale Mappings	299
7.3.2	Lösung	301
7.3.3	Ausblick	307
7.4	Rezept 51: Auf eine MongoDB-Datenbank zugreifen	307
7.4.1	Lösung	308
7.4.2	Ausblick	316
7.5	Rezept 52: Auf eine Redis-Datenbank zugreifen	316
7.5.1	Lösung	316
7.5.2	Redis als Pub/Sub	324
7.5.3	Ausblick	325
7.6	Rezept 53: Auf eine Cassandra-Datenbank zugreifen	326
7.6.1	Lösung	326
7.6.2	Ausblick	332
7.7	Zusammenfassung	332
8	Webanwendungen und Webservices	333
<hr/>		
8.1	Rezept 54: Einen HTTP-Server implementieren	333
8.1.1	Lösung: einen Webserver mit der Standard-Node.js-API erstellen	333
8.1.2	Lösung: einen Webserver mit »Express« implementieren	335
8.1.3	Ausblick	336
8.2	Rezept 55: Eine Webanwendung über HTTPS betreiben	337
8.2.1	Lösung	337
8.2.2	Ausblick	341
8.3	Rezept 56: Eine REST-API implementieren	341
8.3.1	Exkurs: REST-APIs	341
8.3.2	Lösung: eine REST-API mit »Express« implementieren	342

8.3.3	Versionierung	353
8.3.4	Ausblick	354
8.4	Rezept 57: Einen HTTP-Client implementieren	355
8.4.1	Lösung: einen HTTP-Client mit der Standard-Node.js-API implementieren	355
8.4.2	Lösung: einen HTTP-Client mit »superagent« implementieren	356
8.4.3	Ausblick	358
8.5	Rezept 58: Authentifizierung implementieren	359
8.5.1	Lösung	359
8.5.2	Ausblick	365
8.6	Rezept 59: Authentifizierung mit »Passport.js« implementieren	366
8.6.1	Lösung	366
8.6.2	Ausblick	370
8.7	Rezept 60: Eine GraphQL-API implementieren	371
8.7.1	Exkurs: das Problem von REST	371
8.7.2	Lösung: dynamische APIs mit GraphQL	374
8.7.3	Ausblick	379
8.8	Rezept 61: Anfragen an eine GraphQL-API stellen	379
8.8.1	Lösung	380
8.8.2	Ausblick	383
8.9	Rezept 62: Eine GraphQL-API über HTTP betreiben	383
8.9.1	Lösung: GraphQL über HTTP betreiben	384
8.9.2	Lösung: GraphQL über »Express« betreiben	384
8.9.3	Lösung: GraphQL über den Apollo Server betreiben	388
8.9.4	Ausblick	389
8.10	Zusammenfassung	390
9	Socketts und Messaging	391
<hr/>		
9.1	Rezept 63: Einen TCP-Server erstellen	392
9.1.1	Lösung	392
9.1.2	Ausblick	395
9.2	Rezept 64: Einen TCP-Client erstellen	396
9.2.1	Lösung	396
9.2.2	Ausblick	400

9.3	Rezept 65: Einen WebSocket-Server erstellen	400
9.3.1	Lösung	400
9.3.2	Ausblick	405
9.4	Rezept 66: Einen WebSocket-Client erstellen	405
9.4.1	Lösung	405
9.4.2	Ausblick	407
9.5	Rezept 67: Nachrichtenformate für WebSocket-Kommunikation definieren	407
9.5.1	Lösung	407
9.5.2	Ausblick	412
9.6	Rezept 68: Subprotokolle für WebSocket-Kommunikation definieren	412
9.6.1	Lösung	412
9.6.2	Ausblick	415
9.7	Rezept 69: Server-Sent Events generieren	415
9.7.1	Lösung: die Server-Seite implementieren	415
9.7.2	Lösung: die Client-Seite implementieren	418
9.7.3	Ausblick	419
9.8	Rezept 70: Über AMQP auf RabbitMQ zugreifen	419
9.8.1	Exkurs: Messaging	419
9.8.2	Einführung: AMQP	421
9.8.3	Installation eines Message-Brokers für AMQP	425
9.8.4	Lösung	426
9.8.5	Ausblick	430
9.9	Rezept 71: Einen MQTT-Broker erstellen	431
9.9.1	Exkurs: das Nachrichtenprotokoll MQTT	431
9.9.2	Lösung: einen MQTT-Broker installieren	432
9.9.3	Lösung: einen MQTT-Broker über Node.js starten	434
9.9.4	Ausblick	436
9.10	Rezept 72: Über MQTT auf einen MQTT-Broker zugreifen	436
9.10.1	Lösung: einen MQTT-Client verwenden	436
9.10.2	Ausblick	443
9.11	Rezept 73: E-Mails versenden	443
9.11.1	Lösung	443
9.11.2	Ausblick	446
9.12	Zusammenfassung	446

10	Testing und TypeScript	449
10.1	Rezept 74: Unit-Tests schreiben	449
10.1.1	Exkurs: Unit-Tests und testgetriebene Entwicklung	449
10.1.2	Lösung: Unit-Test mit »Jest« schreiben	451
10.1.3	Ausblick	457
10.2	Rezept 75: Unit-Tests automatisch neu ausführen	458
10.2.1	Lösung	458
10.2.2	Ausblick	459
10.3	Rezept 76: Die Testabdeckung ermitteln	459
10.3.1	Lösung	460
10.3.2	Ausblick	463
10.4	Rezept 77: Unit-Tests für REST-APIs implementieren	463
10.4.1	Lösung	463
10.4.2	Ausblick	467
10.5	Rezept 78: Eine Node.js-Applikation in TypeScript implementieren	468
10.5.1	Lösung	469
10.5.2	Ausblick	475
10.6	Rezept 79: TypeScript-basierte Applikationen automatisch neu kompilieren	475
10.6.1	Lösung	475
10.6.2	Ausblick	479
10.7	Zusammenfassung	479
11	Skalierung, Performance und Sicherheit	481
11.1	Rezept 80: Externe Anwendungen als Unterprozess ausführen	482
11.1.1	Exkurs: Multithreading vs. Multiprocessing	482
11.1.2	Lösung	483
11.1.3	Ausblick	485
11.2	Rezept 81: Externe Anwendungen als Stream verarbeiten	486
11.2.1	Lösung	486
11.2.2	Ausblick	489
11.3	Rezept 82: Node.js-Applikationen als Unterprozess aufrufen	489
11.3.1	Lösung	489
11.3.2	Ausblick	495

11.4	Rezept 83: Eine Node.js-Anwendung clustern	495
11.4.1	Lösung	496
11.4.2	Ausblick	499
11.5	Rezept 84: Unterprozesse über einen Prozessmanager verwalten	499
11.5.1	Lösung	499
11.5.2	Ausblick	504
11.6	Rezept 85: Systeminformationen, CPU-Auslastung und Speicher- verbrauch ermitteln	504
11.6.1	Lösung: Systeminformationen, CPU-Auslastung und Speicherverbrauch ermitteln mit der Standard-Node.js-API	504
11.6.2	Lösung: Systeminformationen, CPU-Auslastung und Speicherverbrauch ermitteln mit »systeminformation«	509
11.6.3	Ausblick	511
11.7	Rezept 86: Speicherprobleme identifizieren	511
11.7.1	Lösung	512
11.7.2	Ausblick	519
11.8	Rezept 87: CPU-Probleme identifizieren	520
11.8.1	Lösung	520
11.8.2	Ausblick	528
11.9	Rezept 88: Schwachstellen von verwendeten Abhängigkeiten erkennen	528
11.9.1	Lösung: Schwachstellen erkennen mit npm	528
11.9.2	Lösung: Schwachstellen automatisch beheben mit npm	531
11.9.3	Lösung: Schwachstellen erkennen mit Third-Party-Tools	531
11.9.4	Lösung: Schwachstellen erkennen mit »Snyk«	532
11.9.5	Lösung: Schwachstellen erkennen mit »Retire.js«	533
11.9.6	Ausblick	535
11.10	Rezept 89: JavaScript dynamisch laden und ausführen	535
11.10.1	Einführung	535
11.10.2	Keine Lösung: JavaScript mit eval() ausführen	536
11.10.3	Lösung: JavaScript mit »vm« ausführen	537
11.10.4	Lösung: JavaScript mit »vm2« ausführen	540
11.10.5	Ausblick	541
11.11	Zusammenfassung	541

12 Native Module 543

12.1 Rezept 90: Native Node.js-Module mit der V8-API erstellen	543
12.1.1 Lösung	544
12.1.2 Ausblick	550
12.2 Rezept 91: Native Node.js-Module mit der NAN-API erstellen	550
12.2.1 Lösung	550
12.2.2 Ausblick	554
12.3 Rezept 92: Native Node.js-Module mit der N-API erstellen	554
12.3.1 Lösung	554
12.3.2 Ausblick	561
12.4 Rezept 93: Werte und Objekte zurückgeben mit der N-API	561
12.4.1 Lösung	561
12.4.2 Ausblick	568
12.5 Rezept 94: Callbacks aufrufen mit der N-API	568
12.5.1 Lösung	568
12.5.2 Ausblick	573
12.6 Rezept 95: Promises zurückgeben mit der N-API	573
12.6.1 Lösung	573
12.6.2 Ausblick	576
12.7 Rezept 96: Assertions verwenden mit der N-API	576
12.7.1 Lösung	576
12.7.2 Ausblick	580
12.8 Rezept 97: Native Node.js-Module debuggen	580
12.8.1 Lösung	580
12.8.2 Ausblick	584
12.9 Zusammenfassung	584

13 Publishing, Deployment und Microservices 587

13.1 Rezept 98: Eine private npm-Registry verwenden	588
13.1.1 Einführung: private npm-Registries	588
13.1.2 Lösung: private npm-Registry mit Verdaccio	589
13.1.3 Lösung: private npm-Registry mit Artefakt-Repositorys	592
13.1.4 Ausblick	595

13.2	Rezept 99: Docker verstehen	595
13.2.1	Einführung	595
13.2.2	Grundlagen	596
13.2.3	Node.js unter Docker	598
13.2.4	Docker-Befehlsreferenz	600
13.2.5	Docker Compose	603
13.2.6	Befehlsreferenz Docker Compose	604
13.2.7	Docker User Interfaces	605
13.2.8	Ausblick	606
13.3	Rezept 100: Ein Docker Image für eine Node.js-Applikation erstellen	607
13.3.1	Lösung	607
13.3.2	Ausblick	612
13.4	Rezept 101: Einen Docker-Container starten	612
13.4.1	Lösung	613
13.4.2	Ausblick	616
13.5	Rezept 102: Microservice-Architekturen verstehen	616
13.5.1	Eigenschaften von Microservice-Architekturen	616
13.5.2	Technologien bei Microservice-Architekturen	620
13.5.3	Kommunikation mit Microservices	620
13.5.4	API Gateways	622
13.5.5	Ausblick	625
13.6	Rezept 103: Microservice-Architekturen aufsetzen mit Docker Compose	625
13.6.1	Lösung	625
13.6.2	Ausblick	631
13.7	Rezept 104: Den Quelltext bündeln und komprimieren mit Webpack	631
13.7.1	Lösung	631
13.7.2	Ausblick	638
13.8	Zusammenfassung	639
	Anhang: Rezept 105	641
	Index	643