

ADOBE® DREAMWEAVER® CS3

DREAMWEAVER ERWEITERN

A large rectangular area with a green-to-dark-green gradient background. The letters 'Dw' are prominently displayed in a white, bold, sans-serif font on the left side of this area.

Dw

© 2007 Adobe Systems Incorporated. Alle Rechte vorbehalten.

Adobe® Dreamweaver® für Windows® und Macintosh

Wenn dieses Handbuch mit Software verteilt wird, die einen Endnutzervertrag beinhaltet, stehen das Handbuch sowie die darin beschriebene Software unter dieser Lizenz und dürfen nur in Übereinstimmung mit den Bedingungen dieser Lizenz verwendet oder kopiert werden. Sofern nicht durch eine solche Lizenz gestattet, darf kein Teil dieses Handbuchs ohne die vorherige schriftliche Genehmigung durch Adobe Systems Incorporated in irgendeiner Form oder durch irgendwelche Mittel elektronisch, mechanisch, durch Aufzeichnung oder auf andere Weise reproduziert, in einem Abfragesystem gespeichert oder übertragen werden. Bitte beachten Sie, dass der Inhalt dieses Handbuchs durch Urheberrechtsgesetze geschützt ist, auch wenn es nicht mit Software verteilt wird, die einen Endnutzervertrag beinhaltet. Der Inhalt dieses Handbuchs ist nur für Informationszwecke vorgesehen, kann ohne vorherige Benachrichtigung geändert werden und sollte nicht als eine Verpflichtung durch Adobe Systems Incorporated ausgelegt werden. Adobe Systems Incorporated übernimmt keine Verantwortung oder Haftung für etwaige Fehler oder Ungenauigkeiten, die im informatischen Inhalt in diesem Handbuch vorkommen mögen.

Bitte denken Sie daran, dass vorhandene Illustrationen oder Bilder, die Sie in Ihren Projekten möglicherweise verwenden wollen, durch Urheberrechtsgesetze geschützt sein können. Die unerlaubte Einbindung solcher Materialien in Ihr neues Werk könnte eine Verletzung der Rechte des Urheberrechtseigners darstellen. Bitte stellen Sie sicher, dass Sie jegliche erforderliche Genehmigung vom Urheberrechtseigner einholen. Die Bezugnahme auf Firmennamen in Beispielvorgaben dient ausschließlich Demonstrationszwecken und ist nicht als Bezugnahme auf irgendwelche tatsächlich existierenden Organisationen beabsichtigt.

Adobe, das Adobe-Logo, ActionScript, Adobe Bridge, ColdFusion, Creative Suite, Director, Dreamweaver, Fireworks, Flash, FlashPaper, HomeSite, JRun, Photoshop, Shockwave und Version Cue sind entweder eingetragene Marken oder Marken von Adobe Systems Incorporated in den USA.

ActiveX, Microsoft und Windows sind Marken oder eingetragene Marken der Microsoft Corporation in den USA und/oder anderen Ländern. Apple und Mac OS sind Marken von Apple Inc., eingetragen in den USA und anderen Ländern. Java und Solaris sind Marken oder eingetragene Marken von Sun Microsystems, Inc. in den USA und anderen Ländern. Linux ist eine eingetragene Marke von Linus Torvalds in den USA und anderen Ländern. UNIX ist eine Marke in den USA und in anderen Ländern, exklusiv lizenziert durch X/Open Company, Ltd. Alle anderen Marken gehören den jeweiligen Eigentümern.

Dieses Produkt enthält Software, die von der Apache Software Foundation (<http://www.apache.org/>) entwickelt wurde. Das Graphics Interchange Format © ist urheberrechtlich Eigentum von CompuServe Incorporated. GIF(sm) ist eine Dienstleistungsmarke, die CompuServe Incorporated gehört. MPEG Layer-3 Audiokomprimierungstechnologie lizenziert durch Fraunhofer IIS und Thomson Multimedia (<http://www.mp3licensing.com>). Das in der Software enthaltene MP3-komprimierte Audio kann nicht für Echtzeit- oder Live-Sendungen verwendet werden. Wenn Sie einen MP3-Decoder für Echtzeit- oder Live-Übertragungen benötigen, sind Sie dafür zuständig, die entsprechende MP3-Lizenz zu erhalten. Sprachkomprimierungs- und -dekomprimierungstechnologien stehen unter der Lizenz der Firma Nellymoser, Inc. (www.nellymoser.com). Flash CS3 Video wird betrieben mit der On2 TrueMotion-Videotechnik. © 1992-2005 On2 Technologies, Inc. Alle Rechte vorbehalten. <http://www.on2.com>. Dieses Produkt enthält Software, die von der OpenSymphony Group (<http://www.opensymphony.com/>) entwickelt wurde. Sorenson Spark™ Videokomprimierungs- und -dekomprimierungstechnologien stehen unter der Lizenz der Firma Sorenson Media, Inc.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Hinweis für Endnutzer der Regierung der Vereinigten Staaten. Die Software und das Begleitmaterial sind Handelswaren („Commercial Items“) im Sinne von 48 C.F.R. §2.101, bestehend aus kommerzieller Computersoftware („Commercial Computer Software“) und Begleitmaterial für kommerzielle Computersoftware („Commercial Computer Software Documentation“) im Sinne von 48 C.F.R. §12.212 bzw. 48 C.F.R. §227.7202. Gemäß 48 C.F.R. §12.212 bzw. 48 C.F.R. §§227.7202-1 bis 227.7202-4 werden die kommerzielle Computersoftware und das Begleitmaterial für kommerzielle Computersoftware für Endnutzer der Vereinigten Staaten (a) nur als Handelswaren und (b) nur mit jenen Rechten zur Verfügung gestellt, die allen anderen Endnutzern gewährt werden, gemäß den Bestimmungen des Lizenzvertrags für Endnutzer von Adobe. Unveröffentlichte Rechte bleiben gemäß den Gesetzen zum Urheberrecht der Vereinigten Staaten vorbehalten. Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Für Endnutzer der Regierung der Vereinigten Staaten verpflichtet sich Adobe, alle anwendbaren Gesetze zur Chancengleichheit anzuwenden, einschließlich der Bestimmungen der Executive Order 11246, einschließlich Ergänzungen, Abschnitt 402 des Vietnam Era Veterans Readjustment Assistance Act von 1974 (38 USC 4212), und Abschnitt 503 des Rehabilitation Act von 1973, einschließlich Ergänzungen, sowie Vorschriften in 41 CFR, 60-1 bis 60-60, 60-250 und 60-741. Die Gesetze und Vorschriften über aktive Förderungsmaßnahmen zugunsten von Minderheiten im vorgenannten Satz sind als Bezug in diesen Vertrag aufzunehmen.

Inhaltsverzeichnis

Kapitel 1: Einführung

Hintergrund	1
Erweiterungen installieren	1
Erweiterungen erstellen	2
Weitere Ressourcen für das Schreiben von Erweiterungen	2
Neue Funktionen in Dreamweaver	3
In diesem Handbuch verwendete Konventionen	3

Kapitel 2: Dreamweaver anpassen

Möglichkeiten zur Anpassung von Dreamweaver	5
Dreamweaver in einer Mehrbenutzerumgebung anpassen	12
FTP-Zuordnungen ändern	14
Erweiterbare Dokumenttypen in Dreamweaver	15
Die Zuordnungen von Tastaturbefehlen ändern	27

Kapitel 3: Codeansicht anpassen

Codehinweise	31
Codefarben	43
Codeüberprüfung	63
Die HTML-Standardformatierung ändern	66

Kapitel 4: Dreamweaver erweitern

Typen von Dreamweaver-Erweiterungen	67
Konfigurationsordner und Erweiterungen	69
Erweiterungs-APIs	71
Erweiterungen lokalisieren	73
Mit dem Extension Manager arbeiten	74

Kapitel 5: Benutzeroberflächen für Erweiterungen

Benutzeroberflächen von Erweiterungen gestalten	75
Steuerung der HTML-Darstellung in Dreamweaver	76
Benutzerdefinierte UI-Elemente in Erweiterungen verwenden	77
Flash-Inhalt zu Dreamweaver hinzufügen	84

Kapitel 6: Das Dreamweaver-Dokumentobjektmodell

Welches Dokument-DOM?	87
Das Dreamweaver-DOM	88

Kapitel 7: Objekte der Einfügeleiste

So funktionieren Objektdateien	95
Die Definitionsdatei für die Einfügeleiste	96
Einfügeleiste ändern	101
Einfaches Beispiel für das Einfügen eines Objekts	103
Die Funktionen der Objekt-API	111

Kapitel 8: API für Probleme bei der Browserkompatibilitätsprüfung

Funktionsweise der Ermittlung	117
Ein Beispiel für ein Problem	118
Die Funktionen der Problem-API	119

Kapitel 9: Befehle

Funktionsweise von Befehlen	123
Befehle in das Menü Befehle aufnehmen	124
Einfaches Befehlsbeispiel	124
Die Funktionen der Befehls-API	129

Kapitel 10: Menüs und Menübefehle

Die Datei menus.xml	133
Ändern von Menüs und Menübefehlen	140
Menübefehle	142
Beispiel für einen Menübefehl	144
Ein Beispiel für ein dynamisches Menü	147
Die Funktionen der Menübefehls-API	153

Kapitel 11: Symbolleisten

Funktionsweise von Symbolleisten	157
Eine einfache Symbolleisten-Befehlsdatei	159
Die Symbolleisten-Definitionsdatei	160
Symbolleistenelement-Tags	165
Attribute für Symbolleistenelement-Tags	169
Die API-Funktionen für Symbolleistenbefehle	174

Kapitel 12: Berichte

Site-Berichte	181
Eigenständige Berichte	183
Die API-Funktionen für Berichte	186

Kapitel 13: Tag-Bibliotheken und Tag-Editoren

Das Dateiformat einer Tag-Bibliothek	190
Die Tag-Auswahl	193
Einfaches Beispiel für die Erstellung neuer Tag-Editoren	195
Die API-Funktionen für den Tag-Editor	198

Kapitel 14: Eigenschafteninspektoren

Eigenschafteninspektor-Dateien	201
Funktionsweise von Eigenschafteninspektor-Dateien	202
Einfaches Beispiel für einen Eigenschafteninspektor	203
Die API-Funktionen für den Eigenschafteninspektor	206

Kapitel 15: Schwebende Bedienfelder

Funktionsweise von schwebenden Bedienfeldern	209
Einfaches Beispiel für schwebende Bedienfelder	210
API-Funktionen für schwebende Bedienfelder	215

Kapitel 16: Verhalten

Funktionsweise von Verhalten	221
Einfaches Beispiel für ein Verhalten	222
Die Funktionen der Verhaltens-API	226

Kapitel 17: Serververhalten

Dreamweaver-Architektur	233
Einfaches Beispiel für ein Serververhalten	235
API-Funktionen des Serververhaltens aufrufen	236
Die API für Serververhalten	238
Implementierungsfunktionen für das Serververhalten	242
EDML-Dateien bearbeiten	244
EDML-Datei-Tags gruppieren	246
Mitglieder-EDML-Dateien	251
Serververhaltenstechniken	268

Kapitel 18: Datenquellen

Funktionsweise von Datenquellen	275
Ein einfaches Datenquellenbeispiel	277
Die API-Funktionen für Datenquellen	283

Kapitel 19: Serverformate

Funktionsweise der Datenformatierung	289
Aufruf der Funktionen zur Datenformatierung	291
Die API-Funktionen für Serverformate	291

Kapitel 20: Komponenten

Grundlagen zu Komponenten	295
Das Bedienfeld „Komponenten“ erweitern	295
Anpassen des Bedienfelds „Komponenten“	296
Dateien für das Bedienfeld „Komponenten“	296
API-Funktionen für das Bedienfeld „Komponenten“	298

Kapitel 21: Servermodelle

Anpassung von Servermodellen	307
API-Funktionen für Servermodelle	307

Kapitel 22: Datenübersetzer

Funktionsweise von Datenübersetzern	313
Welche Übersetzer sollen verwendet werden?	314
Übersetztes Attribut in ein Tag aufnehmen	314
Übersetzte Attribute untersuchen	315
Übersetzte Tags und Codeblöcke sperren	316
Eigenschaftenspektoren für gesperrte Inhalte erstellen	317
Fehlersuche im Übersetzer	319
Einfaches Übersetzungsbeispiel	320
Einfaches Beispiel für einen Block/Tag-Übersetzer	323
Die API-Funktionen für Datenübersetzer	327

Kapitel 23: C-Level-Erweiterbarkeit

Integration von C-Funktionen	331
C-Level-Erweiterbarkeit und der JavaScript-Interpreter	333
Datentypen	333
Die C-Level-API	334
API für Dateizugriff- und Mehrbenutzerkonfiguration	341
C-Funktion über JavaScript aufrufen	347

Kapitel 24: Der Ordner „Shared“

Inhalt des Ordners „Shared“	349
Den Ordner „Shared“ verwenden	354

Index	355

Kapitel 1: Einführung

In diesem Handbuch werden das Adobe® Dreamweaver® CS3-Framework und die Anwendungsprogrammierschnittstelle (API) beschrieben, mit der Sie Erweiterungen zu Dreamweaver erstellen können. Das Handbuch enthält Informationen über die Funktionsweise der einzelnen Erweiterungstypen und über die API-Funktionen, die Dreamweaver zum Implementieren der verschiedenen Objekte, Menüs, schwebenden Bedienfelder, Serververhalten und anderer Elemente aufruft, die die Gesamtheit der Leistungsmerkmale von Dreamweaver ausmachen. Zu jedem Erweiterungstyp wird ein einfaches Beispiel geliefert. Darüber hinaus wird erläutert, wie Dreamweaver durch Bearbeiten von Tags in verschiedenen HTML- und XML-Dateien angepasst werden kann, um Menüelemente oder Dokumenttypen hinzuzufügen.

Um Objekte, Menüs, schwebende Bedienfelder oder andere Funktionen zu Dreamweaver hinzuzufügen, müssen Sie Code für die Funktionen des jeweiligen Erweiterungstyps schreiben. In diesem Handbuch werden die Argumente beschrieben, die Dreamweaver an diese Funktionen übergibt, sowie die Werte, die Dreamweaver als Rückgabewerte der Funktionen erwartet.

Weitere Informationen über die Verwendung und den allgemeinen Zweck von JavaScript™-APIs, die Sie zur Durchführung verschiedener Unterstützungsverfahren in Ihren Dreamweaver-Erweiterungen verwenden können, finden Sie im *Dreamweaver API-Referenzhandbuch*. Wenn Sie Erweiterungen erstellen möchten, die mit Datenbanken zusammenarbeiten, sollten Sie auch die Abschnitte unter *Dreamweaver verwenden* durchlesen, in denen erläutert wird, wie Sie Verbindungen zu Datenbanken herstellen können.

Hintergrund

Die meisten Dreamweaver-Erweiterungen sind in HTML und JavaScript geschrieben. In diesem Handbuch wird vorausgesetzt, dass Sie mit Dreamweaver, HTML, XML und der JavaScript-Programmierung vertraut sind. Wenn Sie C-Erweiterungen implementieren, wird davon ausgegangen, dass Sie DLLs (dynamische Bibliotheken) in C erstellen und verwenden können. Wenn Sie Erweiterungen zur Erstellung von Webanwendungen schreiben, sollten Sie darüber hinaus Kenntnisse über serverbasierte Skripts auf mindestens einer Plattform verfügen, wie z. B. Active Server Pages (ASP), ASP.NET, PHP: Hypertext Preprocessor (PHP), Macromedia® ColdFusion® von Adobe® oder JavaServer Pages (JSP).

Erweiterungen installieren

Um sich mit dem Schreiben von Erweiterungen vertraut zu machen, sollten Sie die Erweiterungen und Ressourcen ausprobieren, die auf der Adobe Exchange-Website zur Verfügung stehen (http://www.adobe.com/go/exchange_de). Durch das Installieren einer vorhandenen Erweiterung lernen Sie zahlreiche Tools kennen, die Sie bei der Arbeit mit Ihren eigenen Erweiterungen benötigen.

So installieren Sie eine Erweiterung:

- 1 Laden Sie den Adobe® Extension Manager herunter, und installieren Sie ihn. Dieser steht auf der Adobe Download-Website zur Verfügung (http://www.adobe.com/go/downloads_de).
- 2 Melden Sie sich auf der Adobe Exchange-Website an (http://www.adobe.com/go/exchange_de).
- 3 Wählen Sie aus den angebotenen Erweiterungen diejenige aus, die Sie verwenden möchten. Klicken Sie auf den Hyperlink zum Herunterladen des Erweiterungspakets.
- 4 Speichern Sie das Erweiterungspaket im Ordner **Dreamweaver\Downloaded Extensions** des Dreamweaver-Ordners.
- 5 Wählen Sie im Extension Manager die Option **Datei > Erweiterung installieren**. Wählen Sie in Dreamweaver die Option **Befehle > Erweiterungen verwalten**, um den Extension Manager zu starten.

Der Extension Manager installiert die im Ordner **Downloaded Extensions** befindliche Erweiterung automatisch in Dreamweaver.

In manchen Fällen muss Dreamweaver neu gestartet werden, bevor Sie auf eine Erweiterung zugreifen können. Falls Sie Dreamweaver während der Installation der Erweiterung ausführen, werden Sie unter Umständen dazu aufgefordert, das Programm zu beenden und neu zu starten.

Wechseln Sie in Dreamweaver zum Extension Manager (**Befehle > Erweiterungen verwalten**), um nach der Installation grundlegende Informationen über die Erweiterung einzusehen.

Erweiterungen erstellen

Bevor Sie eine Dreamweaver-Erweiterung erstellen, sollten Sie auf der Adobe Exchange-Website unter http://www.adobe.com/go/exchange_de nachsehen, ob die gewünschte Erweiterung möglicherweise schon vorhanden ist. Wenn Sie keine Erweiterung finden, die Ihren Anforderungen entspricht, führen Sie die folgenden Schritte aus, um eine Erweiterung zu erstellen:

- Bestimmen Sie den Typ der zu erstellenden Erweiterung. Weitere Informationen zu Erweiterungstypen finden Sie unter „Typen von Dreamweaver-Erweiterungen“ auf Seite 67.
- Lesen Sie die Dokumentation zu dem gewünschten Erweiterungstyp durch. Es empfiehlt sich auch, die einfache Beispielerweiterung im jeweiligen Abschnitt zu erstellen. Dadurch machen Sie sich mit der Erstellung des gewünschten Erweiterungstyps vertraut.
- Ermitteln Sie die Dateien, die bearbeitet oder erstellt werden müssen.
- Planen Sie ggf. die Benutzeroberfläche (UI) für die Erweiterung.
- Erstellen Sie die benötigten Dateien und speichern Sie sie in den entsprechenden Ordnern.
- Starten Sie Dreamweaver neu, damit die neue Erweiterung erkannt werden kann.
- Testen Sie die Erweiterung.
- Komprimieren Sie die Erweiterung in einem Paket, um sie anderen Benutzern zur Verfügung stellen zu können. Weitere Informationen finden Sie unter „Mit dem Extension Manager arbeiten“ auf Seite 74.

Weitere Ressourcen für das Schreiben von Erweiterungen

Wenn Sie mit anderen Entwicklern kommunizieren möchten, die Erweiterungen erstellen, können Sie der Dreamweaver Extensibility Newsgroup beitreten. Die Adobe Website dieser Newsgroup finden Sie (in englischer Sprache) unter folgender Adresse: <http://www.adobe.com/support/dreamweaver/extend/form/>.

Neue Funktionen in Dreamweaver

Dreamweaver CS3 umfasst die folgenden neuen Leistungsmerkmale und erweiterbaren Schnittstellen. Für jedes dieser Leistungsmerkmale stehen neue Funktionen zur Verfügung, die im *Dreamweaver API-Referenzhandbuch* aufgeführt sind.

- Datenaustausch mit Adobe Bridge
- Emulation von mobilen Geräten
- Erstellung von Spry-XML-Datensätzen
- Spry-Widgets bearbeiten und einfügen
- Browserkompatibilitätsprüfung

Informationen über die neuen Funktionen der Dienstprogramm-API und JavaScript-API finden Sie im *Dreamweaver API-Referenzhandbuch*.

Resource Center für Dokumentation

Erweitern Sie Ihre Dreamweaver-Kenntnisse mithilfe von Büchern von Adobe. Entdecken Sie mehr über die neuesten, von Experten geschriebenen Bücher auf der Webseite http://www.adobe.com/support/documentation/buy_books.html.

Veraltete Funktionen

In Dreamweaver CS3 wurden verschiedene ältere Funktionen abgelöst. Informationen über die Funktionen, die aus den Dienstprogramm- und JavaScript-APIs entfernt wurden, finden Sie im *Dreamweaver API-Referenzhandbuch*.

In diesem Handbuch verwendete Konventionen

In diesem Handbuch werden die folgenden typografischen Konventionen verwendet:

- *Codeschrift* kennzeichnet Codefragmente und API-Literale, z. B. Klassennamen, Methodenbezeichnungen, Funktionsnamen, Typnamen, Skripts, SQL-Anweisungen, HTML- und XML-Tagnamen sowie Attributnamen.
- *Kursive Codeschrift* kennzeichnet Elemente im Code, die zu ersetzen sind.
- Das Fortsetzungssymbol (–) weist darauf hin, dass ein langer Code über mehrere Zeilen umbrochen wurde. Da die Zeilenlänge in diesem Handbuch durch die im Format festgelegten Ränder begrenzt ist, muss Code, der eigentlich fortlaufend ist, auf mehrere Zeilen verteilt werden. Löschen Sie beim Kopieren der Codezeilen das Fortsetzungssymbol, und geben Sie die Zeilen ohne Umbruch ein.
- Geschweifte Klammern ({ }), die ein Argument einschließen, weisen darauf hin, dass es sich um ein optionales Argument handelt.
- Funktionsnamen mit dem Präfix `dreamweaver.` (wie in `dreamweaver.funktionsname`) können beim Schreiben von Code als `dw.funktionsname` abgekürzt werden. In diesem Handbuch wird das ausführliche Präfix `dreamweaver.` bei der Definition der Funktion und im Index verwendet. In vielen Beispielen wird jedoch das kürzere Präfix `dw.` verwendet.

Folgende Namensgebungskonventionen werden in diesem Handbuch verwendet:

- Sie – die Person, die für das Schreiben von Erweiterungen verantwortlich ist (also der Entwickler).
- Der Benutzer – die Person, die Dreamweaver verwendet.
- Der Besucher – die Person, die die vom Benutzer erstellte Webseite anzeigen lässt.

Kapitel 2: Dreamweaver anpassen

Zusätzlich zu der Erweiterungsfähigkeit lässt sich Adobe Dreamweaver CS3 auf vielfältige Weise anpassen, was Ihnen eine vertraute, einfache und effiziente Arbeitsweise ermöglicht.

Möglichkeiten zur Anpassung von Dreamweaver

Zur Anpassung von Dreamweaver gibt es mehrere Vorgehensweisen. Einige dieser Vorgehensweisen werden in *Dreamweaver verwenden* erläutert. Mithilfe dieser Vorgehensweisen können Sie Ihren Arbeitsbereich anpassen. Darüber hinaus können Sie die Einstellungen für Dialogfelder in Dreamweaver ändern. Sie können über das Bedienfeld **Voreinstellungen** (**Bearbeiten** > **Voreinstellungen** oder **Dreamweaver** > **Voreinstellungen** (Mac OS X)) für eine Vielzahl von Bereichen Benutzereinstellungen festlegen, z. B. für Eingabehilfen, Farbeinstellungen für Code, Schriften, Markierungen und die Vorschau in einem Browser. Außerdem können Sie mithilfe des Editors für Tastaturbefehle (**Bearbeiten** > **Tastaturbefehle**) oder durch die Bearbeitung einer Konfigurationsdatei Tastenkombinationen ändern.

Für die Anpassung von Dreamweaver durch Bearbeitung der Konfigurationsdateien haben Sie die folgenden Möglichkeiten:

- Sie können Objekte in der Einfügeleiste neu anordnen, neue Registerkarten zum Neuorganisieren von Objekten erstellen oder neue Objekte hinzufügen. Weitere Informationen finden Sie unter „Einfügeleiste ändern“ auf Seite 101.
- Sie können die Namen von Menüelementen ändern, neue Befehle in Menüs einfügen und Befehle aus Menüs entfernen. Weitere Informationen finden Sie unter „Menüs und Menübefehle“ auf Seite 133.
- Sie können ändern, wie Tags von Drittanbietern (einschließlich ASP- und JSP-Tags) in der Entwurfsansicht des Dokumentfensters angezeigt werden. Weitere Informationen finden Sie unter „Interpretation von Drittanbieter-Tags anpassen“ auf Seite 8.
- Sie können die Zuordnung von Tastaturbefehlen für Tastaturen ohne US-Englische Tastaturbelegung ändern. Weitere Informationen finden Sie unter „Die Zuordnungen von Tastaturbefehlen ändern“ auf Seite 27.

Darüber hinaus können Sie Dreamweaver durch folgende Schritte an Ihre Bedürfnisse anpassen:

- „Standarddokumente anpassen“ auf Seite 6
- „Seitendesigns anpassen“ auf Seite 6
- „Anzeige von Dialogfeldern anpassen“ auf Seite 6
- „Standard-Dateityp ändern“ auf Seite 7
- „Interpretation von Drittanbieter-Tags anpassen“ auf Seite 8
- „Arbeitsbereich-Layouts anpassen“ auf Seite 22
- „Symbolleiste in der Codeansicht anpassen“ auf Seite 26

Standarddokumente anpassen

Der Ordner **DocumentTypes\NewDocuments** enthält ein (leeres) Standarddokument für jeden Typ, den Sie in Dreamweaver erstellen können. Wenn Sie ein neues, leeres Dokument erstellen, indem Sie die Befehle **Datei > Neu** wählen und anschließend eine Option aus den Kategorien **Einfache Seite**, **Dynamische Seite** oder **Andere** wählen, erstellt Dreamweaver das neue Dokument basierend auf dem entsprechenden Standarddokument in diesem Ordner. Um die Anzeige eines Standarddokuments für einen bestimmten Typ zu ändern, bearbeiten Sie das entsprechende Dokument in diesem Ordner.

Hinweis: Wenn alle Seiten Ihrer Site gemeinsame Elemente (wie z. B. einen Urheberrechtshinweis) oder ein gemeinsames Layout aufweisen sollen, ist es besser, Vorlagen und Bibliothekselemente zu verwenden als die Standarddokumente zu ändern. Weitere Informationen zu Vorlagen und Bibliothekselementen finden Sie in Dreamweaver verwenden.

Seitendesigns anpassen

Dreamweaver stellt eine Reihe von vorentwickelten Cascading Stylesheets, Framesets und Seitendesigns zur Verfügung. Sie können Seiten basierend auf diesen Designs erstellen, indem Sie **Datei > Neu** wählen.

Um die verfügbaren Designs anzupassen, bearbeiten Sie die Dateien in den Ordnern **BuiltIn\css**, **BuiltIn\framesets**, **BuiltIn\Templates** und **BuiltIn\TemplatesAccessible**.

Hinweis: Bei den in den Kategorien **Seitendesigns** und **Seitendesigns (zugänglich)** aufgelisteten Designs handelt es sich um Dreamweaver-Vorlagendateien. Weitere Informationen über Vorlagen finden Sie unter Dreamweaver verwenden.

Sie können darüber hinaus benutzerdefinierte Seitendesigns erstellen, indem Sie Dateien zu den Unterordnern des Ordners **BuiltIn** hinzufügen. Damit eine Beschreibung der Datei im Dialogfeld **Neues Dokument** angezeigt wird, erstellen Sie eine Design Notes-Datei (im Ordner **appropriate _notes**), die der Seitendesign-Datei entspricht.

Anzeige von Dialogfeldern anpassen

Die Layouts der Dialogfelder für Objekte, Befehle und Verhalten sind als HTML-Formulare angegeben. Sie befinden sich im Dreamweaver-Anwendungsordner im Unterordner **Configuration** in HTML-Dateien. Sie können diese Formulare wie jedes andere Formular in Dreamweaver bearbeiten. Weitere Informationen hierzu finden Sie unter *Dreamweaver verwenden*.

Hinweis: Bei Mehrbenutzer-Betriebssystemen sollten Sie Kopien der Konfigurationsdateien in Ihrem Benutzer-Konfigurationsordner bearbeiten, anstatt die Dreamweaver-Konfigurationsdateien selbst zu bearbeiten. Weitere Informationen finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 70.

So ändern Sie die Anzeige von Dialogfeldern:

1 Wählen Sie in Dreamweaver **Bearbeiten > Voreinstellungen**, und wählen Sie anschließend die Kategorie **Codeumschreibung**.

2 Deaktivieren Sie die Option **Formularelemente beim Einfügen umbenennen**.

Durch die Deaktivierung dieser Option wird gewährleistet, dass die Formularelemente beim Kopieren und Einfügen ihren ursprünglichen Namen beibehalten.

3 Klicken Sie auf **OK**, um das Dialogfeld **Grundeinstellungen** zu schließen.

4 Suchen Sie auf der Festplatte in den Ordnern **Configuration\Objects**, **Configuration\Commands** und **Configuration\Behaviors** nach der entsprechenden HTML-Datei.

5 Erstellen Sie eine Kopie der Datei an einem anderem Speicherort als dem Konfigurationsordner.

6 Öffnen Sie die Kopie in Dreamweaver, bearbeiten Sie das Formular, und speichern Sie es.

7 Beenden Sie Dreamweaver.

8 Kopieren Sie die geänderte Datei zurück in den Konfigurationsordner, um die ursprüngliche Datei zu ersetzen. (Sie sollten zunächst eine Sicherungskopie der Originaldatei erstellen, um sie später bei Bedarf wiederherstellen zu können.)

9 Starten Sie Dreamweaver neu, damit die Änderungen wirksam werden.

Ändern Sie nur die Anzeige des Dialogfelds, jedoch nicht dessen Funktionsweise. Das Dialogfeld muss weiterhin die gleichen Arten von Formularelementen mit den gleichen Namen enthalten, damit die durch Dreamweaver vom Dialogfeld abgerufenen Informationen auf die gleiche Weise verwendet werden können.

Das Objekt **Kommentar** beispielsweise nimmt die Texteingabe aus einem Textbereich in einem Dialogfeld entgegen und verwendet eine einfache JavaScript-Funktion, um den Text in einen HTML-Kommentar umzuwandeln und diesen Kommentar in ein Dokument einzufügen. Das Formular, das dieses Dialogfeld beschreibt, befindet sich im Ordner **Configuration\Objects\Invisibles** in der Datei **Comment.htm**. Sie können die Datei öffnen und die Größe sowie andere Attribute des Textbereichs ändern. Wenn Sie jedoch das Tag `textarea` vollständig löschen oder den Wert seines `name`-Attributs ändern, funktioniert das Objekt **Kommentar** nicht ordnungsgemäß.

Standard-Dateityp ändern

Dreamweaver zeigt standardmäßig alle kompatiblen Dateitypen im Dialogfeld **Öffnen (Datei > Öffnen)** an. Mithilfe eines Pop-upmenüs in diesem Dialogfeld können Sie die Anzeige auf bestimmte Dateitypen beschränken. Wenn Sie hauptsächlich mit einem bestimmten Dateityp arbeiten (z. B. mit ASP-Dateien), können Sie die Standardanzeige ändern. Der Dateityp, der in der ersten Zeile der Dreamweaver-Datei **Extensions.txt** angezeigt wird, ist die Standardeinstellung.

Hinweis: Wenn alle Dateitypen im Dialogfeld **Öffnen** angezeigt werden sollen (auch Dateien, die Dreamweaver nicht öffnen kann), müssen Sie **Alle Dateien (*.*)** auswählen. Dies unterscheidet sich von der Option **Alle Dokumente**, die nur die Dateien anzeigt, die Dreamweaver öffnen kann.

So ändern Sie den Standard-Dateityp, den Dreamweaver bei Auswahl von Datei > Öffnen anzeigt:

- 1 Erstellen Sie eine Sicherungskopie der Datei **Extensions.txt** im Ordner **Configuration**.
- 2 Öffnen Sie die Datei **Extensions.txt** in Dreamweaver oder in einem Texteditor.
- 3 Schneiden Sie die Zeile aus, die der neuen Standardeinstellung entspricht, und fügen Sie sie am Anfang der Datei ein, sodass sie sich in der ersten Zeile der Datei befindet.
- 4 Speichern Sie die Datei.
- 5 Starten Sie Dreamweaver neu.

Um die neue Standardeinstellung zu überprüfen, wählen Sie **Datei > Öffnen**, und sehen Sie im Dateityp-Pop-upmenü nach.

So fügen Sie dem Menü im Dialogfeld Öffnen neue Dateitypen hinzu:

- 1 Erstellen Sie eine Sicherungskopie der Datei **Extensions.txt** im Ordner **Configuration**.
- 2 Öffnen Sie die Datei **Extensions.txt** in Dreamweaver oder in einem Texteditor.
- 3 Fügen Sie für jeden neuen Dateityp eine neue Zeile hinzu.
 - a Geben Sie, durch Kommas getrennt, die für den neuen Dateityp möglichen Dateierweiterungen in Großbuchstaben ein.
 - b Geben Sie dann einen Doppelpunkt ein sowie eine kurze Beschreibung, die dann im Dialogfeld **Öffnen** im Pop-upmenü **Dateityp** angezeigt wird.

Geben Sie für JPEG-Dateien beispielsweise Folgendes ein: `JPG, JPEG, JFIF:JPEG-Bilddateien`

- 4 Speichern Sie die Datei.
- 5 Starten Sie Dreamweaver neu.

Um die Änderungen zu überprüfen, wählen Sie **Datei > Öffnen**, und klicken Sie auf das Dateityp-Pop-upmenü.

Interpretation von Drittanbieter-Tags anpassen

Serverbasierte Technologien, wie z. B. ASP, ColdFusion, JSP und PHP, verwenden speziellen Nicht-HTML-Code in HTML-Dateien. Die entsprechenden Server erstellen auf der Grundlage dieses Codes HTML-Inhalt und stellen ihn bereit. Wenn Dreamweaver Nicht-HTML-Tags erkennt, vergleicht das Programm diese mit den Informationen in der Drittanbieter-Tag-Datei, in der festgelegt ist, wie Dreamweaver Nicht-HTML-Tags liest und anzeigt.

ASP-Dateien enthalten beispielsweise neben dem normalen HTML-Code auch ASP-Code, den der Server interpretieren muss. ASP-Code ist HTML-Tags sehr ähnlich, wird jedoch durch spezielle Trennzeichen gekennzeichnet: Der Code beginnt mit `<%` und endet mit `%>`. Der Dreamweaver-Ordner **Configuration\ThirdPartyTags** enthält die Datei **Tags.xml**, in der das Format unterschiedlicher Drittanbieter-Tags beschrieben ist, auch das von ASP-Code, und die definiert, wie Dreamweaver diesen Code anzeigt. Durch die Definition des ASP-Codes in der Datei **Tags.xml** versucht Dreamweaver nicht, den Inhalt zwischen den Trennzeichen zu interpretieren, sondern zeigt in der Entwurfsansicht ein Symbol an, das auf den ASP-Code hinweist. Sie können mithilfe Ihrer eigenen Tag-Datenbankdateien festlegen, wie Dreamweaver Tags liest und anzeigt. Erstellen Sie eine neue Tag-Datenbankdatei für jeden Tag-Satz, um Dreamweaver anzuweisen, wie die Tags anzuzeigen sind.

Hinweis: In diesem Abschnitt wird erläutert, wie die Anzeige von benutzerdefinierten Tags in Dreamweaver definiert wird. Es wird jedoch nicht erläutert, wie der Inhalt oder die Eigenschaften von benutzerdefinierten Tags bearbeitet werden können. Weitere Informationen über die Erstellung eines Eigenschafteninspektors zur Überprüfung und Änderung der Eigenschaften von benutzerdefinierten Tags finden Sie in „Eigenschafteninspektoren“ auf Seite 201.

Jede Tag-Datenbankdatei definiert den Namen, den Typ, das Inhaltsmodell, das Darstellungsschema und das Symbol für ein oder mehrere benutzerdefinierte Tags. Sie können eine beliebige Anzahl von Tag-Datenbankdateien erstellen, die sich jedoch alle im Ordner **Configuration\ThirdPartyTags** befinden müssen, damit Dreamweaver sie lesen und verarbeiten kann. Tag-Datenbankdateien haben die Dateierweiterung „.xml“.

 Wenn Sie an mehreren, unabhängigen Sites gleichzeitig arbeiten (beispielsweise als freiberuflicher Entwickler), können Sie alle Tag-Spezifikationen für eine bestimmte Site in einer Datei zusammenfassen. Geben Sie diese Tag-Datenbankdatei dann einfach zusammen mit den benutzerdefinierten Symbolen und Eigenschafteninspektoren an die Personen weiter, die die Site pflegen.

Sie erstellen eine Tag-Spezifikation mithilfe des XML-Tags `tagspec`. So beschreibt beispielsweise der folgende Code die Spezifikation für das Tag `happy`:

```
<tagsec tag_name="happy" tag_type="nonempty" render_contents="false" content_model="marker_model" icon="happy.gif" icon_width="18" icon_height="18"></tagsec>
```

Mithilfe von `tagsec` können Sie die beiden folgenden Tag-Typen definieren:

- Normale Tags im HTML-Stil

Bei dem Beispiel mit dem Tag `happy` handelt es sich um ein Tag im normalen HTML-Stil. Es beginnt mit einem öffnenden `<happy>`-Tag, enthält zwischen dem öffnenden und schließenden Tag Daten und endet mit dem schließenden `</happy>`-Tag.

- Zeichenfolgenbegrenzte Tags

Zeichenfolgenbegrenzte Tags beginnen mit einer Zeichenfolge und enden mit einer anderen. Sie sind mit leeren HTML-Tags (wie z. B. `img`) in der Hinsicht vergleichbar, dass sie weder Inhalt einschließen noch über schließende Tags verfügen. Wenn es sich bei dem `happy`-Tag um ein zeichenfolgenbegrenztes Tag handelte, müsste die Tag-Spezifikation die Attribute `start_string` und `end_string` umfassen. Bei einem ASP-Tag handelt es sich um ein zeichenfolgenbegrenztes Tag. Es beginnt mit der Zeichenfolge `<%`, endet mit der Zeichenfolge `%>` und enthält kein schließendes Tag.

Die folgenden Informationen beschreiben die Attribute und gültigen Werte für das Tag `tagspec`. Attribute, die mit einem Sternchen (*) gekennzeichnet sind, werden bei zeichenfolgenbegrenzten Tags ignoriert. Optionale Attribute werden in den Attributlisten durch geschweifte Klammern ({}) markiert. Attribute, die nicht durch geschweifte Klammern markiert sind, sind obligatorisch.

<tagspec>**Beschreibung**

Enthält Informationen über das Drittanbieter-Tag.

Attribute

`tag_name`, `{tag_type}`, `{render_contents}`, `{content_model}`, `{start_string}`, `{end_string}`, `{detect_in_attribute}`, `{parse_attributes}`, `icon`, `icon_width`, `icon_height`, `{equivalent_tag}`, `{is_visual}`, `{server_model}`

- `tag_name` ist der Name des benutzerdefinierten Tags. Bei zeichenfolgenbegrenzten Tags wird `tag_name` nur verwendet, um festzulegen, ob ein bestimmter Eigenschafteninspektor für das Tag verwendet werden kann. Wenn die erste Zeile des Eigenschafteninspektors diesen Tag-Namen mit einem Sternchen auf beiden Seiten enthält, kann der Inspektor für Tags dieses Typs verwendet werden. So lautet beispielsweise der Tag-Name für ASP-Code `ASP` Eigenschafteninspektoren, die ASP-Code untersuchen können, müssen in der ersten Zeile den Eintrag `*ASP*` aufweisen. Informationen über die Eigenschafteninspektor-API finden Sie in „Eigenschafteninspektoren“ auf Seite 201.
- `tag_type` legt fest, ob das Tag leer ist (wie das Tag `img`), oder ob sich zwischen den öffnenden und schließenden Tags Daten befinden (wie bei dem Tag `code`). Dieses Attribut wird für normale (nicht zeichenfolgenbegrenzte) Tags benötigt. Bei zeichenfolgenbegrenzten Tags wird es ignoriert, da diese immer leer sind. Gültige Werte sind `"empty"` und `"nonempty"`.
- `render_contents` legt fest, ob der Inhalt des Tags in der Entwurfsansicht angezeigt wird oder ob stattdessen das angegebene Symbol angezeigt wird. Dieses Attribut ist bei nicht leeren Tags erforderlich und wird bei leeren Tags ignoriert. (Leere Tags haben keinen Inhalt.) Dieses Attribut betrifft nur Tags, die außerhalb von Attributen auftreten. Der Inhalt von Tags, die innerhalb der Werte von Attributen anderer Tags auftreten, wird nicht dargestellt. Gültige Werte sind `"true"` und `"false"`.
- `content_model` beschreibt, welche Arten von Inhalt das Tag enthalten kann und wo das Tag in einer HTML-Datei auftreten kann. Gültige Werte sind `"block_model"`, `"head_model"`, `"marker_model"` und `"script_model"`.
 - `block_model` gibt an, dass das Tag Elemente auf Blockebene enthalten kann, wie beispielsweise `div` und `p`. Dieses Tag kann nur im `body`-Abschnitt oder innerhalb anderer `body`-Inhaltstags, wie z. B. `div`, `layer` oder `td`, auftreten.
 - `head_model` gibt an, dass das Tag Text beinhalten und nur im Abschnitt „HEAD“ auftreten kann.
 - `marker_model` gibt an, dass das Tag beliebigen gültigen HTML-Code beinhalten und an beliebiger Stelle in einer HTML-Datei auftreten kann. Der HTML-Validator in Dreamweaver ignoriert Tags, die als `marker_model` angegeben sind. Der Validator ignoriert jedoch nicht den Inhalt dieser Tags. Dies bedeutet, dass der Inhalt des Tags an bestimmten Stellen zu ungültigem HTML-Code führen kann, auch wenn das Tag selbst an beliebiger Stelle auftreten kann. Einfacher Text kann beispielsweise nicht (außerhalb eines gültigen `head`-Elements) im `head`-Abschnitt eines Dokuments auftreten. Sie können also ein `marker_model`-Tag mit einfachem Text nicht im `head`-Abschnitt platzieren. (Um ein benutzerdefiniertes Tag mit einfachem Text im `head`-Abschnitt zu platzieren, geben Sie das Inhaltsmodell des Tags als `head_model` anstatt als `marker_model` an.) Verwenden Sie `marker_model` für Tags, die „inline“ angezeigt werden sollen (innerhalb eines Elements auf Blockebene, wie beispielsweise `p` oder `div`, z. B. innerhalb eines Absatzes). Wenn dieses Tag als eigenständiger Absatz mit Zeilenumbrüchen vor und nach dem Absatz angezeigt werden soll, verwenden Sie dieses Modell nicht.
 - `script_model` ermöglicht, dass das Tag an beliebiger Stelle zwischen den öffnenden und schließenden HTML-Tags des Dokuments platziert werden kann. Wenn Dreamweaver auf ein Tag dieses Modells trifft, wird der gesamte Inhalt des Tags ignoriert. Dieses Tag wird für Markup (etwa bestimmte ColdFusion-Tags) verwendet, das Dreamweaver nicht analysieren soll.
- `start_string` gibt ein Trennzeichen an, das den Anfang eines zeichenfolgenbegrenzten Tags kennzeichnet. Zeichenfolgenbegrenzte Tags können an beliebiger Stelle im Dokument auftreten, an der ein Kommentar angezeigt werden kann. Dreamweaver analysiert die Tags oder dekodiert Entities bzw. URLs zwischen den Tags `start_string` und `end_string` nicht. Dieses Attribut ist erforderlich, wenn `end_string` angegeben ist.
- `end_string` gibt ein Trennzeichen an, das das Ende eines zeichenfolgenbegrenzten Tags kennzeichnet. Dieses Attribut ist erforderlich, wenn `start_string` angegeben ist.

- `detect_in_attribute` gibt an, ob der Inhalt zwischen `start_string` und `end_string` (oder, wenn diese Strings nicht definiert sind, zwischen öffnenden und schließenden Tags) ignoriert werden soll, auch wenn diese Strings innerhalb von Attributnamen oder Werten auftreten. Sie sollten dies in der Regel für zeichenfolgenbegrenzte Tags auf `"true"` setzen. Die Standardeinstellung ist `"false"`. ASP-Tags treten beispielsweise in manchen Fällen innerhalb von Attributwerten auf und enthalten unter Umständen Anführungszeichen ("). Da in der ASP-Tag-Spezifikation `detect_in_attribute="true"` angegeben ist, ignoriert Dreamweaver die ASP-Tags, einschließlich der internen Anführungszeichen, wenn sie innerhalb von Attributwerten auftreten.
- `parse_attributes` gibt an, ob die Attribute des Tags analysiert werden sollen. Wenn das Attribut auf `"true"` (Standardeinstellung) gesetzt ist, analysiert Dreamweaver die Attribute; wenn es auf `"false"` gesetzt ist, ignoriert Dreamweaver alles, bis die nächste schließende spitze Klammer außerhalb von Anführungszeichen auftritt. Dieses Attribut sollte beispielsweise auf `"false"` gesetzt werden für ein Tag wie `cfif` (etwa in `<cfif a is 1>`, das von Dreamweaver als Attributname-Wert-Paar nicht analysiert werden kann).
- `icon` gibt den Pfad und den Dateinamen des Symbols an, das mit dem Tag verknüpft ist. Dieses Attribut ist für leere Tags erforderlich sowie für nicht leere Tags, deren Inhalt nicht in der Entwurfsansicht im Dokumentfenster angezeigt wird.
- `icon_width` gibt die Breite des Symbols in Pixel an.
- `icon_height` gibt die Höhe des Symbols in Pixel an.
- `equivalent_tag` gibt einfache HTML-Entsprechungen für bestimmte formularspezifische ColdFusion-Tags an. Es ist nicht zur Verwendung mit anderen Tags vorgesehen.
- `is_visual` gibt an, ob ein Tag eine direkt sichtbare Auswirkung auf die Seite hat. Das ColdFusion-Tag `cfgraph` gibt beispielsweise keinen Wert für `is_visual` an (d. h. der Standardwert lautet `"true"`). Für das ColdFusion-Tag `cfset` ist angegeben, dass `is_visual` auf `"false"` gesetzt ist. Die Sichtbarkeit von Server-Markup-Tags wird durch die Kategorie **Unsichtbare Elemente** im Dialogfeld **Voreinstellungen** gesteuert. Die Sichtbarkeit von Server-Markup-Tags kann unabhängig von der Sichtbarkeit unsichtbarer Server-Markup-Tags festgelegt werden.
- `server_model` gibt, sofern festgelegt, an, dass das Tag `tagspec` nur für die Seiten Anwendung findet, die zum angegebenen Servermodell gehören. Wenn `server_model` nicht angegeben ist, wird das Tag `tagspec` auf alle Seiten angewendet. Die Trennzeichen für ASP- und JSP-Tags sind beispielsweise identisch, das Tag `tagspec` für JSP gibt jedoch für `server_model` den Wert `"JSP"` an. Wenn also Dreamweaver auf einer JSP-Seite auf Code mit entsprechenden Trennzeichen trifft, zeigt es ein JSP-Symbol an. Wenn das Programm auf solchen Code auf einer Seite trifft, die keine JSP-Seite ist, zeigt es ein ASP-Symbol an.

Inhalt

Keiner (leeres Tag)

Container

Keiner

Beispiel

```
<tagsec tag_name="happy" tag_type="nonempty" render_contents="false" content_model="marker_model" icon="happy.gif" icon_width="18" icon_height="18"></tagsec>
```

Anzeige von benutzerdefinierten Tags in der Entwurfsansicht

Wie benutzerdefinierte Tags in der Entwurfsansicht des Dokumentfensters angezeigt werden, hängt von den Werten der Attribute `tag_type` und `render_contents` des Tags `tagsec` ab. Besitzt `tag_type` den Wert `"empty"`, wird das im Attribut `icon` angegebene Symbol angezeigt. Besitzt `tag_type` den Wert `"nonempty"`, `render_contents` aber den Wert `"false"`, wird das Symbol gleich angezeigt wie für ein leeres Tag. Im folgenden Beispiel ist dargestellt, wie eine Instanz des zuvor definierten Tags `happy` in HTML angezeigt werden kann:

```
<p>This is a paragraph that includes an instance of the <code>happy</code> tag (<happy>Joe</happy>).</p>
```

Da `render_contents` in der Tag-Spezifikation auf `"false"` gesetzt ist, wird der Inhalt des Tags `happy` (das Wort `Joe`) nicht wiedergegeben. Stattdessen werden das Anfangs- und Endtag und ihr Inhalt als einzelnes Symbol angezeigt.

Bei nicht leeren Tags, für die der Wert für `render_contents` auf "true" gesetzt ist, wird das Symbol nicht in der Entwurfsansicht angezeigt. Stattdessen wird der Inhalt zwischen den öffnenden und schließenden Tags angezeigt (wie z. B. der Text zwischen den Tags in `<mytag>Dies ist der Inhalt zwischen den öffnenden und schließenden Tags</mytag>`). Wenn **Ansicht > Unsichtbare Elemente** aktiviert ist, wird der Inhalt in der Farbe markiert, die für Drittanbieter-Tags in den Markierungseinstellungen festgelegt ist. (Markierungen werden nur auf Tags angewendet, die in Tag-Datenbankdateien angegeben sind.)

So ändern Sie die Markierungsfarbe für Drittanbieter-Tags:

- 1 Wählen Sie **Bearbeiten > Voreinstellungen**, und wählen Sie die Kategorie **Markierung** aus.
- 2 Klicken Sie in das Farbfeld für **Drittanbieter-Tags**, um die Farbauswahl anzuzeigen.
- 3 Wählen Sie eine Farbe aus, und klicken Sie auf **OK**, um das Dialogfeld **Voreinstellungen** zu schließen. Weitere Informationen zur Farbauswahl finden Sie in *Dreamweaver verwenden*.

Umschreiben von Drittanbieter-Tags vermeiden

Dreamweaver berichtigt bestimmte Fehlertypen im HTML-Code. Ausführliche Informationen hierzu finden Sie in *Dreamweaver verwenden*. Standardmäßig ändert Dreamweaver den HTML-Code in Dateien mit bestimmten Dateierweiterungen nicht, darunter .asp (ASP), .cfm (ColdFusion), .jsp (JSP) und .php (PHP). Diese Standardeinstellung soll verhindern, dass Dreamweaver versehentlich Code ändert, der in diesen Nicht-HTML-Tags enthalten ist. Sie können das Dreamweaver-Standardverhalten für das Umschreiben ändern, sodass HTML-Code beim Öffnen der oben genannten Dateien umgeschrieben wird, und Sie können weitere Dateitypen hinzufügen, die Dreamweaver nicht umschreiben soll.

Dreamweaver kodiert bestimmte Sonderzeichen, indem das Programm sie durch numerische Werte ersetzt, wenn Sie sie im Eigenschafteninspektor eingeben. Normalerweise ist es empfehlenswert, diese Kodierung durch Dreamweaver zuzulassen, da auf diese Weise eher gewährleistet ist, dass Sonderzeichen über Plattformen und Browser hinweg korrekt angezeigt werden. Sie können das Dreamweaver-Kodierungsverhalten jedoch ändern, da es bei der Arbeit mit entsprechenden Dateien unter Umständen die -Drittanbieter-Tags beeinträchtigen kann.

So ermöglichen Sie, dass Dreamweaver HTML-Code in weiteren Dateitypen umschreibt:

- 1 Wählen Sie **Bearbeiten > Voreinstellungen**, und klicken Sie dann auf die Kategorie **Codeumschreibung**.
- 2 Wählen Sie eine der folgenden Optionen aus:
 - Falsch verschachtelte und nicht geschlossene Tags reparieren
 - Überzählige Schluss-Tags entfernen
- 3 Führen Sie einen der folgenden Schritte aus:
 - Löschen Sie eine oder mehrere Erweiterungen aus der Liste unter **Code niemals umschreiben: in Dateien mit den Erweiterungen**.
 - Deaktivieren Sie die Option **Code niemals umschreiben: in Dateien mit den Erweiterungen**. (Wenn diese Option deaktiviert ist, kann Dreamweaver HTML-Code in allen Dateitypen umschreiben.)

So fügen Sie Dateitypen hinzu, die Dreamweaver nicht umschreiben soll:

- 1 Wählen Sie **Bearbeiten > Voreinstellungen**, und klicken Sie dann auf die Kategorie **Codeumschreibung**.
- 2 Wählen Sie eine der folgenden Optionen aus:
 - Falsch verschachtelte und nicht geschlossene Tags reparieren
 - Überzählige Schluss-Tags entfernen
- 3 Stellen Sie sicher, dass die Option **Code niemals umschreiben: in Dateien mit den Erweiterungen** ausgewählt ist, und fügen Sie die neuen Dateierweiterungen zur Liste im Textfeld hinzu.

Wenn der Dateityp nicht im Popupmenü im Dialogfeld **Öffnen (Datei > Öffnen)** angezeigt wird, können Sie ihn zur Datei **Extensions.txt** im Ordner **Configuration** hinzufügen. Weitere Informationen finden Sie unter „Standard-Dateityp ändern“ auf Seite 7.

So deaktivieren Sie die Dreamweaver-Kodierungsoptionen:

- 1 Wählen Sie **Bearbeiten > Voreinstellungen**, und klicken Sie dann auf die Kategorie **Codeumschreibung**.
- 2 Deaktivieren Sie eine oder beide Sonderzeichenoptionen.

Informationen über weitere Einstellungen für die Codeumschreibung finden Sie in *Dreamweaver verwenden*.

Dreamweaver in einer Mehrbenutzerumgebung anpassen

Sie können Dreamweaver unter einem Mehrbenutzer-Betriebssystem wie Microsoft® Windows® 2000, Windows XP oder Mac OS® X anpassen. Dreamweaver verhindert, dass die angepasste Konfiguration eines Benutzers sich auf die angepasste Konfiguration anderer Benutzer auswirkt. Hierzu kopiert Dreamweaver verschiedene Konfigurationsdateien in den Konfigurationsordner eines Benutzers, wenn Sie Dreamweaver das erste Mal in einem der kompatiblen Mehrbenutzer-Betriebssysteme ausführen. Wenn Sie Dreamweaver mithilfe von Dialog- und Bedienfeldern anpassen, ändert die Anwendung Ihre Benutzer-Konfigurationsdateien und nicht die Konfigurationsdateien von Dreamweaver. Um Dreamweaver durch die Bearbeitung einer Konfigurationsdatei in einer Mehrbenutzerumgebung anzupassen, bearbeiten Sie die entsprechende Benutzer-Konfigurationsdatei und nicht die Dateien im Ordner **Configuration** von Dreamweaver. Um eine Änderung vorzunehmen, die sich auf die meisten Benutzer auswirkt, können Sie eine Dreamweaver-Konfigurationsdatei bearbeiten. Auf Benutzer, die bereits über eine entsprechende Benutzer-Konfigurationsdatei verfügen, wirkt sich diese Änderung jedoch nicht aus. Wenn Sie eine Änderung vornehmen möchten, die sich auf alle Benutzer auswirkt, empfiehlt es sich, eine Erweiterung zu erstellen und diese mithilfe des Extension Managers zu installieren.

Hinweis: In älteren Betriebssystemen (Windows 98, Windows Me oder Mac OS 9.x) verwenden alle Benutzer eine einheitliche Gruppe von Dreamweaver-Konfigurationsdateien, auch wenn das Betriebssystem selbst für mehrere Benutzer konfiguriert ist.

Der Speicherort des Ordners **Configuration** auf dem Computer des Benutzers richtet sich nach dem Betriebssystem.

Bei Windows 2000 und Windows XP befindet sich der Ordner unter:

Festplatte: \Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Adobe\Dreamweaver 9\Configuration

Hinweis: In Windows XP befindet sich dieser Ordner möglicherweise in einem versteckten Ordner.

Für Mac OS X-Plattformen gilt der folgende Speicherort:

Festplatte: Benutzer/Benutzername/Library/Application Support/Adobe/Dreamweaver 9/Configuration

Hinweis: Wenn Sie Erweiterungen installieren möchten, die alle Benutzer in einem Mehrbenutzer-Betriebssystem verwenden können, müssen Sie sich als Administrator (Windows) bzw. als „root“ (Mac OS X) anmelden.

Bei der ersten Ausführung von Dreamweaver werden nur einige der Konfigurationsdateien in Ihren Konfigurationsordner kopiert. (Die zu kopierenden Dateien sind im Ordner **Configuration** in der Datei **version.xml** angegeben.) Wenn Sie Dreamweaver innerhalb der Anwendung anpassen (wenn Sie zum Beispiel eines der vorentwickelten Codefragmente im Bedienfeld **Codefragmente** ändern), kopiert Dreamweaver die relevanten Dateien in Ihren Benutzer-Konfigurationsordner. Die Version einer Datei, die sich in Ihrem Benutzer-Konfigurationsordner befindet, hat stets Vorrang vor der Version im Ordner **Configuration** von Dreamweaver. Um eine Konfigurationsdatei anzupassen, die Dreamweaver nicht in den Configuration-Ordner im Benutzerverzeichnis kopiert hat, müssen Sie die Datei zunächst aus dem Configuration-Ordner im Programmverzeichnis an den entsprechenden Speicherort im Configuration-Ordner im Benutzerverzeichnis kopieren. Anschließend können Sie die Kopie in Ihrem Konfigurationsordner bearbeiten.

Konfigurationsdateien in einer Mehrbenutzerumgebung löschen

Wenn Sie unter einem Mehrbenutzer-Betriebssystem in Dreamweaver einen Vorgang durchführen, durch den eine Konfigurationsdatei gelöscht würde (z. B. durch Löschen eines vorentwickelten Codefragments aus dem Bedienfeld **Codefragmente**), erstellt Dreamweaver in Ihrem Konfigurationsordner eine Datei mit dem Namen **mm_deleted_files.xml**. Wenn eine Datei in **mm_deleted_files.xml** aufgeführt ist, verhält sich Dreamweaver so, als ob diese Datei nicht existieren würde.

So deaktivieren Sie eine Konfigurationsdatei:

- 1 Beenden Sie Dreamweaver.
- 2 Bearbeiten Sie die Datei **mm_deleted_files.xml** in Ihrem Benutzer-Konfigurationsordner mithilfe eines Texteditors. Fügen Sie der Datei ein Element-Tag hinzu, das den Pfad der zu deaktivierenden Konfigurationsdatei angibt (relativ zum Dreamweaver-Ordner **Configuration**).

Hinweis: Bearbeiten Sie die Datei **mm_deleted_files.xml** nicht in Dreamweaver.

- 3 Speichern und schließen Sie die Datei **mm_deleted_files.xml**.
- 4 Starten Sie Dreamweaver erneut.

Die Tag-Syntax in der Datei mm_deleted_files.xml

Die Datei **mm_deleted_files.xml** enthält eine strukturierte Elementliste, mit der die von Dreamweaver zu ignorierenden Konfigurationsdateien angegeben werden. Diese Elemente werden durch XML-Tags angegeben, die Sie in einem Texteditor bearbeiten können.

In den nachfolgenden Abschnitten wird die Syntax der Tags in der Datei **mm_deleted_files.xml** beschrieben. Optionale Attribute werden in den Attributlisten durch geschweifte Klammern ({}) markiert. Attribute, die nicht durch geschweifte Klammern markiert sind, sind obligatorisch.

<deleteditems>**Beschreibung**

Container-Tag, das eine Liste von Elementen enthält, die von Dreamweaver als gelöscht behandelt werden sollen.

Attribute

Keine

Inhalt

Dieses Tag muss mindestens ein `item`-Tag enthalten.

Container

Keiner

Beispiel

```
<deleteditems>
<!-- Element-Tags hier einfügen -->
</deleteditems>
```

<item>**Beschreibung**

Gibt eine Konfigurationsdatei an, die Dreamweaver ignorieren soll.

Attribute

`name`

`name attribute` bestimmt den Pfad zur Konfigurationsdatei, relativ zum Ordner **Configuration**. Verwenden Sie in Windows einen umgekehrten Schrägstrich (\), um die einzelnen Teile des Pfads voneinander zu trennen. Verwenden Sie auf dem Macintosh® einen Doppelpunkt (:).

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `deleteditems`-Tag enthalten sein.

Beispiel

```
<item name="snippets\headers\5columnwith4links.csn" />
```

Dreamweaver in einer Mehrbenutzerumgebung neu installieren und deinstallieren

Wenn Sie nach der Installation von Dreamweaver das Programm neu installieren oder auf eine neuere Version aktualisieren, erstellt Dreamweaver automatische Sicherungskopien von vorhandenen Benutzer-Konfigurationsdateien. Wenn Sie in diesen Dateien Benutzeranpassungen vorgenommen haben, können Sie daher weiterhin auf diese Änderungen zugreifen. Wenn Sie Dreamweaver von einem Mehrbenutzersystem deinstallieren (wozu Sie Administratorrechte benötigen), löscht Dreamweaver die einzelnen Benutzer-Konfigurationsorder.

FTP-Zuordnungen ändern

Die Dateien **FTPExtensionMap.txt** (Windows) und **FTPExtensionMapMac.txt** (Macintosh) ordnen Dateierweiterungen zu FTP-Übertragungsmodi (ASCII oder BINARY) zu.

Jede Zeile in diesen beiden Dateien enthält eine Dateierweiterung (wie z. B. GIF) und entweder das Wort „ASCII“ oder „BINARY“, um anzugeben, welcher FTP-Übertragungsmodus bei der Übertragung von Dateien mit der jeweiligen Erweiterung verwendet werden soll. Für den Macintosh enthält jede Zeile außerdem einen Erstellercode (z. B. „DmWr“) und einen Dateityp (z. B. „TEXT“). Wenn Sie auf dem Macintosh eine Datei mit der entsprechenden Dateierweiterung herunterladen, weist Dreamweaver der Datei den entsprechenden Ersteller und Dateityp zu.

Wenn eine zu übertragende Datei keine Dateierweiterung hat, verwendet Dreamweaver den Übertragungsmodus BINARY.

Hinweis: In Dreamweaver können Dateien nicht im Macbinary-Modus übertragen werden. Wenn Sie Dateien im Macbinary-Modus übertragen müssen, sollten Sie einen anderen FTP-Client verwenden.

Das folgende Beispiel zeigt eine Zeile (aus der Macintosh-Datei), die angibt, dass Dateien mit der Erweiterung „.html“ im ASCII-Modus übertragen werden:

```
HTML DmWr TEXT ASCII
```

Sowohl in der Datei **FTPExtensionMap.txt** als auch in der Datei **FTPExtensionMapMac.txt** (Macintosh) werden alle Elemente in einer Zeile durch Tabulatoren getrennt. Die Erweiterung und der Übertragungsmodus werden in Großbuchstaben angegeben.

Bearbeiten Sie die Datei in einem Texteditor, um die Standardeinstellung zu ändern.

So fügen Sie Informationen über eine neue Dateierweiterung hinzu:

- 1 Bearbeiten Sie die Extension-Map-Datei in einem Texteditor.
- 2 Geben Sie in einer leeren Zeile die Dateierweiterung ein (in Großbuchstaben), und drücken Sie die Tabulatortaste.
- 3 Fügen Sie auf dem Macintosh einen Erstellercode, ein Tabstoppsymbol, den Dateityp und ein weiteres Tabstoppsymbol hinzu.
- 4 Geben Sie zum Festlegen des FTP-Übertragungsmodus **ASCII** oder **BINARY** ein.
- 5 Speichern Sie die Datei.

Erweiterbare Dokumenttypen in Dreamweaver

XML bietet ein umfangreiches System zum Definieren komplexer Dokumente und Datenstrukturen. Dreamweaver verwendet mehrere unterschiedliche XML-Schemata zum Organisieren der Informationen über Serververhalten, Tags und Tag-Dialogfelder, Komponenten, Dokumenttypen und weiterführende Informationen.

Wenn Sie in Dreamweaver Erweiterungen erstellen und verwenden, ergeben sich viele Gelegenheiten, zum Verwalten der von Ihrer Erweiterung verwendeten Daten XML-Dateien zu erstellen oder bereits vorhandene zu bearbeiten. In vielen Fällen können Sie eine vorhandene Datei aus dem entsprechenden Unterordner des Konfigurationsordners kopieren und als Vorlage verwenden.

Dokumenttyp-Definitionsdateien

Die zentrale Komponente erweiterbarer Dokumenttypen ist die Dokumenttyp-Definitionsdatei. Es sind unter Umständen mehrere Definitionsdateien vorhanden, die sich alle im Ordner **Configuration\DocumentTypes** befinden. Jede Definitionsdatei enthält Informationen über mindestens einen Dokumenttyp. Für jeden Dokumenttyp sind wichtige Informationen wie Servermodell, Farbkodierungsstil, Beschreibungen usw. angegeben.

Hinweis: Verwechseln Sie die Dreamweaver-Dokumenttyp-Definitionsdateien nicht mit der XML-Dokumenttypdefinition (DTD). Die Dokumenttyp-Definitionsdateien in Dreamweaver enthalten mehrere `documenttype`-Elemente, die jeweils eine vordefinierte und mit einem Dokumenttyp verknüpfte Reihe von Tags und Attributen definieren. Beim Start analysiert Dreamweaver die Dokumenttyp-Definitionsdateien und erstellt eine speicherinterne Datenbank mit Informationen über alle definierten Dokumenttypen.

Dreamweaver enthält eine grundlegende Dokumenttyp-Definitionsdatei. Diese Datei mit dem Namen **MMDocumentTypes.xml** enthält alle von Adobe bereitgestellten Definitionen für Dokumenttypen:

Dokumenttyp	Servermodell	Interner Typ	Dateierweiterungen	Bisheriges Servermodell
ASP.NET C#	ASP.NET-Csharp	Dynamisch	aspx, ascx	
ASP.NET VB	ASP.NET-VB	Dynamisch	aspx, ascx	
ASP JavaScript	ASP-JS	Dynamisch	asp	
ASP VBScript	ASP-VB	Dynamisch	asp	
ColdFusion	ColdFusion	Dynamisch	cfm, cfml	UltraDev 4 ColdFusion
ColdFusion-Komponente		Dynamisch	cfc	
JSP	JSP	Dynamisch	jsp	
PHP	PHP	Dynamisch	php, php3	
Bibliothekselement		DWExtension	lbi	
ASP.NET-C#-Vorlage		DWTemplate	axcs.dwt	
ASP.NET VB-Vorlage		DWTemplate	axvb.dwt	
ASP-JavaScript-Vorlage		DWTemplate	aspjs.dwt	
ASP-VBScript-Vorlage		DWTemplate	aspvb.dwt	
ColdFusion-Vorlage		DWTemplate	cfm.dwt	
HTML-Vorlage		DWTemplate	dwt	
JSP-Vorlage		DWTemplate	jsp.dwt	
PHP-Vorlage		DWTemplate	php.dwt	
HTML		HTML	htm, html	
ActionScript		Text	as	
CSharp		Text	cs	

Dokumenttyp	Servermodell	Interner Typ	Dateierweiterungen	Bisheriges Servermodell
CSS		Text	css	
Java		Text	java	
JavaScript		Text	js	
VB		Text	vb	
VBScript		Text	vbs	
Text		Text	txt	
EDML		XML	edml	
TLD		XML	tld	
VTML		XML	vtm, vtml	
WML		XML	wml	
XML		XML	xml	

Wenn Sie einen neuen Dokumenttyp erstellen müssen, können Sie entweder Ihren Eintrag der von Adobe bereitgestellten Definitionsdatei hinzufügen (**MMDocumentTypes.xml**) oder dem Ordner **Configuration\DocumentTypes** Ihre eigene Definitionsdatei hinzufügen.

Hinweis: Der Unterordner **NewDocuments** befindet sich im Ordner **Configuration\DocumentTypes**. Dieser Unterordner enthält Standardseiten (Vorlagen) für jeden Dokumenttyp.

Aufbau der Dokumenttyp-Definitionsdateien:

Das folgende Beispiel zeigt eine typische Dokumenttyp-Definitionsdatei:

```
<?xml version="1.0" encoding="utf-8"?>
<documenttypes xmlns:MMString="http://www.Macromedia.com/schemes/data/string/">
  <documenttype
    id="dt-ASP-JS"
    servermodel="ASP-JS"
    internaltype="Dynamic"
    winfileextension="asp,htm, html"
    macfileextension="asp, html"
    previewfile="default_aspjs_preview.htm"
    file="default_aspjs.htm"
    priorversionservermodel="UD4-ASP-JS" >
    <title>
      <loadString id="mmdocumenttypes_0title" />
    </title>
    <description>
      <loadString id="mmdocumenttypes_0descr" />
    </description>
  </documenttype>
  ...
</documenttypes>
```

Hinweis: Die Farbkodierung für die Dokumenttypen ist in den XML-Dateien im Ordner **Configuration\CodeColoring** angegeben.

Im vorangegangenen Beispiel kennzeichnet das Element `loadstring` die lokalisierten Strings, die Dreamweaver für Titel und Beschreibung der Dokumente vom Typ ASP-JS verwenden soll. Weitere Informationen über lokalisierte Strings finden Sie im Abschnitt „Lokalisierte Strings“ auf Seite 20.

In der folgenden Tabelle werden die Tags und Attribute erläutert, die Sie innerhalb einer Dokumenttyp-Definitionsdatei verwenden können.

Elementtyp		Erforderlich	Beschreibung
Tag	Attribut		
documenttype (root)		Ja	Übergeordneter Knoten.
	id	Ja	Einheitlicher Bezeichner in allen Dokumenttyp-Definitionsdateien
	servermodel	Nein	Gibt das entsprechende Servermodell an (Groß- und Kleinschreibung ist zu beachten); standardmäßig sind die folgenden Werte gültig: <pre>ASP.NET C# ASP.NET VB ASP VBScript ASP JavaScript ColdFusion JSP PHP MySQL</pre> Beim Aufruf der Funktion <code>getServerModelDisplayName()</code> werden diese Namen zurückgegeben. Implementierungsdateien für Servermodelle sind im Ordner Configuration\ServerModels abgelegt. Die Entwickler von Erweiterungen können neue Servermodelle erstellen und somit diese Liste erweitern.
	internaltype	Ja	Eine grundsätzliche Klassifizierung zur Verarbeitung von Dateien in Dreamweaver. <code>internaltype</code> gibt an, ob für dieses Dokument die Entwurfsansicht aktiviert ist, und regelt Sonderfälle wie Dreamweaver-Vorlagen oder Dreamweaver-Erweiterungen. Folgende Werte sind gültig: <pre>Dynamisch DWExtension (hat bestimmte Anzeigebereiche) DWTemplate (hat bestimmte Anzeigebereiche) HTML HTML4 Text (nur Codeansicht) XHTML1 XML (nur Codeansicht)</pre> Alle servermodellbezogenen Dokumenttypen sollten auf <code>Dynamic</code> verweisen. <code>HTML</code> sollte auf <code>HTML</code> verweisen. Skriptdateien (wie CSS, JS, VB und CS) sollten auf <code>Text</code> verweisen. Wenn <code>internaltype</code> den Wert <code>DWTemplate</code> hat, sollten Sie auch <code>dynamicid</code> angeben. Wenn Sie in diesem Fall <code>dynamicid</code> auslassen, wird der Dokumenttyp der neuen leeren Vorlage, die mithilfe des Dialogfelds Neues Dokument erstellt wird, nicht vom Bedienfeld Serververhalten oder Bindungen erkannt. Instanzen dieser Vorlage sind nichts weiter als eine HTML-Vorlage.
	dynamicid	Nein	Eine Referenz zum eindeutigen Bezeichner eines dynamischen Dokumenttyps. Dieses Attribut besitzt nur dann eine Bedeutung, wenn <code>internaltype DWTemplate</code> lautet. Mithilfe dieses Attributs können Sie eine dynamische Vorlage mit einem dynamischen Dokumenttyp verknüpfen.
	winfileextension	Ja	Die mit dem Dokumenttyp unter Windows verknüpfte Dateierweiterung. Zur Angabe mehrerer Dateierweiterungen verwenden Sie eine mit Kommas getrennte Liste. Die erste Erweiterung in der Liste wird von Dreamweaver verwendet, wenn der Benutzer ein Dokument des Typs <code>documenttype</code> speichert. Wenn zwei nicht-servermodell-bezogene Dokumenttypen die gleiche Dateierweiterung verwenden, übernimmt Dreamweaver den ersten als Dokumenttyp für die Erweiterung.

Elementtyp		Erforderlich	Beschreibung
Tag	Attribut		
	macfileextension	Ja	Die mit dem Dokumenttyp auf dem Macintosh verknüpfte Dateierweiterung. Zur Angabe mehrerer Dateierweiterungen verwenden Sie eine mit Kommas getrennte Liste. Die erste Erweiterung in der Liste wird von Dreamweaver verwendet, wenn der Benutzer ein Dokument des Typs documenttype speichert. Wenn zwei Dokumenttypen, die mit Nicht-Servermodellen verknüpft sind, die gleiche Dateierweiterung verwenden, übernimmt Dreamweaver den ersten als Dokumenttyp für die Erweiterung.
	previewfile	Nein	Die im Vorschaubereich des Dialogfelds Neues Dokument dargestellte Datei.
	file	Ja	Die Datei im Ordner DocumentTypes\NewDocuments , die Vorlageninhalt für neue Dokumente des Typs documenttype enthält.
	priorversionservermodel	Nein	Wenn das Servermodell dieses Dokuments eine Entsprechung in Dreamweaver UltraDev 4 hat, wird der Name der älteren Version des Servermodells angegeben. UltraDev 4 ColdFusion ist ein gültiges früheres Servermodell.
title (Subtag)		Ja	Der String, der im Dialogfeld Neues Dokument unter Leeres Dokument als Kategorieelement angezeigt wird. Sie können diesen String direkt in die Definitionsdatei einfügen oder zu Lokalisierungszwecken indirekt auf ihn verweisen. Weitere Informationen über die Lokalisierung dieses Strings finden Sie im Abschnitt „Lokalisierte Strings“ auf Seite 20. Eine Formatierung ist nicht zulässig; somit können keine HTML-Tags angegeben werden.
description (subtag)		Nein	Der den Dokumenttyp beschreibende String. Sie können diesen String direkt in die Definitionsdatei einfügen oder zu Lokalisierungszwecken indirekt auf ihn verweisen. Weitere Informationen über die Lokalisierung dieses Strings finden Sie im Abschnitt „Lokalisierte Strings“ auf Seite 20. Eine Formatierung ist zulässig; somit können HTML-Tags angegeben werden.

Hinweis: Wenn der Benutzer ein neues Dokument speichert, untersucht Dreamweaver die Liste der mit dem Dokumenttyp verknüpften Erweiterungen der jeweiligen Plattform (winfileextension und macfileextension). Dreamweaver wählt den ersten String in der Liste aus und verwendet ihn als standardmäßige Dateierweiterung. Um diese standardmäßige Dateierweiterung zu ändern, müssen Sie die in der Liste enthaltenen, durch Kommas getrennten Erweiterungen umgruppieren, sodass die neue standardmäßige Dateierweiterung am Listenanfang steht.

Beim Start liest Dreamweaver alle Dokumenttyp-Definitionsdateien und erstellt eine Liste gültiger Dokumenttypen. Dreamweaver betrachtet alle Einträge in den Definitionsdateien, für die keine Servermodelle vorhanden sind, als Dokumenttypen für Nicht-Servermodelle. Dreamweaver ignoriert Einträge mit fehlerhaften Inhalten oder nicht eindeutigen IDs.

Wenn Dreamweaver beim Durchsuchen des Ordners **Configuration\DocumentTypes** keine Dokumenttyp-Definitionsdateien findet oder eine der Definitionsdateien beschädigt zu sein scheint, schließt Dreamweaver den Vorgang mit einer Fehlermeldung ab.

Dynamische Vorlagen

Sie können Vorlagen auf der Grundlage dynamischer Dokumenttypen erstellen. Solche Vorlagen werden als *dynamische Vorlagen* bezeichnet. Die folgenden beiden Elemente sind beim Definieren einer dynamischen Vorlage von zentraler Bedeutung:

- Der Wert des Attributs `internaltype` für den neuen Dokumenttyp muss `DWTemplate` lauten.
- Das Attribut `dynamicid` muss festgelegt sein, und der Wert muss eine Referenz auf den Bezeichner eines vorhandenen dynamischen Dokumenttyps sein.

Das folgende Beispiel definiert einen dynamischen Dokumenttyp:

```
<documenttype
  id="PHP_MySQL"
  servermodel="PHP_MySQL"
  internaltype="Dynamic"
  winfileextension="php,php3"
  macfileextension="php,php3"
  file="Default.php">
  <title>PHP</title>
  <description><![CDATA[PHP document]]></description>
</documenttype>
```

Sie können nun die folgende dynamische Vorlage auf der Grundlage dieses dynamischen Dokumenttyps `PHP_MySQL` definieren:

```
<documenttype
  id="DWTemplate_PHP"
  internaltype="DWTemplate"
  dynamicid="PHP_MySQL"
  winfileextension="php.dwt"
  macfileextension="php.dwt"
  file="Default.php.dwt">
  <title>PHP Template</title>
  <description><![CDATA[Dreamweaver PHP Template document]]></description>
</documenttype>
```

Wenn ein Benutzer von Dreamweaver eine neue leere Vorlage des Typs „DWTemplate_PHP“ erstellt, kann er PHP-Serververhalten in der Datei erstellen. Weiterhin kann der Benutzer, wenn er Instanzen der neuen Vorlage erstellt, PHP-Serververhalten in der Instanz erstellen.

Im vorangegangenen Beispiel fügt Dreamweaver der Datei automatisch die Erweiterung „.php.dwt“ hinzu, sobald der Benutzer die Vorlage speichert. Wenn der Benutzer eine Instanz der Vorlage speichert, hängt Dreamweaver der Datei die Erweiterung „.php“ an.

Dokumenterweiterungen und Dateitypen

Dreamweaver zeigt standardmäßig alle kompatiblen Dateitypen im Dialogfeld **Öffnen (Datei > Öffnen)** an. Nach dem Erstellen eines neuen Dokumenttyps muss der Entwickler der Erweiterung die entsprechende Datei **Extensions.txt** aktualisieren. Wenn der Benutzer auf einem System arbeitet, das mehrere Benutzer unterstützt (wie z. B. Windows XP, Windows 2000 oder Mac OS X), befindet sich eine weitere **Extensions.txt**-Datei im deren Konfigurationsordnern. Der Benutzer muss die Datei **Extensions.txt** aktualisieren, da es sich um die Instanz handelt, die Dreamweaver sucht und analysiert.

Der Speicherort des Ordners **Configuration** auf dem Computer des Benutzers richtet sich nach dem Betriebssystem.

Bei Windows 2000 und Windows XP befindet sich der Ordner unter:

Festplatte: \Dokumente und Einstellungen\Benutzername\Anwendungsdaten\Adobe\Dreamweaver 9\Configuration

Hinweis: In Windows XP befindet sich dieser Ordner möglicherweise in einem versteckten Ordner.

Für Mac OS X-Plattformen gilt der folgende Speicherort:

Festplatte: Benutzer/Benutzername/Library/Application Support/Adobe/Dreamweaver 9/Configuration

Wenn Dreamweaver die Datei **Extensions.txt** nicht im Konfigurationsordner des Benutzers finden kann, wird sie im Konfigurationsordner von Dreamweaver gesucht.

Hinweis: Wenn Sie auf einer Plattform für mehrere Benutzer die Kopie der Datei **Extensions.txt** im Konfigurationsordner von Dreamweaver bearbeiten und nicht die Kopie im Konfigurationsordner des Benutzers, dann ignoriert Dreamweaver die Änderungen, da Dreamweaver die Kopie der Datei **Extensions.txt** im Konfigurationsordner des Benutzers analysiert und nicht diejenige im Konfigurationsordner von Dreamweaver.

Dazu können Sie entweder einem vorhandenen Dokumenttyp die neue Erweiterung hinzufügen oder einen neuen Dokumenttyp erstellen.

So fügen Sie eine neue Erweiterung zu einem vorhandenen Dokumenttyp hinzu:

- 1 Bearbeiten Sie die Datei **MMDocumentTypes.xml**.
- 2 Fügen Sie den Attributen `winfileextension` und `macfileextension` des vorhandenen Dokumenttyps die neue Erweiterung hinzu.

So fügen Sie einen neuen Dokumenttyp hinzu:

- 1 Erstellen Sie eine Sicherungskopie der Datei **Extensions.txt** im Ordner **Configuration**.
- 2 Öffnen Sie die Datei **Extensions.txt** in Dreamweaver oder in einem Texteditor.
- 3 Fügen Sie für jeden neuen Dateityp eine neue Zeile hinzu. Geben Sie die möglichen Dateierweiterungen für einen neuen Dateityp in Großbuchstaben ein und durch Kommas getrennt. Geben Sie dann einen Doppelpunkt und eine kurze Beschreibung ein, die später im Dialogfeld **Öffnen (Datei > Öffnen)** im Popupmenü **Dateityp** angezeigt wird.

Geben Sie für JPEG-Dateien beispielsweise `JPG, JPEG, JFIF:JPEG-Bilddateien` ein.

- 4 Speichern Sie die Datei **Extensions.txt**.
- 5 Starten Sie Dreamweaver neu.

Um die Änderungen zu überprüfen, wählen Sie **Datei > Öffnen**, und klicken auf das Popupmenü **Dateityp**.

So ändern Sie den Standard-Dateityp, den Dreamweaver bei Auswahl von Datei > Öffnen anzeigt:

- 1 Erstellen Sie eine Sicherungskopie der Datei **Extensions.txt** im Ordner **Configuration**.
- 2 Öffnen Sie die Datei **Extensions.txt** in Dreamweaver oder in einem Texteditor.
- 3 Schneiden Sie die Zeile aus, die der neuen Standardeinstellung entspricht, und fügen Sie sie am Anfang der Datei ein, sodass sie sich in der ersten Zeile der Datei befindet.
- 4 Speichern Sie die Datei **Extensions.txt**.
- 5 Starten Sie Dreamweaver neu.

Um die Änderungen zu überprüfen, wählen Sie **Datei > Öffnen**, und klicken auf das Popupmenü **Dateityp**.

Lokalisierte Strings

In einer Dokumenttyp-Definitionsdatei geben die Subtags `<title>` und `<description>` den Anzeigetitel und die Beschreibung des Dokumenttyps an. Sie können die Direktive `MMString:loadstring` in den Subtags als Platzhalter verwenden, um lokalisierte Strings für die beiden Subtags einzufügen. Dieser Vorgang entspricht Serverskripts, bei denen Sie einen bestimmten String zur Verwendung in Ihrer Seite angeben, indem Sie einen Stringbezeichner als Platzhalter verwenden. Für den Platzhalter können Sie ein spezielles Tag verwenden oder ein Tag-Attribut angeben, dessen Wert ersetzt wird.

So fügen Sie lokalisierte Strings ein:

- 1 Fügen Sie am Anfang der Dokumenttyp-Definitionsdatei die folgende Anweisung ein:

```
<?xml version="1.0" encoding="utf-8"?>
```

- 2 Deklarieren Sie den `MMString`-Namensraum im Tag `<documenttypes>`:

```
<documenttypes  
  xmlns:MMString="http://www.Macromedia.com/schemes/data/string/">
```

3 An der Stelle in der Dokumenttyp-Definitionsdatei, an der Sie einen lokalisierten String einfügen möchten, verwenden Sie die Direktive `MMString:loadstring`, um einen Platzhalter für den lokalisierten String zu definieren. Ihnen stehen zwei Möglichkeiten zur Angabe dieses Platzhalters zur Verfügung:

```
<description>
  <loadstring>myJSPDocType/Description</loadstring>
</description>
oder
<description>
  <loadstring id="myJSPDocType/Description" />
</description>
```

In diesen Beispielen ist `myJSPDocType/Description` ein eindeutiger Stringbezeichner, der als Platzhalter für den lokalisierten String dient. Der lokalisierte String wird im nächsten Schritt definiert.

4 Erstellen Sie im Ordner **Configuration\Strings** eine neue XML-Datei (oder bearbeiten Sie eine vorhandene Datei), die den lokalisierten String definiert. Zum Beispiel definiert der folgende Code, wenn er in die Datei **Configuration\Strings\strings.xml** eingefügt wird, den String `myJSPDocType/Description`:

```
<Strings>
...
  <string id="myJSPDocType/Description"
    value=
      "<![CDATA[JavaServer&nbsp;Page with <em>special</em> features]]>"
    />
...
</strings>
```

Hinweis: Stringbezeichner, wie z. B. `myJSPDocType/Description` im vorangegangenen Beispiel, müssen in der Anwendung eindeutig sein. Beim Start analysiert Dreamweaver alle XML-Dateien im Ordner **Configuration\Strings** und lädt diese eindeutigen Strings.

Regeln bezüglich Dokumenttyp-Definitionsdateien

In Dreamweaver können Dokumenttypen, die mit einem Servermodell verknüpft sind, Dateierweiterungen gemeinsam nutzen. Die Dokumenttypen ASP-JS und ASP-VB können beispielsweise beide die Dateierweiterung „.asp“ beanspruchen. (Informationen darüber, welches Servermodell Vorrang hat, finden Sie unter „`canRecognizeDocument()`“ auf Seite 307.)

Für Dokumenttypen hingegen, die nicht mit einem Servermodell verknüpft sind, lassen sich in Dreamweaver Dateierweiterungen nicht gemeinsam nutzen.

Wenn eine Dateierweiterung von zwei Dokumenttypen beansprucht wird, wobei ein Typ mit einem Servermodell verbunden ist und der andere nicht, erhält der letztere Vorrang. Angenommen, Sie haben einen Dokumenttyp namens „SAM“, der nicht mit einem Servermodell verknüpft ist und die Dateierweiterung „.sam“ aufweist, und Sie fügen dem Dokumenttyp ASP-JS diese Dateierweiterung hinzu. Wenn nun ein Benutzer von Dreamweaver eine Datei mit der Erweiterung „.sam“ öffnet, weist Dreamweaver dieser Datei den Dokumenttyp SAM zu und nicht ASP-JS.

Dokumente in Dreamweaver öffnen

Wenn ein Benutzer eine Dokumentdatei öffnet, ermittelt Dreamweaver in einer Folge von Schritten den Dokumenttyp anhand der Dateierweiterung.

Falls Dreamweaver einen eindeutigen Dokumenttyp findet, verwendet Dreamweaver diesen Typ und lädt ggf. das verknüpfte Servermodell für das Dokument, das der Benutzer öffnet. Wenn der Benutzer das Serververhalten von Dreamweaver UltraDev 4 verwendet, lädt Dreamweaver das entsprechende Servermodell von UltraDev 4.

Wenn die Dateierweiterung auf mehr als einen Dokumenttyp verweist, geht Dreamweaver wie folgt vor:

- Falls sich in der Liste mit Dokumenttypen ein statischer Dokumenttyp befindet, erhält dieser Vorrang.

- Wenn alle Dokumenttypen dynamisch sind, wird in Dreamweaver eine alphabetische Liste der mit diesen Dokumenttypen verknüpften Servermodelle erstellt und anschließend in jedem Servermodell die Funktion `canRecognizeDocument()` aufgerufen (siehe „`canRecognizeDocument()`“ auf Seite 307). Dreamweaver nimmt die Rückgabewerte entgegen und ermittelt, welches Servermodell die größte positive Ganzzahl zurückgegeben hat. Der Dokumenttyp, dessen Servermodell die größte Ganzzahl zurückgegeben hat, wird von Dreamweaver dem gerade geöffneten Dokument zugewiesen. Falls mehrere Servermodelle die gleiche Ganzzahl zurückgeben, ermittelt Dreamweaver in der alphabetischen Liste dieser Servermodelle den ersten Eintrag und verwendet diesen Dokumenttyp. Wenn z. B. sowohl ASP-JS als auch ASP-VB Anspruch auf ein .asp-Dokument erheben und ihre jeweiligen Funktionen `canRecognizeDocument()` die gleichen Werte zurückgeben, weist Dreamweaver das Dokument ASP-JS zu (da ASP-JS in alphabetischer Reihenfolge zuerst steht).

Wenn Dreamweaver die Dateierweiterung keinem Dokumenttyp zuordnen kann, wird das Dokument als Textdatei geöffnet.

Arbeitsbereich-Layouts anpassen

In Dreamweaver lässt sich das Arbeitsbereich-Layout anpassen. Sie können festlegen, welche Bedienfelder in dem angegebenen Layout dargestellt werden, ob Bedienfelder ein- oder ausgeblendet werden und ggf. deren Position und Größe einstellen sowie die Position und Größe des Anwendungs- und Dokumentfensters.

Das Arbeitsbereich-Layout wird anhand von XML-Dateien konfiguriert, die im Ordner **Configuration/Workspace** gespeichert werden. Im folgenden Abschnitt wird die Syntax der XML-Tags beschrieben. Optionale Attribute werden in den Attributlisten durch geschweifte Klammern (`{}`) markiert. Attribute, die nicht durch geschweifte Klammern markiert sind, sind obligatorisch.

<panelset>

Beschreibung

Äußerstes Tag, das den Beginn der Beschreibung des Bedienfeldsatzes signalisiert.

Attribute

Keine

Inhalt

Dieses Tag kann eines oder mehrere der `application`-, `document`- und `panelset`-Tags enthalten.

Container

Keiner

Beispiel

```
<panelset>
  <!-- panelset-Tags hier einfügen -->
</panelset>
```

<application>

Beschreibung

Gibt die Ausgangsposition und -größe des Anwendungsfensters an.

Attribute

`rect`, `maximize`

- `rect` gibt die Position und Größe des Anwendungsfensters an. Der String wird in der Form „links oben rechts unten“ mit Ganzzahlen angegeben.
- `maximize` ist ein Boolescher Wert: `true` zum Maximieren des Fensters beim Start, andernfalls `false`. Der Standardwert ist `false`.

Inhalt

Keiner

Container

Dieses Tag muss in einem `panelset`-Tag enthalten sein.

Beispiel

```
<panelset>
  <application rect="0 0 1000 1200" maximize="false">
  </application>
</panelset>
```

<document>**Beschreibung**

Gibt die Ausgangsposition und -größe des Dokumentfensters an.

Attribute

`rect`, `maximize`

- `rect` gibt die Position und Größe des Dokumentfensters an. Der String wird in der Form „links oben rechts unten“ mit Ganzzahlen angegeben. Weist `maximize` den Wert `true` auf, wird der Wert für `rect` ignoriert.
- `maximize` ist ein Boolescher Wert: `true` zum Maximieren des Dokumentfensters beim Start, andernfalls `false`. Der Standardwert ist `false`.

Inhalt

Keiner

Container

Dieses Tag muss in einem `panelset`-Tag enthalten sein.

Beispiel

```
<panelset>
  <document rect="100 257 1043 1200" maximize="false">
  </document>
</panelset>
```

<panelframe>**Beschreibung**

Beschreibt eine ganze Bedienfeldgruppe.

Attribute

`x`, `y`, `{width, height}`, `dock`, `collapse`

- `x` gibt die linke Position der Bedienfeldgruppe an. Der Wert kann entweder eine Ganzzahl oder ein zum Bildschirm relativer Wert sein. Befindet sich der ganzzahlige Wert nicht auf dem Bildschirm, wird die Bedienfeldgruppe an der diesem Wert nächsten Bildschirmposition angezeigt, damit sie sichtbar wird. Bei den relativen Werten kann es sich um „left“ oder „right“ handeln. Diese Werte geben an, welcher Rand der Bedienfeldgruppe an welchem Rand des sichtbaren Bildschirms ausgerichtet werden soll.
- `y` gibt die obere Position der Bedienfeldgruppe an. Der Wert kann entweder eine Ganzzahl oder ein zum Bildschirm relativer Wert sein. Befindet sich der ganzzahlige Wert nicht auf dem Bildschirm, wird die Bedienfeldgruppe an der diesem Wert nächsten Bildschirmposition angezeigt, damit sie sichtbar wird. Bei den relativen Werten kann es sich um „top“ oder „bottom“ handeln. Diese Werte geben an, welcher Rand der Bedienfeldgruppe an welchem Rand des sichtbaren Bildschirms ausgerichtet werden soll.
- `width` ist die Breite der Bedienfeldgruppe in Pixel. Dieses Attribut ist optional. Wenn Sie die Breite nicht angeben, wird der integrierte Standardwert für die Bedienfeldgruppe verwendet.

- `height` ist die Höhe der Bedienfeldgruppe in Pixel. Dieses Attribut ist optional. Wenn Sie die Höhe nicht angeben, wird der integrierte Standardwert für die Bedienfeldgruppe verwendet.
- `dock` ist ein String, der angibt, an welchen Rand des Anwendungsrahmens die Bedienfeldgruppe andockt. Auf dem Macintosh wird dieses Attribut ignoriert, da Bedienfeldgruppen auf dieser Plattform nicht andocken können.
- `collapse` ist ein Boolescher Wert: `true` gibt an, dass die Bedienfeldgruppe ausgeblendet (reduziert) ist; `false` gibt an, dass die Bedienfeldgruppe eingeblendet (erweitert) wird. Dieses Attribut wird auf dem Macintosh ignoriert, da diese Plattform schwebende Bedienfelder verwendet.

Inhalt

Dieses Tag muss mindestens ein `panelcontainer` enthalten.

Container

Dieses Tag muss in einem `panelset`-Tag enthalten sein.

Beispiel

```
<panelset>
  <panelframe rect="196 453 661 987" visible="true" dock="floating">
    <!-- panelcontainer-Tags hier einfügen -->
  </panelframe>
</panelset>
```

<panelcontainer>

Beschreibung

Beschreibt eine ganze Bedienfeldgruppe.

Attribute

`expanded`, `title`, { `height`, } `activepanel`, `visible`, `maximize`, `maxRestorePanel`, `maxRestoreIndex`, `maxRect`, `tabsinheader`

- `expanded` ist ein Boolescher Wert: `true`, wenn das Bedienfeld erweitert ist, und `false`, wenn es reduziert ist.
- `title` ist ein String, der den Bedienfeldtitel angibt.
- `height` ist eine Ganzzahl, mit der die Höhe des Bedienfelds in Pixel festgelegt wird. Dieses Attribut ist optional. Wenn Sie `height` nicht angeben, wird der integrierte Standardwert für jedes Bedienfeld verwendet.

Hinweis: Die Breite wird vom übergeordneten Element übernommen.

- `activepanel` ist eine Zahl, die die Kennung des vorderen Bedienfelds darstellt.
- `visible` ist ein Boolescher Wert: `true`, wenn das Bedienfeld sichtbar ist, andernfalls `false`.
- `maximize` ist ein Boolescher Wert: `true`, wenn das Bedienfeld bei der Einblendung maximiert angezeigt werden soll, andernfalls `false`.
- `maxRestorePanel` ist die Kennzahl des Bedienfelds, das wiederhergestellt werden soll.
- `maxRect` ist ein String, der die Position und Größe des maximierten Bedienfelds angibt. Der String wird in der Form „links oben rechts unten“ mit Ganzzahlen angegeben.
- `tabsinheader` ist ein Boolescher Wert: `true`, wenn die Registerkarten nicht unter, sondern in der Titelleiste platziert werden sollen; andernfalls `false`.

Inhalt

Dieses Tag muss mindestens ein `panel`-Tag enthalten.

Container

Dieses Tag muss in einem `panelframe`-Tag enthalten sein.

Beispiel

```
<panelset>
  <panelframe rect="196 453 661 987" visible="true" dock="floating">
    <panelcontainer title="Color" height="250" visible="true" expanded="true"
      activepanel="20">
      <!-- panel-Tags hier einfügen --->
    </panelcontainer>
  </panelframe>
</panelset>
```

<panel>

Beschreibung

Gibt das im Bedienfeldcontainer enthaltene Bedienfeld an.

Attribute

`id`, `visibleTab`

- `id` ist eine Zahl, die die Kennung des Bedienfelds darstellt. Die folgende Tabelle enthält eine Liste gültiger Werte:

Software	Kennung	Bedienfeld
Adobe® Flash®	1	Eigenschaften
	2	Aktionen
	3	Ausrichten
	4	Verhalten
	5	Komponenten
	6	Komponenten-Inspektor
	7	Farbmischer
	8	Farbfelder
	9	Verlauf
	10	Info
	11	Bibliothek
	12	Film-Explorer
	13	Ausgabe
	14	Eigenschaften
	15	Projekt
	16	Transformieren
	17	Szene
	18	String-Tabelle
	19	Debugger
	101–110	Bibliothek
Dreamweaver	1	Eigenschaften
Flex Builder	1	Eigenschaften

- `visibleTab` ist ein Boolescher Wert: `true`, wenn Registerkarte und Bedienfeld sichtbar sein sollen, andernfalls `false`.

Inhalt

Keiner

Container

Dieses Tag muss in einem `panelcontainer`-Tag enthalten sein.

Beispiel

```

<panelset>
  <panelframe rect="196 453 661 987" visible="true" dock="floating">
    <panelcontainer title="Color" height="250" visible="true" expanded="true"
      activepanel="20">
      <panel id="20"></panel>
    </panelcontainer>
  </panelframe>
</panelset>

```

Symbolleiste in der Codeansicht anpassen

Die Symbolleiste in der Codeansicht umfasst anfänglich 15 Schaltflächen. Dabei handelt es sich um eine Teilmenge der verfügbaren Schaltflächen. Sie können die Code-Symbolleiste anpassen, indem Sie festlegen, dass entweder andere Schaltflächen oder die vorhandenen Schaltflächen in einer anderen Reihenfolge angezeigt werden. Bearbeiten Sie hierzu die Datei **Toolbars.xml** im Ordner **Configuration/Toolbars**. Sie haben auch die Möglichkeit, mit dem Extension Manager eigene Schaltflächen in die Symbolleiste einzufügen.

So ändern Sie die Reihenfolge der Schaltflächen:

- 1 Öffnen Sie die Datei **toolbars.xml** im Ordner **Configuration/Toolbars**.
- 2 Suchen Sie nach folgendem Kommentar, der den Abschnitt über die Symbolleiste einleitet:

```
<!-- Codeansicht-Symbolleiste -->
```

- 3 Bringen Sie die Schaltflächen-Tags durch Kopieren und Einfügen in die gewünschte Reihenfolge.
- 4 Speichern Sie die Datei.

So entfernen Sie eine Schaltfläche:

- 1 Öffnen Sie die Datei **toolbars.xml** im Ordner **Configuration/Toolbars**.
- 2 Suchen Sie nach folgendem Kommentar, der den Abschnitt über die Code-Symbolleiste einleitet:

```
<!-- Codeansicht-Symbolleiste -->
```

- 3 Schließen Sie die zu entfernende Schaltfläche in Kommentarzeichen ein.

Das folgende Beispiel zeigt eine Schaltfläche, die in einen Kommentar eingeschlossen wurde, um nicht in der Symbolleiste sichtbar zu sein.

```

<!-- Schaltfläche aus Code-Symbolleiste entfernen
  <button id="DW_ExpandAll"
    image="Toolbars\images\MM/T_ExpandAll_Sm_N.png"
    disabledImage="Toolbars\images\MM/T_ExpandAll_Sm_D.png"
    tooltip="Alles einblenden"
    domRequired="false"
    enabled="dw.getFocus(true) == 'textView' || dw.getFocus(true) == 'html' -
      "command="if (dw.getFocus(true) == 'textView' || dw.getFocus(true) -
        == 'html') dw.getDocumentDOM().source.expandAllCodeFragments();"
    update="onViewChange" />
-->

```

- 4 Speichern Sie die Datei.

Um eine noch nicht angezeigte Schaltfläche in der Symbolleiste sichtbar zu machen, entfernen Sie in der XML-Datei den Kommentar um die gewünschte Schaltfläche.

Die Zuordnungen von Tastaturbefehlen ändern

Dreamweaver umfasst viele Tastaturbefehle zu Programmfunktionen. Die Standard-Tastaturbefehle sind in der Datei **menus.xml** aufgelistet und gelten für die US-Tastaturbelegung. Aufgrund der Anzahl möglicher Tastaturbefehle in Dreamweaver müssen bestimmte nicht alphanumerische Tastaturbefehle (Zeichen, die nicht a-z oder 0-9 entsprechen) bei internationalen Tastaturen neu zugeordnet werden. Zu diesem Zweck wird Dreamweaver mit einer Reihe von XML-Dateien geliefert, die die Zuordnungen von Tastaturbefehlen bei internationalen Tastaturen definieren. Diese Dateien befinden sich im Ordner **Configuration\Menus\Adaptive Sets**. Stellt Dreamweaver fest, dass an den Computer eine internationale Tastatur angeschlossen ist, so werden die Tastaturbefehle auf die Zuordnungsdatei der betreffenden Tastatur gesetzt. Ist für eine Tastatur keine Zuordnungsdatei verfügbar, werden alle Tastaturbefehle entfernt, die auf der betreffenden Tastatur nicht funktionieren.

Die Namen der Zuordnungsdateien für Tastaturbefehle bestehen meist aus einem Sprachcode aus zwei Buchstaben für die Tastaturbelegung, die sie jeweils repräsentieren. Die Datei für die deutsche Tastaturbelegung hat beispielsweise den Namen `de.xml`. Gibt es in einer Sprache verschiedene Tastaturbelegungen für verschiedene Länder, bestehen die Namen der Zuordnungsdateien aus den zwei Buchstaben für den Sprachcode, dann kommt ein Bindestrich („-“) und dann zwei Buchstaben für den Ländercode. So ist `de-ch.xml` der Dateiname für die schweizerdeutsche Tastaturbelegung. Die zwei Buchstaben für die Sprachcodes sind in der Norm ISO 639 definiert (http://en.wikipedia.org/wiki/List_of_ISO_639_codes) und die Ländercodes in der Norm ISO 3166 (http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2).

Ändert sich die aktive Tastaturspracheinstellung auf Ihrem Computer, so überprüft Dreamweaver, ob für die betreffende Sprache und das Land eine geeignete Zuordnungsdatei für Tastaturbefehle existiert. Dreamweaver sucht zunächst nach einer länderspezifischen Zuordnungsdatei und, wenn keine vorhanden ist, nach einer Datei für die betreffende Sprache. Angenommen, Sie haben eine Tastatur für Schweizerdeutsch an Ihren Computer angeschlossen, so sucht Dreamweaver zuerst nach `de-ch.xml` für eine schweizerdeutsche Tastaturbelegung. Ist eine Datei dieses Namens nicht vorhanden, sucht Dreamweaver nach `de.xml`. In der folgenden Tabelle sind alle mit Dreamweaver bereitgestellten Zuordnungsdateien aufgeführt.

Dateiname	Windows-Plattform	Macintosh-Plattform
ca.xml	Katalanisch (Catalan)	Katalanisches Spanisch
de.xml	Deutsch (Deutschland, Österreich)	Österreich, Deutschland
de-ch.xml	Deutsch (Schweiz)	Schweizerdeutsch
es.xml	Spanisch (Internationale Sortierung) Spanisch (Traditionelle Sortierung)	Spanisch - ISO
fr.xml	Französisch (Frankreich)	Französisch
fr-ca.xml	Französisch (Kanada)	Kanadisch - CSA
fr-ch.xml	Französisch (Schweiz)	Schweizer Französisch
it.xml	Italienisch (Italien) Italienisch (Schweiz)	Italienisch - Pro
it-mac.xml	N/V	Italienisch
ja.xml	Japanisch	Japanisch
nl-be.xml	Holländisch (Belgien) Französisch (Belgien)	Belgisch
zh-cn.xml	Chinesisch (PRC) Chinesisch (Singapur)	Vereinfachtes Chinesisch

Wenn Sie eine Tastaturbelegung verwenden, die von Dreamweaver nicht vorbereitet wurde, können Sie eine eigene Zuordnungsdatei erstellen und im Ordner **Configuration\Menus\Adaptive Sets** ablegen.

So erstellen Sie eine Tastaturzuordnungsdatei:

1 Legen Sie eine Kopie von einer der Tastaturzuordnungsdateien im Ordner **Configuration\Menus\Adaptive Sets** an, und ändern Sie ihren Namen entsprechend der ISO-Sprach- und Ländercodes. Achten Sie darauf, dass die Erweiterung „.xml“ erhalten bleibt.

Beim Erstellen der Kopie verwenden Sie als Basis am besten eine Tastaturzuordnungsdatei, die Ihrer Tastaturbelegung möglichst ähnlich ist. Wenn Sie etwa eine Tastaturzuordnungsdatei für eine Schwedische Tastatur erstellen, empfiehlt es sich, die Datei **de.xml** zu kopieren, da die schwedische Tastaturbelegung der deutschen sehr ähnlich ist.

2 Verschieben Sie die soeben erstellte Tastaturzuordnungsdatei in einen anderen Ordner als **Configuration\Menus\Adaptive Sets**.

3 Öffnen Sie die Tastaturzuordnungsdatei in Dreamweaver.

4 Entfernen Sie unerwünschte oder in Ihrer Tastaturbelegung unbrauchbare Kurzbefehl-Tags und fügen Sie andere hinzu.

Damit Ihnen die Entscheidung leichter fällt, welche Kurzbefehl-Tags Sie ändern wollen, können Sie die für die US-Tastatur gesetzten Kurzbefehle mit den für Ihre Sprache gesetzten vergleichen. (In der nächsten Anleitung wird erläutert, wie Sie die Kurzbefehle aus zwei verschiedenen Tastaturbelegungen vergleichen.)

5 Nachdem Sie Ihre Änderungen an den Tastaturbefehlen vorgenommen haben, speichern Sie die Datei und verschieben sie wieder in den Ordner **Configuration\Menus\Adaptive Sets**.

So können Sie feststellen, welche Tastaturbefehl-Tags Sie am besten ändern:

1 Ändern Sie die aktive Tastaturspracheinstellung auf Ihre Sprache, wenn dies nicht bereits der Fall ist. (Dies wird über das Betriebssystem Ihres Computers eingestellt. Auf Windows-Systemen beispielsweise können Sie die Sprache in der Systemsteuerung auswählen.)

2 Starten Sie den Tastaturbefehl-Editor von Dreamweaver, indem Sie **Bearbeiten > Tastaturbefehle** wählen.

3 Klicken Sie oben rechts im Dialogfeld auf die dritte Symbolschaltfläche (Wenn Sie den Mauszeiger über die Schaltfläche halten, erscheint die Quickinfo **Satz als HTML exportieren**).

Das Dialogfeld **Als HTML-Datei speichern** erscheint. Geben Sie hier einen Namen und Pfad für die Zusammenfassungsdatei mit den Kurzbefehlen für Ihre aktuellen Tastaturbelegung an.

4 Nachdem Sie die Zusammenfassungsdatei gespeichert haben, schließen Sie auch das Dialogfeld **Tastaturbefehle**.

5 Ändern Sie die Belegung Ihrer Tastatur auf die US-Tastatur (über das Betriebssystem Ihres Computers).

6 Starten Sie den Tastaturbefehl-Editor von Dreamweaver, indem Sie **Bearbeiten > Tastaturbefehle** wählen.

7 Klicken Sie oben rechts im Dialogfeld auf die dritte Symbolschaltfläche, um den Satz als HTML-Datei zu exportieren.

8 Nachdem Sie die Zusammenfassungsdatei gespeichert haben, schließen Sie auch das Dialogfeld **Tastaturbefehle**.

9 Sie können jetzt die beiden Zusammenfassungsdateien für Tastaturbefehle ausdrucken und vergleichen, um festzustellen, welche Kurzbefehle Dreamweaver für die Tastaturbelegung Ihrer Sprache entfernt hat. Die folgenden Tastaturbefehle müssen Sie neu zuweisen. Dazu verwenden Sie Tasten, die nur auf der Tastaturbelegung Ihrer Sprache verfügbar sind.

Mithilfe der Informationen, die Sie durch den Vergleich der beiden Dateien erhalten, können Sie Ihre Tastaturzuordnungsdateien aktualisieren, indem Sie für jeden Kurzbefehl, den Sie zuweisen wollen, die Kurzbefehl-Tags hinzufügen oder entfernen.

Die XML-Dateien für die Tastaturzuordnung

Das folgende Beispiel zeigt das Format der französischen Zuordnungsdatei für die Tastaturbelegung (**fr.xml**):

```
<shortcutset language="French">
<shortcut key="Cmd+[ " newkey="Cmd+&ugrave;" />
<shortcut key="Cmd+]" newkey="Cmd+)" />
<shortcut platform="win" key="Cmd+Shift+>" newkey="Cmd+Opt+Shift+," />
<shortcut platform="mac" key="Cmd+Shift+>" newkey="Cmd+<" />
<shortcut platform="win" key="Cmd+Shift+<" newkey="Cmd+Shift+," />
<shortcut key="Cmd+' " newkey="Cmd+Shift+= " />
...
</shortcutset>
```

Und hier die allgemeine Syntax einer XML-Datei für die Tastaturbelegung:

```
<shortcutset language="language_name">
<shortcut key="key_combination" newkey="key_combination" />
<shortcut platform="op_system" key="key_combination" newkey="key_combination" />
</shortcutset>
```

Dabei gilt:

- *language_name* steht für die Sprache der Tastatur, also z. B. Französisch, Spanisch, Deutsch usw.
- *key_combination* bezeichnet den Tastaturbefehl, z. B. Cmd+[(die Strg-Taste auf Windows- bzw. die Befehlstaste auf Macintosh-Systemen und [), Cmd+Shift+> (die Strg- bzw. Befehlstaste und die Umschalttaste und >), Ctrl+\$ (die Strg-Taste und \$).
- *key* bezeichnet den zu ersetzenden Tastaturbefehl.
- *newkey* bezeichnet den Tastaturbefehl, der den Tastaturbefehl *key* ersetzen soll.
- *platform=op_system* bezeichnet das System, für das der Kurzbefehl gilt. Geben Sie entweder win oder mac an. Wird keine Plattform angegeben, gilt der Kurzbefehl für beide Betriebssysteme.

Kapitel 3: Codeansicht anpassen

Adobe Dreamweaver CS3 bietet in der Codeansicht zwei Werkzeuge, mit denen die Eingabe von Code vereinfacht sowie der Code deutlich lesbar und klar formatiert dargestellt wird. Bei diesen beiden Werkzeugen handelt es sich um die Codehinweise und die farbige Codehervorhebung. Zusätzlich überprüft Dreamweaver den eingegebenen Code für die von Ihnen angegebenen Zielbrowser und ermöglicht die Änderung der HTML-Standardformatierung.

Sie können die Codehinweise und die farbige Codehervorhebung durch Ändern der XML-Dateien anpassen, die diese Funktionen implementieren. Sie können zu den Codehinweise-Menüs Optionen hinzufügen, indem Sie Einträge in die Datei **CodeHints.xml** oder **SpryCodeHints.xml** einfügen. Ändern Sie die Codefarbstil-Datei **Colors.xml**, um Farbschemen anzupassen, oder ändern Sie eine der Syntaxdateien für Codefarben, wie z. B. **CodeColoring.xml**, um Code-Farbschemen anzupassen bzw. neu hinzuzufügen. Sie können auch die CSS-Profildatei (Cascading Style Sheet) an den Zielbrowser anpassen, um die Auswertung von CSS-Eigenschaften und -Werten in Dreamweaver zu beeinflussen. Die Änderung der standardmäßigen HTML-Formatierung von Dreamweaver erfolgt im Dialogfeld **Voreinstellungen**. In den folgenden Abschnitten wird die Anpassung dieser Funktion beschrieben.

Codehinweise

Codehinweise sind Menüs, die Dreamweaver einblendet, wenn Sie in der Codeansicht bestimmte Zeichenmuster eingeben. Codehinweise ermöglichen eine schnellere Eingabe, indem sie eine Liste der Strings anbieten, die das von Ihnen eingegebene Zeichenmuster vervollständigen. Wird der String, den Sie gerade eingeben möchten, im Menü angezeigt, können Sie diesen aus der Liste auswählen und die Eingabetaste bzw. den Zeilenschalter betätigen, um Ihre Eingabe zu ergänzen. Wenn Sie beispielsweise < eingeben, zeigt das Popupmenü eine Liste mit Tag-Namen an. Anstatt den Rest des Tag-Namens einzugeben, können Sie das Tag im Menü auswählen, um es in den Text einzufügen. Dreamweaver bietet auch Codehinweise für das Spry-Framework.

Dreamweaver lädt Menüs mit Codehinweisen aus den Dateien **CodeHints.xml** und anderen XML-Dateien im Ordner **Configuration/CodeHints**. Sie können Codehinweise zu Dreamweaver-Menüs hinzufügen, indem Sie sie in Ihren eigenen XML-Dateien definieren. Dazu verwenden Sie das in diesem Abschnitt beschriebene XML-Schema-Format und kopieren diese XML-Dateien in den Ordner **Configuration/CodeHints**.

Nachdem Dreamweaver den Inhalt einer Codehinweisdatei geladen hat, können Sie über JavaScript auch dynamisch neue Codehinweise-Menüs hinzufügen. JavaScript kann beispielsweise die Liste der Sitzungsvariablen im Bedienfeld **Bindungen** auffüllen. Mit demselben Code können Sie auch ein Codehinweise-Menü hinzufügen. Wenn ein Benutzer in der Codeansicht „**session**“ eingibt, zeigt Dreamweaver ein Menü mit Sitzungsvariablen an. Weitere Informationen zum Hinzufügen oder Modifizieren eines Codehinweisenmenüs mit JavaScript finden Sie im *Dreamweaver API-Referenzhandbuch* unter „Codefunktionen“.

Bestimmte Typen von Codehinweise-Menüs können in Dreamweaver nicht über die XML-Datei oder die JavaScript-API dargestellt werden. Die Dateien **CodeHints.xml** und **SpryCodeHints.xml** sowie die JavaScript-API stellen eine Teilmenge des Codehinweise-Moduls bereit. Die gesamte Dreamweaver-Funktionalität ist darüber jedoch nicht verfügbar. Beispielsweise gibt es keine JavaScript-Funktion zum Einblenden der Farbauswahl. Somit kann Dreamweaver das Menü **Attribute** nicht über JavaScript darstellen. Sie können lediglich ein Menü mit Textelementen einblenden, mit dessen Hilfe Sie Text einfügen können.

Hinweis: Wenn Sie auf diese Weise Text einfügen, wird die Einfügemarke nach dem eingefügten String angezeigt.

Die Datei CodeHints.xml

Die Datei **CodeHints.xml** enthält die folgenden Komponenten:

- Eine Liste aller Menügruppen

Dreamweaver zeigt die Liste der Menügruppen an, wenn Sie im Dialogfeld **Voreinstellungen** die Kategorie **Codehinweise** auswählen. Sie können das Dialogfeld **Voreinstellungen** öffnen, indem Sie **Bearbeiten > Voreinstellungen** wählen. Dreamweaver MX verfügt über die folgenden Menügruppen bzw. -typen von Codehinweise-Menüs: **Tag-Namen, Attributnamen, Attributwerte, Funktionsargumente, Objektmethoden und Variablen, HTML-Entities** und **Namen von CSS-Eigenschaften**.

- Die Beschreibungen der einzelnen Menügruppen

Die Beschreibung wird im Dialogfeld **Voreinstellungen** für die Kategorie **Codehinweise** angezeigt, wenn Sie die Menügruppe in der Liste auswählen. Die Beschreibung für den ausgewählten Eintrag wird unterhalb der Menügruppenliste angezeigt.

- Codehinweise-Menüs

Für ein Menü benötigen Sie ein Muster, das das Menü **Codehinweise** aufruft, sowie eine Liste von Befehlen. Beispielsweise kann ein Muster wie **&** ein Menü wie `&, > , < ;` aufrufen.

Das folgende Beispiel zeigt das Format der Datei **CodeHints.xml** (Die fett ausgezeichneten Tags werden in „Die Datei CodeHints.xml“ auf Seite 32 beschrieben):

```
<codehints>
<menugroup name="HTML Entities" enabled="true" id="CodeHints_HTML_Entities">
  <description>
    <![CDATA[ When you type a '&', a drop-down menu shows
      a list of HTML entities. The list of HTML entities
      is stored in Configuration/CodeHints.xml. ]]>
  </description>

  <menu pattern="&amp; ">
    <menuitem value="&amp;amp;" texticon="&amp;"/>
    <menuitem value="&amp;lt; ;" icon="lessThan.gif"/>
  </menu>
</menugroup>

<menugroup name="Tag Names" enabled="true" id="CodeHints_Tag_Names">
  <description>
    <![CDATA[ When you type '<', a drop-down menu shows
      all possible tag names. You can edit the list of tag
      names using the
      <a href="javascript:dw.popupTagLibraryEditor()"> Tag Library
      Editor </a>]]>
  </description>
</menugroup>

<menugroup name="Function Arguments" enabled="true"
  id="CodeHints_Function_Arguments">
  <description>
    ...
  </description>
  <function pattern="ArraySort(array, sort_type, sort_order)"
    doctypes="CFML"/>
  <function pattern="Response.addCookie(Cookie cookie)"
    doctypes="JSP"/>
</menugroup>
</codehints>
```

JavaScript-Codehinweise

Dreamweaver unterstützt auch Codehinweise für das Spry-Framework. Die Datei **SpryCodeHints.xml** besitzt dasselbe grundlegende Format wie die Datei **CodeHints.xml**. Sie verwendet bestimmte neue Schlüsselwörter wie `method`, und es gibt das neue Attribut `classpattern`, mit dem sich die Liste der Klassenmitglieder mit der Klasse verbinden lässt (zum Beispiel `Spry.Data.XMLDataSet`). Die Klassenmitgliederliste für die Klassen ist innerhalb des Menüs verschachtelt (Methoden, Eigenschaften und Ereignisse).

Das Tag `<method>` und seine Attribute entsprechen dem Tag `function` und seinen Attributen, doch muss das übergeordnete Tag `menu` das Attribut `classpattern` besitzen, um die Verbindung voranzutreiben. Außerdem gibt es ein `property`-Tag für Eigenschaften und ein `event`-Tag für Ereignisse. Sie werden durch die entsprechenden Symbole im Codehinweis-Popupmenü repräsentiert. Es gibt zudem die Tags `parammenu` und `parammenuitem` für die Unterstützung von Parameterhinweisen.

Das folgende Beispiel zeigt das Format einer `Spry-CodeHints.xml`-Datei. (Die fett ausgezeichneten Tags werden in „Die Datei `CodeHints.xml`“ auf Seite 32 beschrieben.):

```
<function pattern="XMLDataSet(xmlsource, xpath, {options})"
  caseSensitive="true" />
<menu classpattern="Spry.Data.XMLDataSet">
  <property pattern="url" icon="shared/mm/images/hintMisc.gif" />
  <property pattern="xpath" icon="shared/mm/images/hintMisc.gif" />
  ...
  ...
  <method pattern="getData()" icon="shared/mm/images/hintMisc.gif" />
  <method pattern="getData()" icon="shared/mm/images/hintMisc.gif" />
  <method pattern="loadData()" icon="shared/mm/images/hintMisc.gif" />
  <method pattern="getCurrentRow()" icon= ".../hintMisc.gif" />
  <method pattern="setCurrentRow(rowID)" icon= ".../hintMisc.gif" />
  <method pattern="setCurrentRowNumber(rowNumber)" icon= ".../hintMisc.gif" />
  <method pattern="getRowNumber(rowObj)" icon= ".../hintMisc.gif" />
  <method pattern="setColumnType(columnName, columnType)" icon= ".../hintMisc.gif" />
    <parammenu pattern="" name="columnName" index="0" type="spryDataReferences">
      </parammenu>
    <parammenu pattern="" name="columnType" index="1" type="enumerated">
      <parammenuitem label="integer" value="integer" icon=".../hintMisc.gif"/>
      <parammenuitem label="image" value="image" icon=".../hintMisc.gif"/>
      <parammenuitem label="date" value="date" icon=".../hintMisc.gif"/>
      <parammenuitem label="string" value="string" icon=".../hintMisc.gif"/>
    </parammenu>
  </method>
  <method pattern="getColumnType(columnName)" icon= ".../hintMisc.gif" />
  <method pattern="distinct()" icon= ".../hintMisc.gif" />
  <method pattern="getSortColumn()" icon= ".../hintMisc.gif" />
  <method pattern="sort()" icon= ".../hintMisc.gif" />
  ...
  ...
  <event pattern="onCurrentRowChanged" icon= ".../hintMisc.gif" />
  <event pattern="onDataChanged" icon= ".../hintMisc.gif" />
  ...
  ...
</menu>
<function pattern="Accordion(element,{options})" caseSensitive="true" />
<menu classpattern="Spry.Widget.Accordion">
  <method pattern="openNextPanel()" icon= ".../hintMisc.gif" />
  <method pattern="openPreviousPanel()" icon= ".../hintMisc.gif" />
  <method pattern="openFirstPanel()" icon= ".../hintMisc.gif" />
  ...
  ...
</menu>
</function>
<function pattern="XMLDataSet(xmlsource, xpath, {options})" caseSensitive="true">
  <parammenu pattern='{,' name="options" index="2" type="optionArray"
    allowwhitespaceprefix="true">
    <parammenuitem label="sortOnLoad" value="sortOnLoad:"
```

```

        icon="shared/mm/images/hintMisc.gif" datatype="string"/>
    <optionparammenu pattern="sortOrderOnLoad" label="sortOrderOnLoad"
        value="sortOrderOnLoad;" icon="shared/mm/images/hintMisc.gif"
        type="enumerated" datatype="string">
        <optionparammenuitem label="ascending" value="ascending"
            icon="shared/mm/images/hintMisc.gif"/>
        <optionparammenuitem label="descending" value="descending"
            icon="shared/mm/images/hintMisc.gif"/>
    </optionparammenu>
</parammenu>
</function>

```

Klassen deklarieren

Das folgende Format deklariert eine Klasse, indem der Klasse eine Variable zugeordnet wird:

```
<variablename> [space] [= operator] [new keyword] [space] <classname>
```

Beispiel:

```

var dsFoo = new Spry.Data.XMLDataSet("products.xml", "products/product");
var fooAccordion = new Spry.Widget.Accordion("accordionDivId");

```

Den Klassennamen `Spry.XML.DataSet` müssen Sie in der Datei **ColorCoding.xml** erneut deklarieren, damit die Farbzustands-Engine erkennt, dass es sich um die Instanz einer Klasse handelt und den auf der linken Seite der Deklaration definierten Variablennamen akzeptiert und ihn zusammen mit den entsprechenden Klassentypen für diese Seite (etwa die Variable `fooAccordion` oder der Klassentyp `Spry.Widget.Accordion` aus dem obigen Beispiel) in der Liste der Variablen speichert.

Die Syntax für die Neudeklaration des Klassennamens in der Datei **CodeColoring.xml** ist wie folgt:

```

<classlibrary name="Spry Keywords" id="CodeColor_JavascriptSpry">
    <class>Spry.Data.XMLDataSet</class>
    <class>Spry.Widget.Accordion</class>
</classlibrary>

```

Dabei gilt:

- `classlibrary` ist ein neues Tag zum Gruppieren der Klassen in der Farb-ID `"CodeColor_JavascriptSpry"`.
- `class` dient zur einzelnen Auflistung jeder in der Klassenbibliothek verfügbaren Klasse. Die Liste der Klassen kann anwachsen, so dass sie auch andere Spry-Klassen aus unterschiedlichen Spry-Paketen enthält (z. B. Debug, Data, Utils, Widgets, Effects) oder andere asynchrone JavaScript- und XML- (bzw. Ajax-) Toolkits oder JavaScript-Bibliotheken.

Codehinweise für Crosstag-Attribute

Dreamweaver bietet Codehinweise für die Spry-Attributenamen und -werte. Da diese Attribute in den verschiedenen Tags immer wieder vorkommen, muss nicht jede *Tag.vtm*-Datei geöffnet und die Spry-Attributliste eingefügt werden, denn Dreamweaver unterstützt ein neues XML-Format für Attributgruppen (etwa `spry:region`, `spry:repeat`) und Tag-Gruppen, das auf eine einzige VTM-Datei mit dem Namen `Spry.vtm` im Ordner `Configuration/TagLibraries` angewendet werden kann.

Das Spry-Attributgruppenformat

Der folgende Code zeigt das Format der VTM-Datei. Dieses Format erlaubt es Ihnen die Attribute festzulegen, die auf bestimmte Tags angewendet werden.

Hinweis: Das Format für Spry-Attributgruppen kann auch außerhalb des Spry-Frameworks verwendet werden.


```

<crosstag_attributes>
  <attribgroup id="group_id_1" name="group_name_1">
    <attrib name = "fooAttr1">
    <attrib name = "barAttr1">
    ...
    <taggroup>
      <tag name = "fooTag1">
      <tag name = "barTag1">
      ...
    </taggroup>
  </attribgroup>
  <attribgroup id="group_id_2" name="group_name_2">
    <attrib name = "fooAttr2">
    <attrib name = "barAttr2">
    ...
    <taggroup>
      <tag name = "fooTag2">
      <tag name = "barTag2">
      ...
    </taggroup>
  </attribgroup>
</crosstag_attributes>

```

Dabei gilt:

- `attribgroup` listet die Attribute der nachfolgenden Tag-Gruppe auf.
- `taggroup` listet die Tags auf, die zu den vorangegangenen Attributen gehören.

Beispiel

```

<crosstag_attributes>
  <attribgroup id="spryRegionAttrs" name="Spry1.2">
    <attrib name="spry:region" type="spryDataSet" allowmultiplevalues="yes"/>
    <attrib name="spry:detailregion" type="spryDataSet" allowmultiplevalues="yes"/>
    <attrib name="spry:content"/>
    <attrib name="spry:if"/>
    <attrib name="spry:choose">
    <attrib name="spry:when"/>
    <attrib name="spry:default"/>
    <attrib name="spry:state" type="Enumerated">
      <attriboption value="read" caption="read"/>
      <attriboption value="loading" caption="loading"/>
      <attriboption value="failed" caption="failed"/>
    </attrib>
    <taggroup>
      <tag name="div"/>
      <tag name="span"/>
      ...
    </taggroup>
  </attribgroup>
  <attribgroup id="spryBehaviorAttrs" name="Spry1.2">
    <attrib name="spry:hover" type="cssStyle"/>
    <attrib name="spry:select" type="cssStyle"/>
    <attrib name="spry:odd" type="cssStyle"/>
    <attrib name="spry:even" type="cssStyle"/>
    <taggroup>
      <tag name="a"/>
      <tag name="abbr"/>
      <tag name="acronym"/>
      ...
    </taggroup>
  </attribgroup>
</crosstag_attributes>

```

Tags für Codehinweise

Die XML-Codehinweisdateien enthalten die folgenden Tags, die Codehinweismenüs definieren, mit denen Sie weitere Codehinweise-Menüs festlegen können:

<codehints>

Beschreibung

Das Tag `codehints` bildet den Stamm der Dateien **CodeHints.xml** und **SpryCodeHints.xml**.

Attribute

Keiner

Inhalt

Eines oder mehrere `menugroup`-Tags

Container

Keine

Beispiel

```
<codehints>
```

<menugroup>

Beschreibung

Jedes `menugroup`-Tag entspricht einem bestimmten Menütyp. Sie können die in Dreamweaver definierten Menütypen anzeigen, indem Sie im Dialogfeld **Voreinstellungen** die Kategorie **Codehinweise** auswählen. Um dieses Dialogfeld anzuzeigen, klicken Sie im Menü **Bearbeiten** auf **Voreinstellungen**.

Sie können eine neue Menügruppe erstellen oder eine vorhandene Gruppe erweitern. Menügruppen sind logische Zusammenstellungen von Menüs, die der Benutzer über das Dialogfeld **Voreinstellungen** ein- oder ausblenden kann.

Attribute

`name`, `enabled`, `id`

- `name` ist der lokalisierte Name, der in der Liste der Menügruppen in der Kategorie **Codehinweise** des Dialogfelds **Voreinstellungen** angezeigt wird.
- `enabled` gibt an, ob die Menügruppe derzeit ausgewählt bzw. aktiviert ist. Eine aktivierte Menügruppe wird in der Kategorie **Codehinweise** des Dialogfelds **Voreinstellungen** durch ein Häkchen gekennzeichnet. Geben Sie den Wert `true` ein, um die Menügruppe zu aktivieren, oder `false`, um die Menügruppe zu deaktivieren.
- `id` ist ein nicht lokalisierter Bezeichner, der sich auf die Menügruppe bezieht.

Inhalt

`description`-, `menu`- und `function`-Tags

Container

Das `codehints`-Tag

Beispiel

```
<menugroup name="Session Variables" enabled="true" id="Session_Code_Hints">
```

<description>

Beschreibung

Das Tag `description` enthält Text, der in Dreamweaver angezeigt wird, wenn Sie die Menügruppe im Dialogfeld **Voreinstellungen** auswählen. Der Beschreibungstext wird unterhalb der Liste der Menügruppen angezeigt. Der Text kann optional ein einzelnes `a`-Tag enthalten. Das zugehörige `href`-Attribut muss eine JavaScript-URL sein, die Dreamweaver aufruft, wenn der Benutzer auf den Hyperlink klickt. Sonderzeichen oder unzulässige Zeichen im String müssen Sie in das Konstrukt „XML CDATA“ einschließen, damit sie von Dreamweaver als Text behandelt werden.

Attribute

Keine

Inhalt

Beschreibungstext

Container

Das `menugroup`-Tag

Beispiel

```
<description>
<![CDATA[ Zum Hinzufügen oder entfernen von Tags und Attributen, verwenden Sie den
    <a href="javascript:dw.tagLibrary.showTagLibraryEditor()">Tag-Bibliothek-Editor</a>]]>
</description>
```

<menu>

Beschreibung

Dieses Tag beschreibt ein einzelnes Popupmenü. Dreamweaver blendet das Menü ein, sobald der Benutzer das letzte Zeichen des Strings im Musterattribut eingegeben hat. Beispielsweise kann ein Menü, das den Inhalt einer Sitzungsvariablen anzeigt, das Musterattribut „Sitzung“ aufweisen.

Attribute

`pattern`, `doctypes`, `casesensitive`, `classpattern`

- Das Attribut `pattern` gibt das Muster der einzugebenden Zeichen an, die in Dreamweaver die Anzeige des Codehinweise-Menüs auslösen. Ist das erste Zeichen des Musters ein Buchstabe, eine Ziffer oder ein Unterstrich, zeigt Dreamweaver das Menü nur an, wenn das dem Muster vorausgehende Zeichen weder ein Buchstabe noch eine Ziffer oder ein Unterstrich ist. Lautet das Muster beispielsweise „Session.“, zeigt Dreamweaver das Menü nicht an, wenn der Benutzer „my_Session.“ eingibt.
- `doctypes` legt fest, dass das Menü nur für die angegebenen Dokumenttypen aktiv ist. Mit diesem Attribut können Sie verschiedene Listen von Funktionsnamen für ASP-JavaScript (ASP-JS), Java Server Pages (JSP), ColdFusion usw. angeben. Sie können `doctypes` als eine durch Kommas getrennte Liste von Dokumenttyp-IDs angeben. Eine Liste der Dreamweaver-Dokumenttypen finden Sie in der Datei **Configuration\DocumentTypes\MMDocumentTypes.xml** im Dreamweaver-Installationsordner.
- `casesensitive` gibt an, ob bei dem Muster zwischen Groß- und Kleinschreibung unterschieden werden soll. Die möglichen Werte von `casesensitive` sind `true`, `false` oder eine Teilmenge der durch Kommas getrennten Liste, die Sie für das Attribut `doctypes` angeben. Über die Liste der Dokumenttypen können Sie festlegen, dass für bestimmte Dokumenttypen zwischen Groß- und Kleinschreibung des Musters unterschieden werden soll, bei anderen jedoch nicht. Bei Auslassung dieses Attributs wird der Wert standardmäßig auf `false` gesetzt. Wenn `casesensitive` auf `true` gesetzt ist, wird das Codehinweise-Menü nur eingeblendet, wenn der vom Benutzer eingegebene Text genau mit dem Musterattribut übereinstimmt. Wenn `casesensitive` den Wert `false` hat, wird das Menü auch eingeblendet, wenn sich die Groß- und Kleinschreibung von Muster und Text unterscheidet.
- Das Attribut `classpattern` verbindet die Klassenmitgliederliste mit der Klasse.

Inhalt

Das `menuitem`-Tag

Container

Das `menugroup`-Tag

Beispiel

```
<menu pattern="CGI." doctypes="ColdFusion">
```

<menuitem>

Beschreibung

Gibt den Text für ein Menüelement in einem Codehinweise-Popupmenü an. Das Tag `menuitem` gibt auch den Wert an, der in den Dokumenttext eingefügt werden soll, wenn Sie das Element auswählen.

Attribute

`label`, `value`, `{icon}`, `{texticon}`

- `label` ist der String, den Dreamweaver im Popupmenü anzeigt.
- `value` ist der String, den Dreamweaver in das Dokument einfügt, wenn Sie den Befehl auswählen. Wenn der Benutzer das Element im Menü auswählt und die Eingabetaste bzw. den Zeilenschalter betätigt, ersetzt Dreamweaver den gesamten Text, den der Benutzer seit dem Einblenden des Menüs eingegeben hat. Da der Benutzer die mit dem Muster übereinstimmenden Zeichen bereits vor dem Einblenden des Menüs eingegeben hat, fügt Dreamweaver diese nicht erneut ein. Möchten Sie beispielsweise `&` eingeben, die HTML-Entität für das Et-Zeichen (&), können Sie die folgenden `menu`- und `menuitem`-Tags definieren:

```
<menu pattern="&";>  
<menuitem label="&"; value="&"; texticon="&"; />
```

Das `value`-Attribut enthält das Et-Zeichen (&) nicht, weil es der Benutzer bereits vor dem Einblenden des Menüs eingegeben hat.

- `icon` ist ein optionales Attribut, das den Pfad zu einer Bilddatei angibt. Dreamweaver zeigt dieses Bild links neben dem Menütext als Symbol an. Der Pfad wird als URL relativ zum Ordner **Configuration** ausgedrückt.
- `texticon` ist ein optionales Attribut, das einen Textstring angibt. Dieser wird im Symbolbereich anstelle einer Bilddatei angezeigt. Dieses Attribut wird für das Menü **HTML-Entities** verwendet.

Inhalt

Keiner

Container

Das `menu`-Tag

Beispiel

```
<menuitem label="CONTENT_TYPE" value=""CONTENT_TYPE";"  
  " icon="shared/mm/images/hintMisc.gif" />
```

<function>

Beschreibung

Wird in der Datei **CodeHints.xml** verwendet. Ersetzt das `menu`-Tag, um Funktionsargumente und Objektmethoden für ein Pop-up-Menü mit Codehinweisen anzugeben. Wenn Sie in der Codeansicht einen Funktions- oder Methodennamen eingeben, zeigt Dreamweaver ein Menü der Funktionsprototypen an, worin das aktuelle Argument fett hervorgehoben ist. Immer wenn Sie ein Komma eingeben, aktualisiert Dreamweaver das Menü und formatiert das nächste Argument fett. Wenn Sie beispielsweise den Funktionsnamen `ArrayAppend` in ein ColdFusion-Dokument eingegeben haben, wird im Codehinweise-Menü `ArrayAppend(array, value)` angezeigt. Nachdem Sie das Komma nach `array` eingegeben haben, wird das Menü aktualisiert und zeigt nun `ArrayAppend(array, value)` an.

Wenn Sie bei Objektmethoden den Objektnamen eingeben, zeigt Dreamweaver ein Menü der Methoden an, die für dieses Objekt definiert sind.

Die erkannten Funktionen werden in den XML-Dateien im Ordner **Configuration\CodeHints** gespeichert.

Attribute

`pattern`, `doctype`s, `casesensitive`

- Das Attribut `pattern` gibt den Namen der Funktion und deren Argumentliste an. Bei Methoden beschreibt das Attribut `pattern` den Namen des Objekts sowie den Namen und die Argumente der Methode. Bei einem Funktionsnamen wird das Codehinweise-Menü eingeblendet, wenn der Benutzer `functionname(` eingibt. Das Menü zeigt die Liste der Argumente für die Funktion an. Bei einer Objektmethode wird das Codehinweise-Menü eingeblendet, wenn der Benutzer `objectname.` eingibt (einschließlich des Punkts). Dieses Menü zeigt die für das Objekt angegebenen Methoden an. Anschließend wird das Codehinweise-Menü mit einer Liste der Argumente für die Methode eingeblendet, wie bei einer Funktion.
- `doctype`s legt fest, dass das Menü nur für die angegebenen Dokumenttypen aktiv ist. Mit diesem Attribut können Sie verschiedene Listen von Funktionsnamen für ASP-JavaScript (ASP-JS), Java Server Pages (JSP), ColdFusion usw. angeben. Sie können `doctype`s als eine durch Kommas getrennte Liste von Dokumenttyp-IDs angeben. Eine Liste der Dokumenttypen von Dreamweaver finden Sie in der Dreamweaver-Datei **Configuration\Documenttypes\MMDocumentTypes.xml**.
- `casesensitive` gibt an, ob bei dem Muster zwischen Groß- und Kleinschreibung unterschieden werden soll. Die möglichen Werte von `casesensitive` sind `true`, `false` oder eine Teilmenge der durch Kommas getrennten Liste, die Sie für das Attribut `doctype`s angeben. Über die Liste der Dokumenttypen können Sie festlegen, dass für bestimmte Dokumenttypen zwischen Groß- und Kleinschreibung des Musters unterschieden werden soll, bei anderen jedoch nicht. Bei Auslassung dieses Attributs wird der Wert standardmäßig auf `false` gesetzt. Wenn `casesensitive` auf `true` gesetzt ist, wird das Codehinweise-Menü nur eingeblendet, wenn der vom Benutzer eingegebene Text genau mit dem Musterattribut übereinstimmt. Wenn `casesensitive` den Wert `false` hat, wird das Menü auch eingeblendet, wenn sich die Groß- und Kleinschreibung von Muster und Text unterscheidet.

Inhalt

Keiner

Container

Das `menugroup`-Tag

Beispiel

```
// Funktionsbeispiel
<function pattern="CreateDate(year, month, day)" DOCTYPES="ColdFusion" />
// Objektmethodenbeispiel
<function pattern="application.getAttribute(String name)" DOCTYPES="JSP" />
```

<method>**Beschreibung**

Wird für das Spry-Framework verwendet. Dieses Tag ersetzt das `menu`-Tag, um Methoden für ein Pop-up-Menü mit Codehinweisen anzugeben. Wenn Sie in der Codeansicht einen Methodennamen eingeben, zeigt Dreamweaver ein Menü der Methodenprototypen mit einer Liste der Parameter für die Funktion an und protokolliert die Parametersequenz, während sie eingefügt werden. Bei Methoden ohne Parameter schließt Dreamweaver den Methodenaufruf, indem die Klammern "()" gesetzt werden.

Das aktuelle Argument ist fett hervorgehoben. Immer wenn Sie ein Komma eingeben, aktualisiert Dreamweaver das Menü und formatiert das nächste Argument fett. Wenn Sie bei Objektmethoden den Objektnamen eingeben, zeigt Dreamweaver ein Menü der Methoden an, die für dieses Objekt definiert sind.

Attribute

`pattern, icon`

- Das Attribut `pattern` gibt den Namen der Methode und deren Argumentliste an. Das Attribut `pattern` beschreibt den Namen des Objekts sowie den Namen und die Argumente der Methode. Das Menü zeigt die Liste der Argumente für die Methode an. Das Codehinweise-Menü wird eingeblendet, wenn der Benutzer `objectname.` eingibt (einschließlich des Punkts). Dieses Menü zeigt die für das Objekt angegebenen Methoden an. Anschließend wird das Codehinweise-Menü mit einer Liste der Argumente für die Methode eingeblendet, wie bei einer Funktion.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das verwendet wird.

Inhalt

Keiner

Container

Das `menu`-Tag

<parammenu>**Beschreibung**

Wird für ein beliebiges (JavaScript-)Objekt verwendet, um Parameterhinweise für die Parameter festzulegen, die die Methode akzeptiert.

Attribute

`pattern, name, index, type`

- Das Attribut `pattern` legt das (die) Zeichen fest, das das Codehinweismenü aufruft. Dieses Argument ist erforderlich.
- Das Attribut `name` legt den Namen des Parameters fest. Dieses Argument ist erforderlich.
- Das Attribut `index` legt die (nullbasierte) Indexnummer des Parameters mit dem Hinweis fest. Dieses Argument ist erforderlich.
- Das Argument `type` legt den Datentyp fest. Folgende Datentypen werden unterstützt:
 - `enumerated` (der Standard) bezeichnet eine Liste der anzuzeigenden verschachtelten `<optionparammenuitem>`-Elemente.
 - `spryDataReferences` bezeichnet eine List von Spry-Datensatzspalten.
 - `cssStyle` bezeichnet eine Liste der für die Seite verfügbaren CSS-Klassen.
 - `cssId` bezeichnet eine Liste der für die Seite verfügbaren CSS-Selektor-ID-Regeln.
 - `optionArray` bezeichnet eine Liste anzuzeigender, verschachtelter `<optionparammenu>`- und `<parammenuitem>`-Elemente (die zur Unterstützung des Optionsarray-Parametertyps dienen).

Inhalt

Keiner

Container

Das Tag `method` oder `function`.

<parammenuitem>

Beschreibung

Wird für ein beliebiges (JavaScript-)Objekt verwendet, um Parameterhinweise für die Parameter festzulegen, die die Methode akzeptiert.

Attribute

`label`, `value`, `icon`, `datatype`

- Das Attribut `label` legt den anzuzeigenden Namen fest. Dieses Argument ist erforderlich.
- Das Attribut `value` bestimmt den Wert, der abgelegt werden soll, wenn das Element im Codehinweisemenü ausgewählt wird. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das im Codehinweisemenü verwendet wird. Dieses Argument ist erforderlich.
- Das Attribut `datatype` erlaubt es Ihnen, `string` festzulegen, d. h. es werden schließende Anführungszeichen hinzugefügt, wenn der Benutzer einen Wert aus dem Codehinweisemenü auswählt. Dieses Argument ist optional.

Inhalt

Keiner

Container

Das `parammenu`-Tag.

<optionparammenu>

Beschreibung

Wird für ein beliebiges (JavaScript-)Objekt verwendet, um Optionsarrayhinweise für die Argumente festzulegen, die die Methode akzeptiert. Ein Optionsarray ist ein Argument, das Unterargumente im Format `Option:Wert` besitzen kann. Die meisten Spryobjekte verwenden ein Optionsarray-Argument, um den Benutzern die Möglichkeit zu geben, das Verhalten eines Objekts (etwa eines Datensatzes, eines Widgets oder eines Effekts) zu konfigurieren. Optionsarrays erscheinen in der Regel im Format `{Option1: Wert1, Option2: Wert2, Option3: Wert3, ...}`

Attribute

`pattern`, `label`, `value`, `icon`, `type`

- Das Attribut `pattern` legt das (die) Zeichen fest, das das Codehinweisemenü aufruft. Dieses Argument ist erforderlich.
- Das Attribut `label` legt den Namen des Parameters fest. Dieses Argument ist erforderlich.
- Das Attribut `value` bestimmt den Wert des Parameters, der eingefügt werden soll, wenn der Benutzer das Element im Codehinweisemenü auswählt. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das verwendet wird. Dieses Argument ist erforderlich.
- Das Argument `type` legt den Datentyp fest. Folgende Datentypen werden unterstützt:
 - `enumerated` (der Standard) bezeichnet eine Liste der anzuzeigenden verschachtelten `optionparammenuitem`-Elemente.
 - `spryDataReferences` bezeichnet eine List von Spry-Datensatzspalten.
 - `cssStyle` bezeichnet eine Liste der für die Seite verfügbaren CSS-Klassen.
 - `cssId` bezeichnet eine Liste der für die Seite verfügbaren CSS-Selektor-ID-Regeln.

Inhalt

Keiner

Container

Das `parammenu`-Tag vom Typ `optionArray`.

<optionparammenuitem>

Beschreibung

Wird für ein beliebiges (JavaScript-)Objekt verwendet, um Parameterhinweise für die Parameter festzulegen, die die Methode akzeptiert.

Attribute

`label`, `value`, `icon`, `datatype`

- Das Attribut `label` legt den anzuzeigenden Namen fest. Dieses Argument ist erforderlich.
- Das Attribut `value` bestimmt den Wert, der abgelegt werden soll, wenn das Element im Codehinweisenmenü ausgewählt wird. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das im Codehinweisenmenü verwendet wird. Dieses Argument ist erforderlich.
- Das Attribut `datatype` erlaubt es Ihnen, `string` festzulegen, d. h. es werden schließende Anführungszeichen hinzugefügt, wenn der Benutzer einen Wert aus dem Codehinweisenmenü auswählt. Dieses Argument ist optional.

Inhalt

Keiner

Container

Das `<optionparammenu>`-Tag.

<property>

Beschreibung

Dieses Tag beschreibt Eigenschaften/Felder eines Objekts und besitzt die folgenden Standardattribute.

Attribute

`label`, `value`, `icon`

- `label` ist der String, den Dreamweaver im Popupmenü anzeigt.
- `value` ist der String, den Dreamweaver in das Dokument einfügt, wenn Sie den Befehl auswählen. Wenn der Benutzer das Element im Menü auswählt und die Eingabetaste bzw. den Zeilenschalter betätigt, ersetzt Dreamweaver den gesamten Text, den der Benutzer seit dem Einblenden des Menüs eingegeben hat. Da der Benutzer die mit dem Muster übereinstimmenden Zeichen bereits vor dem Einblenden des Menüs eingegeben hat, fügt Dreamweaver diese nicht erneut ein.
- `icon` ist ein optionales Attribut, das den Pfad zu einer Bilddatei angibt. Dreamweaver zeigt dieses Bild links neben dem Menütext als Symbol an. Der Pfad wird als URL relativ zum Ordner **Configuration** ausgedrückt.

Inhalt

Keiner

Container

Das `menu`-Tag

Codefarben

Dreamweaver ermöglicht Ihnen das Anpassen oder Erweitern der Code-Farbschemen in der Codeansicht. Sie können neue Schlüsselwörter zu einem Schema hinzufügen oder Code-Farbschemen für neue Dokumenttypen einrichten. Wenn Sie JavaScript-Funktionen zur Verwendung in Ihrem clientseitigen Skript entwickeln, können Sie die Namen dieser Funktionen im Abschnitt keywords hinzufügen, damit sie in der im Dialogfeld **Voreinstellungen** angegebenen Farbe angezeigt werden. Außerdem haben Sie die Möglichkeit, ein Code-Farbschema für einen Dokumenttyp hinzuzufügen, wenn Sie beispielsweise eine neue Programmiersprache für einen Anwendungsserver entwickeln und einen neuen Dokumenttyp verteilen möchten, mit dem Benutzer von Dreamweaver Seiten erstellen können.

Dreamweaver enthält die JavaScript-Funktion `dreamweaver.reloadCodeColoring()`, mit der Sie XML-Dateien für Codefarben, die manuell bearbeitet wurden, neu laden können. Weitere Informationen über diese Funktion finden Sie im *Dreamweaver API-Referenzhandbuch*.

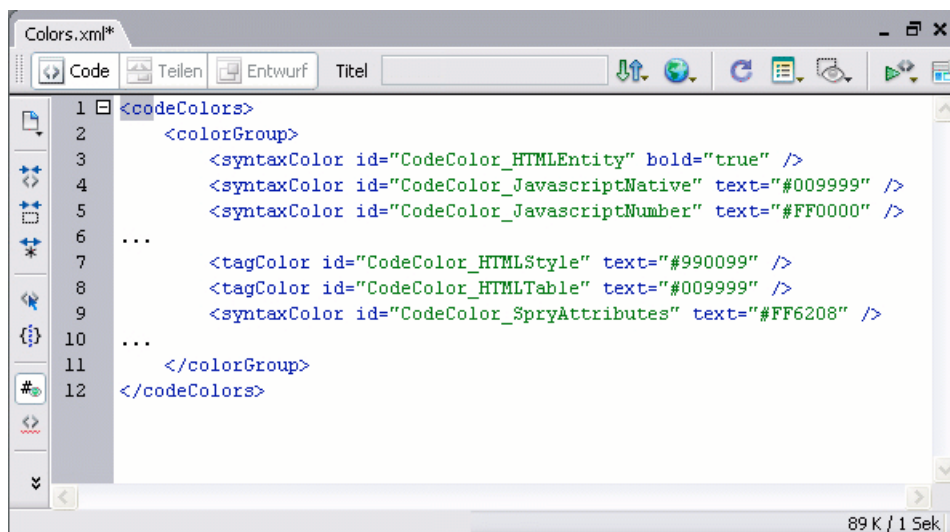
Um ein Code-Farbschema zu aktualisieren oder ein neues Schema hinzuzufügen, bearbeiten Sie die Codefarben-Definitionsdateien.

Farbeinstellungsdateien für Codes

Dreamweaver definiert Codefarbstile und -schemen in XML-Dateien, die sich im Ordner **Configuration\CodeColoring** befinden. Eine Code-Farbstildatei bestimmt die Stile für Felder, die in Syntaxdefinitionen definiert sind. Der Stammknoten ist `<codeColors>`. Eine Code-Farbschemadatei bestimmt die Codefarbensyntax und hat den Stammknoten `<codeColoring>`.

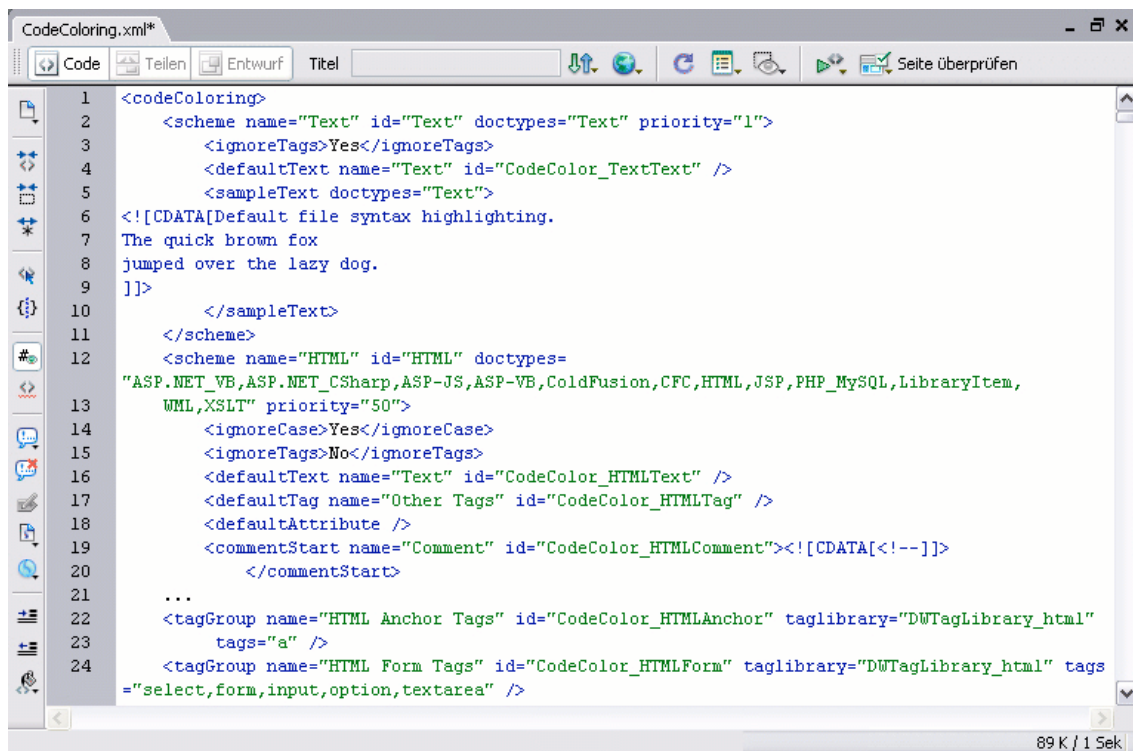
Die von Dreamweaver bereitgestellte Code-Farbstildatei ist **Colors.xml**. Die Code-Farbsyntaxdateien in Dreamweaver sind **CodeColoring.xml**, **ASP JavaScript.xml**, **ASP VBScript.xml**, **ASP.NET CSharp.xml** und **ASP.NET VB.xml**.

Der folgende Auszug aus der Datei **Colors.xml** veranschaulicht die Hierarchie in einer Code-Farbstildatei:



Farben werden als RGB-Hexadezimalwerte ausgedrückt. Die Anweisung `text="#009999"` in dem oben angegebenen XML-Code weist der ID "CodeColor_JavascriptNative" eine blaugüne Farbe zu.

Der folgende Auszug aus der Datei **CodeColoring.xml** veranschaulicht die Tag-Hierarchie in einer Code-Farbschemadatei sowie die Beziehung zwischen den Stildateien und den Schemadateien für die Code-Farben:



Die Tags `syntaxColor` und `tagColor` in der Datei **Colors.xml** ordnen dem `id`-Stringwert Farb- und Stilwerte zu. Der `id`-Wert wird dann in der Datei **CodeColoring.xml** verwendet, um einem `scheme`-Tag einen Stil zuzuordnen. Beispiel: Das Tag `defaultTag` in dem Auszug aus der Datei **CodeColoring.xml** hat den `id`-Wert `"CodeColor_HTMLComment"`. In der Datei **Colors.xml** wird dem `id`-Wert `"CodeColor_HTMLComment"` der `text`-Wert `"#999999"` zugeordnet, also grau.

Dreamweaver enthält die folgenden Code-Farbschemen: Default, HTML, JavaScript, ASP_JavaScript, ASP_VBScript, JSP und ColdFusion. Das Standardschema hat einen `id`-Wert gleich `"Text"`. Dreamweaver verwendet das Standardschema für Dokumenttypen, für die kein Code-Farbschema definiert wurde.

Eine Codefarbendatei enthält die folgenden, unten beschriebenen Tags:

`scheme`, `blockEnd`, `blockStart`, `brackets`, `charStart`, `charEnd`, `charEsc`, `commentStart`, `commentEnd`, `cssImport`, `cssMedia`, `cssProperty`, `cssSelector`, `cssValue`, `defaultAttribute`, `defaultTag`, `defaultText`, `endOfLineComment`, `entity`, `functionKeyword`, `idChar1`, `idCharRest`, `ignoreCase`, `ignoreMMTPParams`, `ignoreTags`, `isLocked`, `keyword`, `keywords`, `numbers`, `operators`, `regexp`, `sampleText`, `searchPattern`, `stringStart`, `stringEnd`, `stringEsc`, `tagGroup`

<scheme>

Beschreibung

Das `scheme`-Tag gibt die Codefarbe für einen Code-Textblock an. Sie können innerhalb einer Datei verschiedene Schemen verwenden, um verschiedene Farben für unterschiedliche Skript- oder Tag-Sprachen einzusetzen. Jedes Schema hat eine eigene Priorität, sodass Sie einen Textblock mit einem Schema in einem Textblock mit einem anderen Schema verschachteln können.

Attribute

name, id, priority, doctypes

- `name="scheme_name"` Ein String, der dem Schema einen Namen zuweist. Der Name des Schemas wird in Dreamweaver im Dialogfeld **Farbschema bearbeiten** angezeigt. Dreamweaver zeigt eine Kombination aus Schemenname und Feldname an, wie z. B. `HTML Comment`. Wenn Sie keinen Namen eingeben, werden die Felder des Schemas nicht im Dialogfeld **Farbschema bearbeiten** aufgeführt. Weitere Informationen über das Dialogfeld „Farbschema bearbeiten“ finden Sie unter „Schemen bearbeiten“ auf Seite 60.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.
- `priority="string"` Die Werte liegen zwischen „1“ und „99“. Die höchste Priorität ist „1“. Gibt die Rangfolge des Schemas an. Blöcke innerhalb von anderen Blöcken mit höherer Priorität werden ignoriert. Blöcke innerhalb von anderen Blöcken mit derselben oder einer niedrigeren Priorität haben Vorrang. Wenn Sie keine Priorität angeben, ist der Standardwert „50“.
- `doctypes="doc_list"` Optional. Gibt eine durch Kommas getrennte Liste von Dokumenttypen an, für die dieses Codefarbschema gilt. Dieser Wert ist notwendig, um Konflikte zu vermeiden, wenn unterschiedliche Anfangs- und Endblöcke dieselben Erweiterungen verwenden.

Inhalt

```
blockEnd, blockStart, brackets, charStart, charEnd, charEsc, commentStart, commentEnd,
cssProperty/, cssSelector/, cssValue/, defaultAttribute, defaultText/, endOfLineComment,
entity/, functionKeyword, idChar1, idCharRest, ignoreCase, ignoreMMTPParam, ignoreTags,
keywords, numbers/, operators, regexp, sampleText, searchPattern, stringStart, stringEnd,
stringEsc, urlProtocol, urlProtocols
```

Container

Das `codeColoring`-Tag

Beispiel

```
<scheme name="Text" id="Text" doctypes="Text" priority="1">
```

<blockEnd>

Beschreibung

Optional. Textwerte, die das Ende des Textblocks für dieses Schema angeben. Die Tags `blockEnd` und `blockStart` müssen paarweise auftreten, und die Kombination muss eindeutig sein. Bei den Werten wird die Groß- und Kleinschreibung nicht berücksichtigt. Der Wert für `blockEnd` kann ein einzelnes Zeichen sein. Es sind mehrere Instanzen dieses Tags zulässig. Weitere Informationen über `blockEnd`-Strings finden Sie im Abschnitt „Platzhalterzeichen“ auf Seite 58.

Attribute

Keine

Beispiel

```
<blockEnd><![CDATA[---]]></blockEnd>
```

<blockStart>

Beschreibung

Optional. Wird nur angegeben, wenn das Farbschema innerhalb eines anderen Farbschemas eingebettet werden kann. Die Tags `blockStart` und `blockEnd` müssen paarweise auftreten, und die Kombination muss eindeutig sein. Bei den Werten wird die Groß- und Kleinschreibung nicht berücksichtigt. Der Wert für `blockStart` muss aus mindestens zwei Zeichen bestehen. Es sind mehrere Instanzen dieses Tags zulässig. Weitere Informationen über `blockStart`-Strings finden Sie im Abschnitt „Platzhalterzeichen“ auf Seite 58. Weitere Informationen über das Attribut `blockStart scheme` finden Sie im Abschnitt „Farben von Schemenblocktrennzeichen“ auf Seite 55.

Attribute

canNest, doctypes, id, name, scheme

- **canNest** Gibt an, ob das Schema in sich selbst verschachtelt werden kann. Zulässige Werte sind `yes` und `no`. Der Standardwert ist `no`.
- **doctypes**="doc_type1, doc_type2,..." Erforderlich. Gibt eine durch Kommas getrennte Liste von Dokumenttypen an, in denen dieses Codefarbschema verschachtelt werden kann. Eine Definition der Dokumenttypen finden Sie in der Dreamweaver-Datei **Configuration\Document Types\MMDocumentTypes.xml**.
- **id**="id_string" Erforderlich, wenn **scheme**="customText". Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.
- **name**="display_name" Ein String, der im Dialogfeld **Farbschema bearbeiten** angezeigt wird, wenn gilt: **scheme**="customText".
- **scheme** Erforderlich. Bestimmt, welche Farbe die Strings `blockStart` und `blockEnd` haben. Weitere Informationen über mögliche Werte für das **scheme**-Attribut finden Sie im Abschnitt „Farben von Schemenblocktrennzeichen“ auf Seite 55.

Beispiel

```
<blockStart doctypes="ColdFusion,CFC" scheme="innerText" canNest="Yes"><![CDATA[<!--]]>
</blockStart>
```

<brackets>

Beschreibung

Eine Liste von Zeichen, die für Klammern stehen.

Attribute

name, id

- **name**="bracket_name" Ein String, der der Liste der Klammern einen Namen zuweist.
- **id**="id_string" Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<brackets name="Bracket" id="CodeColor_JavaBracket"><![CDATA[{ [ ( ) ]}]>
</brackets>
```

<charStart>

Beschreibung

Enthält einen Textstring, der als Trennzeichen für den Beginn eines Zeichens dient. Sie müssen die Tags `charStart` und `charEnd` paarweise angeben. Es sind mehrere `charStart`-`charEnd`-Paare zulässig.

Attribute

Keine

Beispiel

```
<charStart><![CDATA[']]></charStart>
```

<charEnd>

Beschreibung

Enthält einen Textstring, der als Trennzeichen für das Ende eines Zeichens dient. Sie müssen die Tags `charStart` und `charEnd` paarweise angeben. Es sind mehrere `charStart`-`charEnd`-Paare zulässig.

Attribute

Keine

Beispiel

```
<charEnd><![CDATA['']]></charEnd>
```

<charEsc>**Beschreibung**

Enthält einen Textstring, der als Codewechselzeichen (Escape-Zeichen) dient. Es sind mehrere `charEsc`-Tags zulässig.

Attribute

Keine

Beispiel

```
<charEsc><![CDATA[\]]></charEsc>
```

<commentStart>**Beschreibung**

Ein Textstring, der den Beginn eines Kommentars kennzeichnet. Sie müssen die Tags `commentStart` und `commentEnd` paarweise angeben. Es sind mehrere `commentStart.../commentEnd`-Paare zulässig.

Attribute

Keine

Beispiel

```
<commentStart><![CDATA[<!--]]></commentStart>
```

<commentEnd>**Beschreibung**

Ein Textstring, der das Ende eines Kommentars kennzeichnet. Sie müssen die Tags `commentStart` und `commentEnd` paarweise angeben. Es sind mehrere `commentStart.../commentEnd`-Paare zulässig.

Attribute

Keine

Beispiel

```
<commentEnd><![CDATA[- -%>]]></commentEnd>
```

<cssImport/>**Beschreibung**

Ein leeres Tag, das die Codefarbenregel für die `@import`-Funktion des `style`-Elements in einem CSS angibt.

Attribute

`name`, `id`

- `name="cssImport_name"` Ein String, der der `@import`-Funktion des CSS einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet. <

Beispiel

```
<cssImport name="@import" id="CodeColor_CSSImport" />
```

<cssMedia/>**Beschreibung**

Ein leeres Tag, das die Codefarbenregel für die `@media`-Funktion des `style`-Elements in einem CSS angibt.

Attribute

name, id

- `name="cssMedia_name"` Ein String, der der @media-Funktion des CSS einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<cssMedia name="@media" id="CodeColor_CSSMedia" />
```

<cssProperty/>

Beschreibung

Ein leeres Tag, das die CSS-Regeln angibt und Codefarbenattribute enthält.

Attribute

name, id

- `name="cssProperty_name"` Ein String, der der CSS-Eigenschaft einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Voreinstellung für Codefarben

CSS-Eigenschaft

Beispiel

```
<cssProperty name="Property" id="CodeColor_CSSProperty" />
```

<cssSelector/>

Beschreibung

Ein leeres Tag, das die CSS-Regeln angibt und Codefarbenattribute enthält.

Attribute

name, id

- `name="cssSelector_name"` Ein String, der dem CSS-Selektor einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<cssSelector name="Selector" id="CodeColor_CSSSelector" />
```

<cssValue/>

Beschreibung

Ein leeres Tag, das die CSS-Regeln angibt und Codefarbenattribute enthält.

Attribute

name, id

- `name="cssValue_name"` Ein String, der dem CSS-Wert einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<cssValue name="Value" id="CodeColor_CSSValue" />
```

<defaultAttribute>**Beschreibung**

Optional. Dieses Tag gilt nur für tagbasierte Syntax (d. h. `ignoreTags="No"`). Wenn dieses Tag vorhanden ist, erhalten alle Tagattribute die Farbe des Stils, der diesem Tag zugewiesen wurde. Ist dieses Tag nicht vorhanden, erhalten alle Attribute die gleiche Farbe wie das Tag.

Attribute

name

- Ein String, der dem Standardattribut einen Namen zuweist

Beispiel

```
<defaultAttribute name="Attribute"/>
```

<defaultTag>**Beschreibung**

Dieses Tag dient zur Angabe der Standardfarbe und des Standardstils für Tags eines Schemas.

Attribute

name, id

- `name="display_name"` Ein String, der in Dreamweaver im Codefarben-Editor angezeigt wird.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<defaultTag name="Other Tags" id="CodeColor_HTMLTag" />
```

<defaultText/>**Beschreibung**

Optional. Wenn dieses Tag vorhanden ist, erhält Text ohne spezielle Tagdefinition die Farbe des Stils, der diesem Tag zugewiesen wurde. Ist dieses Tag nicht vorhanden, wird der Text schwarz angezeigt.

Attribute

name, id

- `name="cssSelector_name"` Ein String, der dem CSS-Selektor einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<defaultText name="Text" id="CodeColor_TextText" />
```

<endOfLineComment>**Beschreibung**

Ein Textstring, der den Beginn eines Kommentars kennzeichnet, der bis zum Ende der aktuellen Zeile reicht. Es sind mehrere `endOfLineComment-/endOfLineComment`-Tags zulässig.

Attribute

Keine

Beispiel

```
<endOfLineComment><![CDATA[//]]></endOfLineComment>
```

<entity/>**Beschreibung**

Dieses leere Tag gibt an, dass HTML-Sonderzeichen erkannt werden sollen, die Farbattribute enthalten.

Attribute

name, id

- `name="entity_name"` Ein String, der der Entität einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<entity name="Special Characters" id="CodeColor_HTMLEntity" />
```

<functionKeyword>**Beschreibung**

Kennzeichnet Schlüsselwörter, die eine Funktion definieren. Dreamweaver verwendet diese Schlüsselwörter zur Code-Navigation. Es sind mehrere `functionKeyword`-Tags zulässig.

Attribute

name, id

- `name="functionKeyword_name"` Ein String, der dem `functionKeyword`-Block einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<functionKeyword name="Function Keyword" id="CodeColor_JavascriptFunction">function</functionKeyword>
```

<idChar1>**Beschreibung**

Eine Liste von Zeichen, die Dreamweaver jeweils als erstes Zeichen eines Bezeichners erkennt.

Attribute

name, id

- `name="idChar1_name"` Ein String, der der Liste der Bezeichner einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<idChar1>_ $abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</idChar1>
```

<idCharRest>**Beschreibung**

Eine Liste von Zeichen, die als restliche Zeichen in einem Bezeichner erkannt werden. Ist `idChar1` nicht angegeben, werden alle Zeichen des Bezeichners anhand dieser Liste überprüft.

Attribute

name, id

- `name="idCharRest_name"` Ein String, der dem `functionKeyword`-Block einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<idCharRest name="Identifizier" id="CodeColor_JavascriptIdentifizier">
  _$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</idCharRest>
```

<ignoreCase>**Beschreibung**

Gibt an, ob die Groß- und Kleinschreibung beim Vergleichen von Tokens und Schlüsselwörtern ignoriert werden soll. Zulässige Werte sind `yes` und `no`. Die Standardeinstellung ist `yes`.

Attribute

Keine

Beispiel

```
<ignoreCase>yes</ignoreCase>
```

<ignoreMMTParams>**Beschreibung**

Gibt an, ob die Tags `MMTInstance:Param`, `<!-- InstanceParam` und `<!-- #InstanceParam` farblich besonders gekennzeichnet werden sollen. Zulässige Werte sind `yes` und `no`. Die Standardeinstellung ist `yes`. Dies dient zur korrekten Farbzweisung in Seiten, in denen Vorlagen verwendet werden.

Attribute

Keine

Beispiel

```
<ignoreMMTParams>no</ignoreMMTParams>
```

<ignoreTags>**Beschreibung**

Gibt an, ob Markup-Tags ignoriert werden sollen. Zulässige Werte sind `yes` und `no`. Die Standardeinstellung ist `yes`. Legen Sie `no` fest, wenn die Syntax für eine Markup-Sprache verwendet wird, die durch `<` und `>` begrenzt wird. Legen Sie `yes` fest, wenn die Syntax für eine Programmiersprache verwendet wird.

Attribute

Keine

Beispiel

```
<ignoreTags>no</ignoreTags>
```

<isLocked>**Beschreibung**

Gibt an, ob Text, der diesem Schema entspricht, für die Bearbeitung in der Codeansicht gesperrt wird. Zulässige Werte sind `yes` und `no`. Die Standardeinstellung ist `no`.

Attribute

Keine

Beispiel

```
<isLocked>yes</isLocked>
```

<keyword>

Beschreibung

Ein Textstring, der ein Schlüsselwort kennzeichnet. Es sind mehrere `keyword`-Tags zulässig. Ein Schlüsselwort kann mit einem beliebigen Zeichen beginnen, als nachfolgende Zeichen sind jedoch nur `a-z`, `A-Z`, `0-9`, `_`, `$` und `@` zulässig.

Die Codefarbe wird durch die enthaltenen `keyword`-Tags angegeben.

Attribute

Keine

Beispiel

```
<keyword>.getdate</keyword>
```

<keywords>

Beschreibung

Eine Liste von Schlüsselwörtern für Typen, die im Kategorieattribut angegeben sind. Es sind mehrere `keywords`-Tags zulässig.

Attribute

`name`, `id`

- `name="keywords_name"` Ein String, der der Liste der Schlüsselwörter einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Inhalt

```
<keyword></keyword>
```

Beispiel

```
<keywords name="Reserved Keywords" id="CodeColor_JavascriptReserved">  
  <keyword>break</keyword>  
  <keyword>case</keyword>  
</keywords>
```

<numbers/>

Beschreibung

Dieses leere Tag gibt Nummern an, die erkannt werden sollen, und enthält Farbattribute.

Attribute

`name`, `id`

- `name="number_name"` Ein String, der dem `numbers`-Tag einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<numbers name="Number" id="CodeColor_CFScriptNumber" />
```

<operators>

Beschreibung

Eine Liste von Zeichen, die als Operatoren erkannt werden sollen.

Attribute

name, id

- `name="operator_name"` Ein String, der der Liste der Operatoren einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.

Beispiel

```
<operators name="Operator" id="CodeColor_JavaOperator"><![CDATA[+-*/%<?!?:=|^~]]></operators>
```

<regexp>

Beschreibung

Gibt eine Liste mit `searchPattern`-Tags an.

Attribute

name, id, delimiter, escape

- `name="stringStart_name"` Ein String, der der Liste der Suchmuster einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.
- `delimiter` Das Zeichen oder der String, das bzw. der den Anfang oder das Ende eines regulären Ausdrucks kennzeichnet.
- `escape` Das Zeichen oder der String, das bzw. der eine besondere Zeichenverarbeitung ankündigt, ein so genanntes Codewechselzeichen (Escape-Zeichen).

Inhalt

```
<searchPattern></searchPattern>
```

Beispiel

```
<regexp name="RegExp" id="CodeColor_JavascriptRegExp" delimiter="/" escape="\\">
  <searchPattern><![CDATA[(\s*/\e*\[/)]></searchPattern>
  <searchPattern><![CDATA[=\s*/\e*\[/)]></searchPattern>
</regexp>
```

<sampleText>

Beschreibung

Repräsentativer Text, der im Vorschauenfenster des Dialogfelds **Farbschema bearbeiten** angezeigt wird. Weitere Informationen über das Dialogfeld „Farbschema bearbeiten“ finden Sie unter „Schemen bearbeiten“ auf Seite 60.

Attribute

doctypes

- `doctypes="doc_type1, doc_type2, ..."` Die Dokumenttypen, für die dieser Beispielttext angezeigt wird.

Beispiel

```
<sampleText doctypes="JavaScript"><![CDATA[/* JavaScript */
function displayWords(arrayWords) {
    for (i=0; i < arrayWords.length(); i++) {
        // Inline-Kommentar
        alert("Word " + i + " is " + arrayWords[i]);
    }
}

var tokens = new Array("Hello", "world");
displayWords(tokens);
]]></sampleText>
```

<searchPattern>**Beschreibung**

Ein String, der ein reguläres Suchmuster unter Verwendung von unterstützten Platzhalterzeichen definiert. Es sind mehrere `searchPattern`-Tags zulässig.

Attribute

Keine

Container

Das `regexp`-Tag

Beispiel

```
<searchPattern><![CDATA[(\s*/\e*\[/)]></searchPattern>
```

<stringStart>**Beschreibung**

Enthält einen Textstring, der als Trennzeichen für den Beginn eines Strings dient. Sie müssen die Tags `stringStart` und `stringEnd` paarweise angeben. Es sind mehrere `stringStart`-`stringEnd`-Paare zulässig.

Attribute

`name`, `id`, `wrap`

- `name="stringStart_name"` Ein String, der dem `stringStart`-Block einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichner-String, der diesem Syntaxelement Farbe und Stil zuordnet.
- `wrap="true"` oder `"false"`. Gibt an, ob bei Codefarben Textstrings, die auf die nächste Zeile umbrechen, erkannt werden. Die Standardeinstellung ist `true`.

Beispiel

```
<stringStart name="Attribute Value" id="CodeColor_HTMLString"><![CDATA["]]></stringStart>
```

<stringEnd>**Beschreibung**

Enthält einen Textstring, der als Trennzeichen für das Ende eines Codestrings dient. Sie müssen die Tags `stringStart` und `stringEnd` paarweise angeben. Es sind mehrere `stringStart`-`stringEnd`-Paare zulässig.

Attribute

Keine

Beispiel

```
<stringEnd><![CDATA["]]></stringEnd>
```

<stringEsc>**Beschreibung**

Enthält einen Textstring, der als Trennzeichen für ein Codewechselzeichen (Escape-Zeichen) dient. Es sind mehrere `stringEsc`-Tags zulässig.

Attribute

Keine

Beispiel

```
<stringEsc><![CDATA[\]]></stringEsc>
```


Beispielcode

Hier sehen Sie die Darstellung des `customText`-Werts am Beispiel von Trennzeichen für PHP-Skriptblöcke, die rot angezeigt werden:

```
<?php
    if ($loginMsg <> "")
        echo $loginMsg;
?>
```

Beispiel

```
<blockStart name="Block Delimiter" id="CodeColor_JavaBlock" doctype="JSP"
    scheme="customText"><![CDATA[<%]]></blockStart>
```

outerTag

Der `outerTag`-Wert gibt an, dass die Tags `blockStart` und `blockEnd` vollständige Tags sind. Er weist Dreamweaver an, die Tags mit der gleichen Farbe wie die Tags des Umgebungsschemas zu kennzeichnen.

Ein Beispiel für diesen Wert ist das JavaScript-Schema, in dem `<script>` und `</script>` den Tags `blockStart` und `blockEnd` entsprechen. Dieses Schema ordnet Blöcke in JavaScript-Code zu, der jedoch keine Tags erkennt. Daher müssen die Trennzeichen durch das Umgebungsschema farblich gekennzeichnet werden.

Beispielcode

```
<script language="JavaScript">
    // Kommentar
    if (true)
        window.alert("Hello, World");
</script>
```

Beispiel

```
<blockStart doctype="PHP_MySQL" scheme="outerTag">
    <![CDATA[<script\s+language="php">]]></blockStart>
```

innerTag

Dieser Wert ist mit dem Wert `outerTag` identisch, außer dass die Tag-Farbe aus dem Schema innerhalb der Trennzeichen übernommen wird. Dies trifft derzeit auch für `html`-Tags zu.

nameTag

Dieser Wert gibt an, dass `blockStart` das Anfangs-Tag und `blockEnd` das End-Tag ist und diese Trennzeichen farblich an die Tag-Einstellungen des Schemas angepasst werden.

Diese Tags zur Schema-Anzeige können auch in andere Tags eingebettet werden, z. B. in das Tag `cfoutput`.

Beispielcode

```
<input type="text" name="zip"
    <cfif newRecord IS "no">
    <cfoutput query="employee"> Value="#zip#" </cfoutput>
    </cfif>
>
```

Beispiel

```
<blockStart doctype="ColdFusion,CFC" scheme="nameTag">
    <![CDATA[<cfoutput\n]]></blockStart>
```

nameTagScript

Dieser Wert ist mit dem Schema `nameTag` identisch. Der Inhalt besteht jedoch im Unterschied zu `name=value`-Attributpaaren aus Skripts, wie beispielsweise Zuordnungsanweisungen oder Ausdrücken.

Dieser Schematyp zeigt einen eindeutigen Tag-Typ an, der innerhalb des Tags Skripts enthält, wie z. B. die ColdFusion-Tags `cfset`, `cfif` und `cfifelse`, und kann in andere Tags eingebettet werden.

Beispielcode

Siehe Beispieltext für „nameTag“ auf Seite 56.

Beispiel

```
<blockStart doctypes="ColdFusion,CFC" scheme="nameTagScript"><![CDATA[<cfset\n]]></blockStart>
```

Schemaverarbeitung

Dreamweaver verfügt über drei grundlegende Codefarben-Modi: CSS-Modus, Skript-Modus und Tags-Modus.

In jedem Modus weist Dreamweaver bestimmten Feldern Codefarben zu. Nachfolgend finden Sie eine Übersicht darüber, welche Felder in welchem Modus Codefarben erhalten.

Feld	CSS	Tags	Skript
defaultText		X	X
defaultTag		X	
defaultAttribute		X	
comment	X	X	X
string	X	X	X
cssProperty	X		
cssSelector	X		
cssValue	X		
character		X	X
function keyword			X
identifier			X
number		X	X
operator			X
brackets		X	X
keywords		X	X

Um die Schemadefinition flexibler zu gestalten, ermöglicht Dreamweaver die Angabe von bestimmten Platzhalter- und Codewechselzeichen (Escape-Zeichen).

Platzhalterzeichen

Die folgende Liste enthält die von Dreamweaver unterstützten Platzhalterzeichen sowie die spezifischen Strings und eine Beschreibung der Anwendung.

Platzhalter	Escape-Zeichen	Beschreibung
Platzhalter	*	Alle Zeichen in der Regel werden übersprungen, bis das Zeichen nach dem Platzhalter gefunden wurde. Beispiel: Mit <code><MMTInstance:Editable name="*"></code> werden alle Tags dieses Typs mit angegebenem name-Attribut gefunden.
Platzhalter mit Codewechselzeichen (Escape-Zeichen)	\e*x	x steht für das Codewechselzeichen (Escape-Zeichen). Die Verwendung erfolgt wie beim Platzhalter, jedoch kann zusätzlich ein Codewechselzeichen (Escape-Zeichen) bestimmt werden. Das auf ein Escape-Zeichen folgende Zeichen wird ignoriert. Dadurch kann das Zeichen, das auf den Platzhalter folgt, im String angezeigt werden, ohne dass die Platzhalterverarbeitung beendet wird. Beispiel: <code>/\e**/</code> wird zur Kennzeichnung eines regulären JavaScript-Ausdrucks verwendet, der mit einem Schrägstrich (/) beginnt und endet sowie Schrägstriche enthalten kann, denen ein umgekehrter Schrägstrich (\) vorangestellt ist. Da der umgekehrte Schrägstrich das Escape-Zeichen für die Codefarbe ist, müssen Sie ihm einen weiteren umgekehrten Schrägstrich voranstellen, wenn Sie ihn im Codefarben-XML-Code angeben.
Optionaler Leerraum	\s*	Entspricht einer beliebigen Anzahl von Zeichen für Leerräume oder Zeilenumbrüche, einschließlich der Anzahl 0. Beispiel: Mit <code><!--\s*#include</code> stimmen ASP-include-Anweisungen überein, unabhängig davon, ob vor dem <code>#include</code>-Token ein Leerraum steht, da beide Fälle gültig sind. Die Leerraum-Platzhalter gelten für alle Kombinationen aus Zeichen für Leerräume und Zeilenumbrüche.
Erforderlicher Leerraum	\s+	Entspricht einem oder mehreren Zeichen für Leerräume oder Zeilenumbrüche. Beispiel: Mit <code><!--#include\s+virtual</code> stimmen ASP-include-Anweisungen überein, unabhängig davon, in welcher Kombination Leerräume zwischen <code>#include</code> und <code>virtual</code> auftreten. Es muss ein Leerraum zwischen diesen Tokens angegeben werden, dieser kann jedoch aus einer beliebigen Kombination gültiger Leerraum-Zeichen bestehen. Die Leerraum-Platzhalter gelten für alle Kombinationen aus Zeichen für Leerräume und Zeilenumbrüche.

Escape-Zeichen

Die folgende Liste enthält die von Dreamweaver unterstützten Codewechselzeichen (Escape-Zeichen) sowie die spezifischen Strings und eine Beschreibung der Anwendung.

Escape-Zeichen	Escape-Zeichen	Beschreibung
Umgekehrter Schrägstrich	\\	Der umgekehrte Schrägstrich (\) ist das Escape-Zeichen für Codefarben. Daher muss er bei Verwendung in einer Codefarbenregel gekennzeichnet werden.
Leerraum	\s	Dieses Escape-Zeichen entspricht den nicht sichtbaren Zeichen, außer die als Zeichen für Zeilenumbrüche aufgelisteten Escape-Zeichen wie Leerzeichen und Tabulatorzeichen. Die Platzhalter für optionale und erforderliche Leerräume stimmen mit den Zeichen für Leerräume und für Zeilenumbrüche überein.
Zeilenumbruch	\n	Dieses Escape-Zeichen entspricht den Zeichen für Zeilenumbruch (auch als Zeilenvorschub bezeichnet) und Wagenrücklauf.

Maximale Stringlänge

Die maximale Länge für Datenstrings beträgt 100 Zeichen. Das folgende `blockEnd`-Tag enthält zum Beispiel ein Platzhalterzeichen.

```
<blockEnd><![CDATA[<!--\s*#BeginEditable\s*\\"*\s*-->]]></blockEnd>
```

Wenn man davon ausgeht, dass die optionalen Platzhalter für Leerraum (`\s*`) einstellige Zeichen sind, die Dreamweaver automatisch erzeugt, ist der Datenstring 26 Zeichen lang, einschließlich des Platzhalterstrings (`\s*`) für den Namen.

```
<!-- #BeginEditable "\"*" -->
```

Der bearbeitbare Bereichsname kann also bis zu 74 Zeichen lang sein, dies ist die maximale Länge von 100 Zeichen minus 26.

Die Rangfolge eines Schemas

Dreamweaver verwendet zur farbigen Darstellung der Textsyntax in der Codeansicht den folgenden Algorithmus:

1 Dreamweaver bestimmt das anfängliche Syntaxschema anhand des Dokumenttyps der aktuellen Datei. Der Dateidokumenttyp wird mit dem Attribut `scheme.documentType` verglichen. Wird keine Übereinstimmung gefunden, wird das Schema `scheme.documentType = "Text"` verwendet.

2 Schemen können verschachtelt werden, wenn sie in `blockStart`-`blockEnd`-Paaren angegeben sind. Alle verschachtelbaren Schemen, deren `blockStart.doctype`-Attribut die aktuelle Dateierweiterung enthält, sind für die aktuelle Datei zulässig. Alle anderen sind deaktiviert.

Hinweis: Alle `blockStart`/`blockEnd`-Kombinationen müssen eindeutig sein.

Ein Schema kann nur dann mit einem anderen Schema verschachtelt werden, wenn seine Priorität (`scheme.priority`) größer als die oder gleich der Priorität des äußeren Schemas ist. Ist die Priorität gleich, kann das Schema nur im `body`-Status des äußeren Schemas verschachtelt werden. Beispiel: Der Block `<script>...</script>` kann nur dort im Block `<html>...</html>` verschachtelt werden, wo Tags zulässig sind, also nicht innerhalb eines Tags, Attributs, Strings, Kommentars usw.

Schemen mit einer höheren Priorität als das äußere Schema können fast überall im äußeren Schema verschachtelt werden. Beispiel: Zusätzlich zum `body`-Status des Blocks `<html>...</html>` kann der Block `<%...%>` innerhalb eines Tags, Attributs, Strings, Kommentars usw. verschachtelt werden.

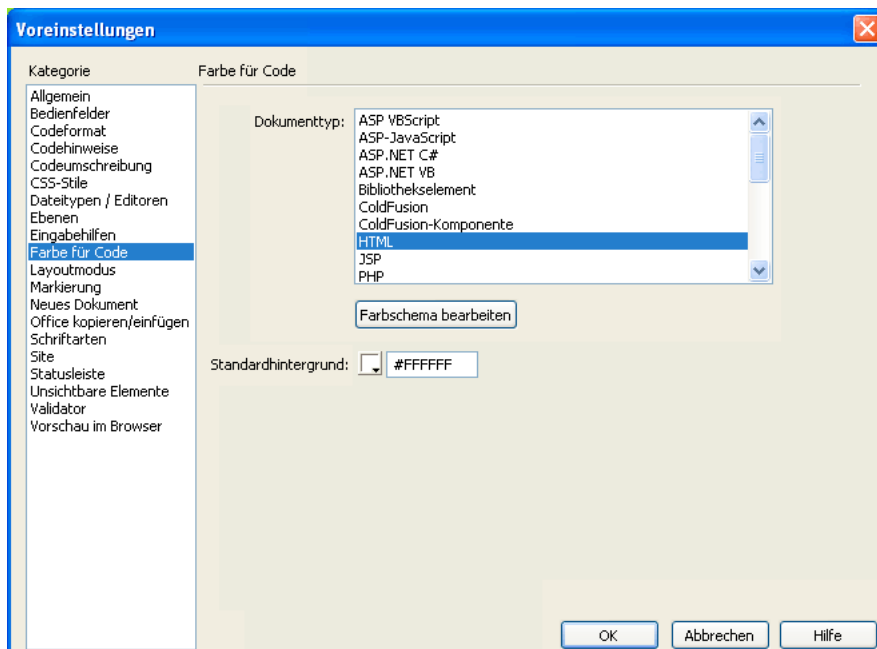
Es sind maximal 4 Verschachtelungsebenen zulässig.

3 Beim Vergleichen von `blockStart`-Strings verwendet Dreamweaver immer die längste Übereinstimmung.

4 Nachdem der `blockEnd`-String für das aktuelle Schema erreicht wurde, wird die Syntax wieder auf die Farbe zurückgesetzt, die sie vor dem Auftreten des `blockStart`-Strings hatte. Wenn der Block `<%...%>` beispielsweise in einem HTML-String aufgetreten ist, wird anschließend wieder die HTML-Stringfarbe verwendet.

Schemen bearbeiten

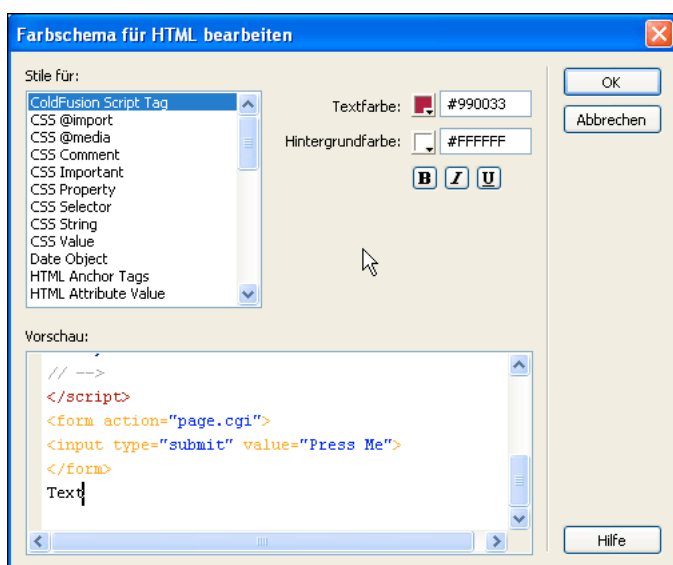
Sie können die Stile für ein Codefarbschema entweder durch Bearbeiten der Codefarbendatei oder durch Auswählen der Kategorie **Farbe für Code** im Dialogfeld **Voreinstellungen** von Dreamweaver anpassen, wie in der folgenden Abbildung dargestellt:



Bei Feldern, die mehrmals angegeben werden können (z. B. `stringStart`), müssen Sie die Farb- und Stileinstellungen nur für das erste Tag eingeben. Wenn Sie die Farb- und Stileinstellungen auf mehrere Tags verteilen und die Farben oder Stile später im Dialogfeld **Voreinstellungen** bearbeiten, gehen Daten verloren.

Hinweis: Sie sollten eine Sicherungskopie aller XML-Dateien erstellen, bevor Sie Änderungen vornehmen. Überprüfen Sie alle manuellen Änderungen, bevor Sie die Farb- und Stileinstellungen im Dialogfeld **Voreinstellungen** bearbeiten. Wenn Sie eine ungültige XML-Datei unter Verwendung des Dialogfelds **Voreinstellungen** bearbeiten, gehen Daten verloren.

Um die Stile eines Schemas mithilfe der Kategorie **Farben für Code** im Dialogfeld **Voreinstellungen** anzupassen, doppelklicken Sie auf einen Dokumenttyp. Alternativ klicken Sie auf die Schaltfläche **Farbschema bearbeiten**, um das Dialogfeld **Farbschema für HTML anpassen** zu öffnen.



Wenn Sie den Stil für ein bestimmtes Element bearbeiten möchten, wählen Sie es in der Liste **Stile für** aus. Zu den im Fenster **Stile für** aufgelisteten Elementen gehören auch die Felder für das bearbeitete Schema sowie die Schemen, die als Blöcke in diesem Schema vorkommen. Wenn Sie beispielsweise das HTML-Schema bearbeiten, werden auch die Felder für CSS- und JavaScript-Blöcke aufgeführt.

Die für ein Schema aufgelisteten Felder entsprechen den in der XML-Datei definierten Feldern. Jedem Feld im Fenster **Stile für** wird der Wert für das Attribut `schema.name` vorangestellt. Felder ohne Namen werden nicht aufgelistet.

Zum Stil oder Format eines Elements gehören neben den Codefarben auch Fett- und Kursivformatierung, Unterstreichung und Hintergrundfarbe. Nachdem Sie ein Element im Fenster **Stile für** ausgewählt haben, können Sie die Stileigenschaften ändern.

Im Vorschaubereich wird angezeigt, wie ein Beispieltext mit den aktuellen Einstellungen aussehen würde. Der Beispieltext wird aus der `sampleText`-Einstellung des Schemas entnommen.

Wählen Sie im Vorschaubereich ein Element aus, um die Auswahl in der Liste **Stile für** zu ändern.

Wenn Sie die Einstellung für ein Element des Schemas ändern, speichert Dreamweaver den Wert in der Codefarbendatei und überschreibt die ursprüngliche Einstellung. Sobald Sie auf **OK** klicken, lädt Dreamweaver alle Änderungen an den Codefarben automatisch neu.

Beispiele für Codefarben

Die folgenden Beispiele für Codefarben zeigen die Code-Farbschemen für ein CSS-Dokument und ein JavaScript-Dokument. Die Liste der Schlüsselwörter für das JavaScript-Beispiel wurde gekürzt, um das Beispiel übersichtlich zu halten.

CSS-Codefarben

```
<scheme name="CSS" id="CSS" doctypes="CSS" priority="50">
  <ignoreCase>Yes</ignoreCase>
  <ignoreTags>Yes</ignoreTags>
  <blockStart doctypes="ASP-JS,ASP-VB,ASP.NET_CSharp,ASP.NET_VB,ColdFusion,
    CFC,HTML,JSP,LibraryItem,DWTemplate,PHP_MySQL" scheme="outerTag">
    <![CDATA[<style>]]></blockStart>
  <blockEnd><![CDATA[</style>]]></blockEnd>
  <blockStart doctypes="ASP-JS,ASP-VB,ASP.NET_CSharp,ASP.NET_VB,ColdFusion,
    CFC,HTML,JSP,LibraryItem,DWTemplate,PHP_MySQL" scheme="outerTag">
    <![CDATA[<style\s+*>]]></blockStart>
  <blockEnd><![CDATA[</style>]]></blockEnd>
  <commentStart name="Comment" id="CodeColor_CSSComment"><![CDATA[ /*]]></commentStart>
  <commentEnd><![CDATA[ */]]></commentEnd>
  <endOfLineComment><![CDATA[!--]]></endOfLineComment>
  <endOfLineComment><![CDATA[-->]]></endOfLineComment>
  <stringStart name="String" id="CodeColor_CSSString"><![CDATA["]]></stringStart>
  <stringEnd><![CDATA["]]></stringEnd>
  <stringStart><![CDATA[']]></stringStart>
  <stringEnd><![CDATA[']]></stringEnd>
  <stringEsc><![CDATA[\\]]></stringEsc>
  <cssSelector name="Selector" id="CodeColor_CSSSelector" />
  <cssProperty name="Property" id="CodeColor_CSSProperty" />
  <cssValue name="Value" id="CodeColor_CSSValue" />
  <sampleText doctypes="CSS"><![CDATA[/* Comment */
H2, .head2{
    font-family : 'Sans-Serif';
    font-weight : bold;
    color : #339999;
  }]]>
</sampleText>
</scheme>
```

CSS-Beispieltext

Der folgende Beispieltext für das CSS-Schema veranschaulicht das CSS-Codefarbschema.

```

/* Kommentar */
H2, .head2{
    font-family : 'Sans-Serif';
    font-weight : bold;
    color : #339999;
}

```

Die folgenden Zeilen aus der Datei Colors.xml bestimmen die Farb- und Stilwerte, die im Beispieltext angezeigt werden und durch das Codefarbschema zugewiesen wurden:

```

<syntaxColor id="CodeColor_CSSSelector" text="#FF00FF" />
<syntaxColor id="CodeColor_CSSProperty" text="#000099" />
<syntaxColor id="CodeColor_CSSValue" text="#0000FF" />

```

JavaScript-Codefarben

```

<scheme name="JavaScript" id="JavaScript" doctypes="JavaScript" priority="50">
    <ignoreCase>No</ignoreCase>
    <ignoreTags>Yes</ignoreTags>
    <blockStart doctypes="ASP-JS,ASP-VB,ASP.NET_CSharp,ASP.NET_VB,ColdFusion,
        CFC,HTML,JSP,LibraryItem,DWTemplate,PHP_MySQL" scheme="outerTag">
        <![CDATA[<script>]]></blockStart>
    <blockEnd><![CDATA[</script>]]></blockEnd>
    <blockStart doctypes="ASP-JS,ASP-VB,ASP.NET_CSharp,ASP.NET_VB,ColdFusion,
        CFC,HTML,JSP,LibraryItem,DWTemplate,PHP_MySQL" scheme="outerTag">
        <![CDATA[<script\s+\*>]]></blockStart>
    <blockEnd><![CDATA[</script>]]></blockEnd>
    <commentStart name="Comment" id="CodeColor_JavascriptComment">
        <![CDATA[/\*]]></commentStart>
    <commentEnd><![CDATA[/\*]]></commentEnd>
    <endOfLineComment><![CDATA[/\*]]></endOfLineComment>
    <endOfLineComment><![CDATA[/\*]]></endOfLineComment>
    <endOfLineComment><![CDATA[/\*]]></endOfLineComment>
    <stringStart name="String" id="CodeColor_JavascriptString">
        <![CDATA["']]></stringStart>
    <stringEnd><![CDATA["']]></stringEnd>
    <stringStart><![CDATA['']]></stringStart>
    <stringEnd><![CDATA['']]></stringEnd>
    <stringEsc><![CDATA[\\]]></stringEsc>
    <brackets name="Bracket" id="CodeColor_JavascriptBracket">
        <![CDATA[{[ ( ) ]}]></brackets>
    <operators name="Operator" id="CodeColor_JavascriptOperator">
        <![CDATA[+-*/%<>!?:=&|^]]></operators>
    <numbers name="Number" id="CodeColor_JavascriptNumber" />
    <regex name="RegExp" id="CodeColor_JavascriptRegExp" delimiter="/" escape="\\">
        <searchPattern><![CDATA[(\s*/\e*\[/]]></searchPattern>
        <searchPattern><![CDATA[=\s*/\e*\[/]]></searchPattern>
    </regex>
    <idChar1>_abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</idChar1>
    <idCharRest name="Identifier" id="CodeColor_JavascriptIdentifier">
        _abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</idCharRest>
    <functionKeyword name="Function Keyword" id="CodeColor_JavascriptFunction">
        function</functionKeyword>
    <keywords name="Reserved Keywords" id="CodeColor_JavascriptReserved">
        <keyword>break</keyword>
    . . .
    </keywords>
    <keywords name="Native Keywords" id="CodeColor_JavascriptNative">
        <keyword>abs</keyword>
    . . .

```

```

        </keywords>
        <keywords id="CodeColor_JavascriptNumber">
            <keyword>Infinity</keyword>
            <keyword>NaN</keyword>
        </keywords>
        <keywords name="Client Keywords" id="CodeColor_JavascriptClient">
            <keyword>alert</keyword>
        </keywords>
        <sampleText><![CDATA[/* JavaScript */
function displayWords(arrayWords) {
    for (i=0; i < arrayWords.length(); i++) {
        // Inline-Kommentar
        alert("Word " + i + " is " + arrayWords[i]);
    }
}
var tokens = new Array("Hello", "world");
displayWords(tokens);
]]></sampleText>
</scheme>

```

JavaScript-Beispieltext

Der folgende Beispieltext für das JavaScript-Schema veranschaulicht das JavaScript-Codefarbschema.

```

/* JavaScript */
function displayWords(arrayWords) {
    for (i=0; i < arrayWords.length(); i++) {
        // Inline-Kommentar
        alert("Word " + i + " is " + arrayWords[i]);
    }
}
var tokens = new Array("Hello", "world");
displayWords(tokens);

```

Die folgenden Zeilen aus der Datei **Colors.xml** bestimmen die Farb- und Stilwerte, die im Beispieltext angezeigt werden und durch das Codefarbschema zugewiesen wurden:

```

<syntaxColor id="CodeColor_JavascriptComment" text="#999999" italic="true" />
<syntaxColor id="CodeColor_JavascriptFunction" text="#000000" bold="true" />
<syntaxColor id="CodeColor_JavascriptBracket" text="#000099" bold="true" />
<syntaxColor id="CodeColor_JavascriptNumber" text="#FF0000" />
<syntaxColor id="CodeColor_JavascriptClient" text="#990099" />
<syntaxColor id="CodeColor_JavascriptNative" text="#009999" />

```

Codeüberprüfung

Wenn Sie ein Dokument in der Codeansicht öffnen, überprüft Dreamweaver automatisch, ob im Dokument Tags, Attribute, CSS-Eigenschaften oder CSS-Werte verwendet werden, die in den vom Benutzer ausgewählten Zielbrowsern nicht zur Verfügung stehen. Dreamweaver unterstreicht Fehler mit einer roten Wellenlinie.

Dreamweaver verfügt auch über eine neue Funktion zur Browserkompatibilitätsprüfung. Damit lassen sich Kombinationen an HTML- und CSS-Code ausfindig machen, die Probleme bei der Browserdarstellung auslösen können.

Die Browserprofile werden in Dreamweaver im Ordner **Configuration\BrowserProfiles** gespeichert. Jedes Browserprofil ist als Textdatei definiert, die nach dem Browser benannt ist. Das Browserprofil für Internet Explorer Version 6.0 trägt beispielsweise den Namen **Internet_Explorer_6.0.txt**. Um die Zielbrowser-Prüfung für CSS zu ermöglichen, speichert Dreamweaver CSS-Profilinformationen für Browser in einer XML-Datei, deren Name dem Browserprofil entspricht, jedoch mit der Erweiterung „_CSS.xml“. Das CSS-Profil für Internet Explorer 6.0 heißt etwa **Internet_Explorer_6.0_CSS.xml**. Wenn Dreamweaver Fehler anzeigt, die Sie ignorieren möchten, können Sie Änderungen an der CSS-Profildatei vornehmen.

Die CSS-Profildatei besteht aus drei XML-Tags: `css-support`, `property` und `value`. Im folgenden Abschnitt werden diese Tags näher erläutert.

<css-support>

Beschreibung

Dieses Tag ist der Stammknoten für eine Gruppe von `property`- und `value`-Tags, die von einem bestimmten Browser unterstützt werden.

Attribute

Keine

Inhalt

Die Tags `property` und `value`.

Container

Keiner

Beispiel

```
<css-support>
. . .
</css-support>
```

<property>

Beschreibung

Definiert eine unterstützte CSS-Eigenschaft für das Browserprofil.

Attribute

`name`, `names`, `supportlevel`, `message`

- `name="property_name"` Der Name der Eigenschaft, für die Sie eine Unterstützung festlegen.
- `names="property_name, property_name, ..."` Eine durch Kommas getrennte Liste mit Eigenschaftsnamen, für die Sie eine Unterstützung festlegen.

Das Attribut `names` ist eine Art Kurzschrift. Das folgende `names`-Attribut ist zum Beispiel eine Kurzschriftmethode zur Definition des darauffolgenden `name`-Attributs:

```
<property names="foo,bar">
  <value type="named" name="top"/>
  <value type="named" name="bottom"/>
</property>
<property name="foo">
  <value type="named" name="top"/>
  <value type="named" name="bottom"/>
</property>
<property name="bar">
  <value type="named" name="top"/>
  <value type="named" name="bottom"/>
</property>
```

- `supportlevel="error", "warning", "info" oder "supported"` gibt die Unterstützungsebene für die Eigenschaft an. Wenn nichts angegeben ist, wird `supported` angenommen. Wenn Sie eine andere Unterstützungsebene als `supported` ohne das `message`-Attribut angeben, verwendet Dreamweaver die Standardmeldung „CSS property name `property_name` is not supported“.
- `message="message_string"` Das Attribut `message` definiert eine Meldung, die Dreamweaver anzeigt, wenn eine Eigenschaft in einem Dokument gefunden wird. Diese Meldung beschreibt mögliche Einschränkungen oder Umgehungen des Eigenschaftswerts.

Inhalt

value

Container

css-support

Beispiel

```
<property name="background-color" supportLevel="supported">
```

<value>**Beschreibung**

Definiert eine Liste von Werten, die von der aktuellen Eigenschaft unterstützt werden.

Attribute

type, name, names, supportlevel, message

- type="any", "named", "units", "color", "string" oder "function" gibt den Werttyp an. Wenn Sie "named", "units", or "color" verwenden, müssen über die Attribute name oder names die zuzuordnenden Wert-IDs für dieses Element angegeben werden. Der Wert "units" entspricht einem numerischen Wert, gefolgt von einem im names-Attribut festgelegten Einheitswert.
- name="value_name" ist ein Bezeichner für CSS-Werte. Es sind keine Leerzeichen oder Satzzeichen erlaubt, mit Ausnahme von Bindestrichen (-). Der Name eines der Werte, die für die CSS-Eigenschaft gültig sind, die in dem übergeordneten Eigenschaftsknoten angegeben wurden. Dabei kann es sich um einen bestimmten Wert oder einen Einheits-spezifikator handeln.
- names="Name1, Name2, . . ." gibt eine durch Kommas getrennte Liste mit Wert-IDs an.
- supportlevel="error", "warning", "info" oder "supported" gibt die Unterstützungsebene für diesen Wert im Browser an. Wenn nichts angegeben ist, wird der Wert supported angenommen.
- message="message_string". Das Attribut message definiert eine Meldung, die Dreamweaver anzeigt, wenn der Eigenschaftswert in einem Dokument gefunden wird. Ist das Attribut message nicht angegeben, zeigt Dreamweaver die Meldung „value_name is not supported“ an.

Inhalt

Keiner

Container

property

Beispiel

```
<property name="margin">
  <value type="units" name="ex" supportLevel="warning"
    message="The implementation of ex units is buggy in Safari 1.0."/>
  <value type="units" names="% ,em,px,in,cm,mm,pt,pc"/>
  <value type="named" name="auto"/>
  <value type="named" name="inherit"/>
</property>
```

Die HTML-Standardformatierung ändern

Um die allgemeinen Einstellungen für die Codeformatierung zu ändern, verwenden Sie die Kategorie **Codeformat** des Dialogfelds **Voreinstellungen**. Um das Format bestimmter Tags und Attribute zu ändern, verwenden Sie den Tag-Bibliothek-Editor (**Bearbeiten** > **Tag-Bibliotheken**). Weitere Informationen finden Sie in der Dreamweaver-Hilfe unter *Dreamweaver verwenden*.

Sie können die Tag-Formatierung auch ändern, indem Sie die zugehörige VTM-Datei bearbeiten (in einem Unterordner des Konfigurationsordners der Tag-Bibliothek). Es ist jedoch einfacher, die Formatierung in Dreamweaver zu ändern.

Wenn Sie eine VTM-Datei hinzufügen oder entfernen, müssen Sie dies in der Datei **TagLibraries.vtm** vermerken. Dreamweaver ignoriert alle VTM-Dateien, die nicht in TagLibraries.vtm aufgelistet sind.

Hinweis: Bearbeiten Sie diese Datei in einem Texteditor, nicht in Dreamweaver.

Kapitel 4: Dreamweaver erweitern

In der Regel werden Adobe Dreamweaver CS3-Erweiterungen erstellt, um mit ihnen Vorgänge der folgenden Arten durchzuführen:

- Automatische Änderungen am aktuellen Dokument des Benutzers – z. B. HTML, CFML oder JavaScript einfügen, Text oder Bildeigenschaften ändern, Tabellen sortieren.
- Interaktion mit der Anwendung zum automatischen Öffnen oder Schließen von Fenstern oder Dokumenten, zum Ändern von Tastenkombinationen und mehr.
- Verbindung zu Datenquellen herstellen, wodurch die Dreamweaver-Benutzer dynamische, datengesteuerte Seiten erstellen können.
- Einfügen und Verwalten von Servercodeblöcken im aktuellen Dokument.

Eine Erweiterung können Sie schreiben, um eine häufig verwendete und sich daher wiederholende Aufgabe zu erledigen. Möglicherweise haben Sie auch besondere Anforderungen, die nur durch das Schreiben einer Erweiterung erfüllt werden können. In beiden Fällen bietet Dreamweaver eine umfangreiche Palette an Tools, mit denen Sie die Funktionalität erweitern und anpassen können.

Gehen Sie beim Erstellen einer Dreamweaver-Erweiterung entsprechend den Anweisungsschritten unter „Erweiterungen erstellen“ auf Seite 2 vor.

Mithilfe der folgenden Erweiterungen von Dreamweaver können Sie Erweiterungen erstellen:

- Ein HTML-Parser (auch *Renderer* genannt), durch den Benutzeroberflächen (UIs) für Erweiterungen unter Verwendung von Formularfeldern, absolut positionierten Elementen, Bildern und anderen HTML-Elementen erstellt werden können. Dreamweaver verfügt über einen eigenen HTML-Parser.
- Eine Ordnerstruktur, in der die Dateien organisiert und gespeichert werden, die Dreamweaver-Elemente und -Erweiterungen implementieren und konfigurieren.
- Eine Reihe von APIs (Anwendungsprogrammierschnittstellen), die Zugriff auf die Dreamweaver-Funktionalität über JavaScript bieten.
- Ein JavaScript-Interpreter, der den JavaScript-Code in Erweiterungsdateien ausführt. Dreamweaver verwendet den JavaScript-Interpreter von Netscape in der Version 1.5. Weitere Informationen über die Unterschiede dieser Interpreter-Version gegenüber früheren Versionen finden Sie im Abschnitt „Verarbeitung von JavaScript in Erweiterungen durch Dreamweaver“ auf Seite 71.

Typen von Dreamweaver-Erweiterungen

In der folgenden Liste werden die in diesem Handbuch dokumentierten Typen von Dreamweaver-Erweiterungen beschrieben:

Einfügeleiste-Objekterweiterungen führen Änderungen in der Einfügeleiste durch. Ein Objekt wird in der Regel dazu verwendet, das Einfügen von Code in ein Dokument zu automatisieren. Es kann darüber hinaus ein Formular enthalten, das Benutzereingaben aufnimmt, sowie JavaScript, das die Eingabe verarbeitet. Objektdateien werden im Ordner **Configuration\Objects** gespeichert.

Befehlserweiterungen können fast alle Arten von Aufgaben übernehmen – mit oder ohne Benutzereingabe. Befehlsdateien werden meistens über das Menü **Befehle** aufgerufen, können aber auch von anderen Erweiterungen aufgerufen werden. Befehlsdateien werden im Ordner **Configuration\Commands** gespeichert.

Menübefehlserweiterungen erweitern die Befehls-API um Funktionen im Zusammenhang mit dem Aufruf von Befehlen aus Menüs. Mithilfe der Menübefehls-API können Sie außerdem ein dynamisches Untermenü erstellen.

Symbolleistenerweiterungen dienen dazu, in der Benutzeroberfläche von Dreamweaver den Symbolleisten neue Elemente hinzuzufügen oder neue Symbolleisten zu erstellen. Neue Symbolleisten werden unterhalb der standardmäßigen Symbolleiste angezeigt. Symbolleistendateien werden im Ordner **Configuration\Toolbars** gespeichert.

Berichtserweiterungen dienen zum Hinzufügen benutzerdefinierter Site-Berichte oder Ändern der mit Dreamweaver gelieferten vordefinierten Berichte. Außerdem können Sie die API für das Ergebnisfenster verwenden, um eigenständige Berichte zu erstellen.

Tag-Bibliothekserweiterungen und Tag-Editorerweiterungen arbeiten mit den entsprechenden Tag-Bibliothek-Dateien zusammen. Tag-Bibliothekserweiterungen und Tag-Editorerweiterungen können dazu verwendet werden, Attribute vorhandener Tag-Dialogfelder zu ändern, neue Tag-Dialogfelder zu erstellen und Tags zur Tag-Bibliothek hinzuzufügen. Die Erweiterungsdateien für Tag-Bibliotheken und den Tag-Editor werden im Ordner **Configuration\TagLibraries** gespeichert.

Eigenschaftensinspektor-Erweiterungen werden im Bedienfeld des Eigenschaftensinspektors angezeigt. Die meisten Inspektoren von Dreamweaver sind Bestandteil des Kernproduktcodes und können nicht geändert werden, aber benutzerdefinierte Eigenschaftensinspektor-Dateien können die integrierten Benutzeroberflächen der Dreamweaver-Eigenschaftenspektoren außer Kraft setzen oder neue Eigenschaftenspektoren erstellen, die benutzerdefinierte Tags überprüfen. Inspektoren werden im Ordner **Configuration\Inspectors** gespeichert.

Erweiterungen für schwebende Bedienfelder dienen dazu, der Dreamweaver-Benutzeroberfläche schwebende Bedienfelder hinzuzufügen. Schwebende Bedienfelder können sich auf die Auswahl, das Dokument oder den Arbeitsschritt beziehen. Außerdem können sie nützliche Informationen anzeigen. Dateien für schwebende Bedienfelder werden im Ordner **Configuration\Floaters** gespeichert.

Verhaltenserweiterungen bieten Benutzern die Möglichkeit, ihren Dokumenten JavaScript-Code hinzuzufügen. Der JavaScript-Code führt einen bestimmten Arbeitsschritt als Reaktion auf ein Ereignis aus, wenn das Dokument in einem Browser angezeigt wird. Verhaltenserweiterungen werden im Menü mit dem Pluszeichen (+) des Dreamweaver-Bedienfelds **Verhalten** angezeigt. Die Verhaltensdateien werden im Ordner **Configuration\Behaviors\Actions** gespeichert.

Serververhalten Serververhaltenserweiterungen fügen dem Dokument Blöcke mit serverbasiertem Code (ASP, JSP oder ColdFusion) hinzu. Serverbasierter Code führt Arbeitsschritte auf dem Server durch, wenn das Dokument in einem Browser angezeigt wird. Serververhalten werden im Menü mit dem Pluszeichen (+) des Dreamweaver-Bedienfelds **Serververhalten** angezeigt. Serververhaltensdateien werden im Ordner **Configuration\Server Behaviors** gespeichert.

Datenquelle Datenquellenerweiterungen dienen dazu, eine Verbindung zu dynamischen Daten in einer Datenbank herzustellen. Datenquellenerweiterungen werden im Menü mit dem Pluszeichen (+) des Bedienfelds **Bindungen** angezeigt. Datenquellenerweiterungsdateien werden im Ordner **Configuration\DataSources** gespeichert.

Serverformaterweiterungen dienen dazu, die Formatierung von dynamischen Daten festzulegen.

Komponentenerweiterungen dienen dazu, neue Komponententypen zum Bedienfeld **Komponenten** hinzuzufügen. Dreamweaver verwendet den Begriff „Komponenten“ für einige der häufiger verwendeten und modernen Verkapselungsstrategien, wie z. B. für Webdienste, JavaBeans und ColdFusion-Komponenten (CFCs).

Servermodellerweiterungen ermöglichen die zusätzliche Unterstützung neuer Servermodelle. Dreamweaver unterstützt die gängigsten Servermodelle (ASP, JSP, ColdFusion, PHP und ASP.NET). Servermodellerweiterungen werden somit nur für individuelle Serverlösungen, andere Sprachen oder einen benutzerdefinierten Server benötigt. Servermodelldateien werden im Ordner **Configuration\ServerModels** gespeichert.

Datenübersetzererweiterungen konvertieren Nicht-HTML-Code in HTML-Code, der in der Entwurfsansicht des Dokumentfensters angezeigt wird. Diese Erweiterungen sperren zudem den Nicht-HTML-Code, sodass er nicht von Dreamweaver analysiert werden kann. Übersetzerdateien werden im Ordner **Configuration\Translators** gespeichert.

Andere Möglichkeiten zur Erweiterung von Dreamweaver

Sie können auch die folgenden Elemente von Dreamweaver erweitern, um die Möglichkeiten des Programms zu erweitern oder es an Ihre Anforderungen anzupassen.

Dokumenttypen. definieren, wie Dreamweaver mit unterschiedlichen Servermodellen zusammenarbeitet. Informationen über Dokumenttypen für Servermodelle werden im Ordner **Configuration\DocumentTypes** gespeichert. Weitere Informationen finden Sie unter „Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 15.

Codefragmente sind wieder verwendbare Codeblöcke, die als CSN-Dateien im Dreamweaver-Ordner **Configuration\Snippets** gespeichert werden und die Dreamweaver im Bedienfeld **Codefragmente** zur Verfügung stellt. Sie können weitere Codefragmentdateien erstellen und im Codefragmentordner ablegen, um sie zur Verfügung zu stellen.

Codehinweise sind Menüs, die eine schnellere Eingabe ermöglichen, indem sie eine Liste der Strings anbieten, die das von Ihnen eingegebene Zeichenmuster vervollständigen. Wenn einer der Strings im Menü den ersten von Ihnen eingegebenen Zeichen entspricht, können Sie ihn durch Auswählen einfügen. Codehinweismenüs sind im Ordner **Configuration\CodeHints** in der Datei **codehints.xml** definiert. Sie können dieser Datei neue Codehinweise für neue Tags oder Funktionen hinzufügen.

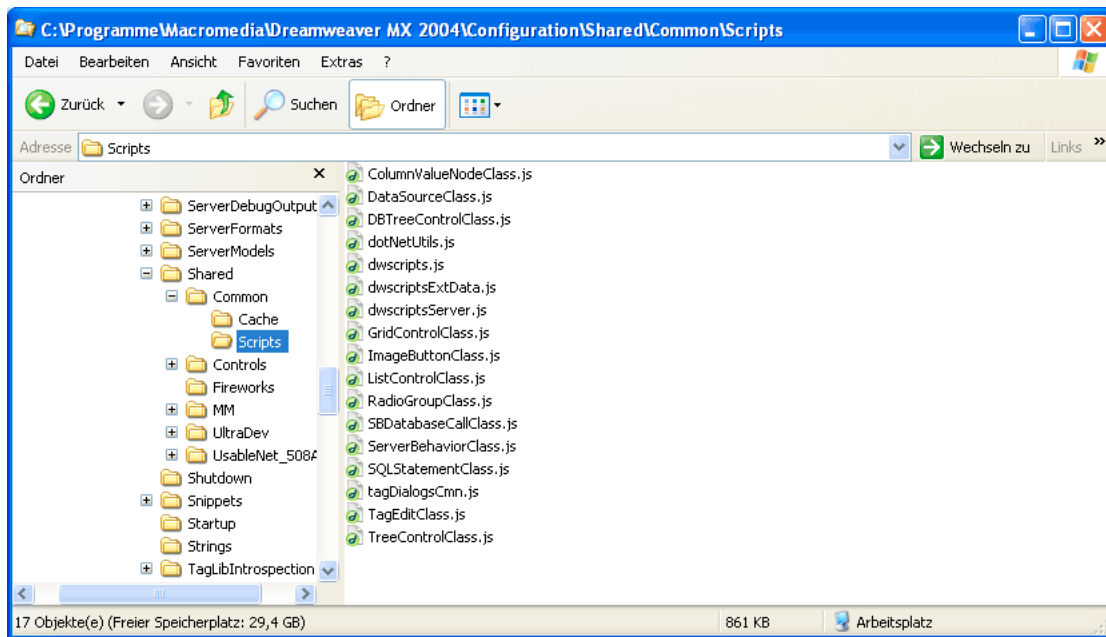
Menüs sind im Ordner **Configuration\Menus** in der Datei **menus.xml** definiert. Sie können neue Dreamweaver-Menüs für die Erweiterungen hinzufügen, indem Sie die Menü-Tags für diese Erweiterungen zur Datei **menus.xml** hinzufügen. Weitere Informationen finden Sie unter „Menüs und Menübefehle“ auf Seite 133.

Konfigurationsordner und Erweiterungen

Die im Konfigurationsordner von Dreamweaver gespeicherten Ordner und Dateien enthalten die mit Dreamweaver gelieferten Erweiterungen. Wenn Sie eine Erweiterung schreiben, müssen Sie die Dateien im richtigen Ordner speichern, sodass Dreamweaver sie erkennt. So sind beispielsweise die Dateien einer von Ihnen erstellten Erweiterung für den Eigenschaftenspektor im Ordner **Configuration\Inspectors** zu speichern. Wenn Sie eine Erweiterung von der Adobe Exchange-Website (www.adobe.com/go/exchange_de), herunterladen und installieren, speichert der Extension Manager die Erweiterungsdateien automatisch in den richtigen Ordnern.

Sie können die Dateien im Konfigurationsordner von Dreamweaver als Beispiel verwenden. Diese Dateien sind in der Regel jedoch komplexer als die meisten Erweiterungen, die Sie auf der Adobe Exchange-Website finden. Weitere Informationen über den Inhalt der einzelnen Unterordner im Konfigurationsordner finden Sie in der Datei **Configuration_ReadMe.htm**.

Der Ordner **Configuration\Shared** bezieht sich nicht auf einen bestimmten Erweiterungstyp. Er ist der zentrale Speicherplatz für Dienstprogrammfunktionen, Klassen und Bilder, die von mehr als einer Erweiterung verwendet werden. Die Dateien im Ordner **Configuration\Shared\Common** sind für zahlreiche Erweiterungen von Nutzen. Diese Dateien sind sowohl als Beispiele für JavaScript-Techniken als auch als Dienstprogramme sehr nützlich. Wenn Sie eine Funktion suchen, die bestimmte Aufgaben übernehmen soll (Erstellen einer gültigen DOM-Referenz (Dokumentobjektmodell) auf ein Objekt; Prüfen, ob die aktuelle Auswahl sich in einem bestimmten Tag befindet, Sonderzeichen in Strings einfügen usw.), dann sollten Sie zunächst hier nachsehen. Wenn Sie gemeinsame Dateien erstellen, sollten Sie im Ordner **Configuration\Shared\Common**, der in der folgenden Abbildung dargestellt ist, zunächst einen eigenen Unterordner zum Speichern dieser Dateien erstellen.



Dateistruktur des Ordners **Configuration\Shared\Common\Scripts**

Weitere Informationen zum Ordner **Shared** finden Sie in „Der Ordner „Shared““ auf Seite 349.

Konfigurationsordner bei mehreren Benutzern

Bei den Mehrbenutzer-Betriebssystemen Windows XP, Windows 2000, Windows NT und Macintosh OS X erstellt Dreamweaver zusätzlich zum Dreamweaver-Konfigurationsordner für jeden Benutzer einen eigenen Konfigurationsordner. Jedes Mal, wenn Dreamweaver oder eine JavaScript-Erweiterung in den Konfigurationsordner schreibt, verwendet Dreamweaver automatisch den Konfigurationsordner des Benutzers. Auf diese Weise kann jeder Dreamweaver-Benutzer eigene Konfigurationseinstellungen festlegen, ohne dabei die Einstellungen der anderen Benutzer zu beeinträchtigen. Weitere Informationen finden Sie unter „Dreamweaver in einer Mehrbenutzerumgebung anpassen“ auf Seite 12 und „API für Dateizugriff und Mehrbenutzerkonfiguration“ im *Dreamweaver API-Referenzhandbuch*.

Skripts beim Starten bzw. Beenden ausführen

Wenn Sie eine Befehlsdatei im Ordner **Configuration\Startup** speichern, wird der Befehl beim Start von Dreamweaver ausgeführt. Startbefehle werden vor der Datei `menus.xml`, vor den Dateien im Ordner **ThirdPartyTags** und vor allen Befehlen, Objekten, Verhalten, Inspektoren, schwebenden Bedienfeldern und Übersetzern geladen. Sie können Startbefehle dazu verwenden, die Datei `menus.xml` oder andere Erweiterungsdateien zu ändern. Sie können auch Warnungen oder Eingabeaufforderungen für den Benutzer anzeigen und die Funktion `dreamweaver.runCommand()` aufrufen. Allerdings können Sie aus dem Ordner **Startup** keinen Befehl aufrufen, für den ein gültiges Dokumentobjektmodell (DOM) erforderlich ist. Weitere Informationen zum Dreamweaver-DOM finden Sie in „Das Dreamweaver-Dokumentobjektmodell“ auf Seite 87.

Wenn Sie analog dazu eine Befehlsdatei im Ordner **Configuration\Shutdown** ablegen, wird diese beim Beenden von Dreamweaver ausgeführt. Mit Beenden-Befehlen können Sie die Funktion `dreamweaver.runCommand()` aufrufen sowie Warnungen oder Eingabeaufforderungen für den Benutzer anzeigen. Der Vorgang des Beendens selbst kann jedoch nicht unterbrochen werden.

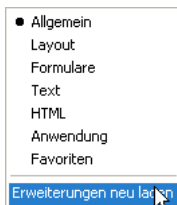
Weitere Informationen zu den Befehlen finden Sie in „Befehle“ auf Seite 123. Weitere Informationen zur Funktion `dreamweaver.runCommand()` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Erweiterungen neu laden

Wenn Sie während der Arbeit mit Dreamweaver eine Erweiterung ändern, können Sie die Erweiterungen neu laden, damit Dreamweaver die Änderung erkennt.

So laden Sie Erweiterungen neu:

1 Klicken Sie bei gedrückter Taste <Strg> (Windows) bzw. Wahl taste (Macintosh) in der Titelleiste der Einfügeleiste auf das Menü **Kategorien**.



2 Wählen Sie die Option **Erweiterungen neu laden**.

Hinweis: Bei einem Mehrbenutzer-Betriebssystem sollten Sie nicht die Hauptkonfigurationsdateien bearbeiten, sondern die Kopien dieser Dateien im Ordner **Configuration** des Benutzers. Weitere Informationen finden Sie unter „Konfigurationsordner und Erweiterungen“ auf Seite 69.

Erweiterungs-APIs

Die Erweiterungs-APIs stellen Ihnen die Funktionen zur Verfügung, die Dreamweaver zur Implementierung der einzelnen Erweiterungstypen aufruft. Sie schreiben die Textteile dieser Funktionen wie für jeden Erweiterungstyp beschrieben und legen die erforderlichen, von Dreamweaver erwarteten Rückgabewerte fest.

Für Entwickler, die direkt in der Programmiersprache C arbeiten möchten, ist eine Erweiterungs-API für C vorhanden, mit der DLLs (dynamische Bibliotheken) erstellt werden können. Die von diesen APIs bereitgestellte Funktionalität bettet Ihre C-DLLs in JavaScript ein, sodass Ihre Erweiterung in Dreamweaver reibungslos funktioniert.

In der Dokumentation über Erweiterungs-APIs wird erläutert, was jede Funktion beim Aufruf durch Dreamweaver bewirkt und was als Rückgabewert erwartet wird.

Informationen über die Dienstprogramm-APIs und die JavaScript-APIs, die Funktionen zur Ausführung bestimmter Arbeitsschritte in den Erweiterungen bereitstellen, finden Sie im *Dreamweaver API-Referenzhandbuch*.

Verarbeitung von JavaScript in Erweiterungen durch Dreamweaver

Dreamweaver überprüft beim Start den Ordner **Configuration\Erweiterungstyp**. Wenn Dreamweaver eine Erweiterungsdatei im Ordner entdeckt, wird das JavaScript wie folgt verarbeitet:

- Kompilieren des gesamten Inhalts zwischen den öffnenden und schließenden `SCRIPT`-Tags
- Ausführen des gesamten Codes zwischen den `SCRIPT`-Tags, der nicht Teil einer Funktionsdeklaration ist.

Hinweis: Diese Prozedur muss in der Startphase erfolgen, da für manche Erweiterungen die Initialisierung globaler Variablen erforderlich ist.

Für alle externen JavaScript-Dateien, die in den Attributen `SRC` der Tags `SCRIPT` angegeben sind, führt Dreamweaver die folgenden Aktionen aus:

- Einlesen der Datei
- Kompilieren des Codes
- Ausführen der Prozeduren

Hinweis: Falls im JavaScript-Code der Erweiterungsdatei der String `</SCRIPT>`, enthalten ist, wird dies vom JavaScript-Interpreter als ein beendendes `script`-Tag interpretiert, und es wird eine Fehlermeldung ausgegeben. Sie können dieses Problem vermeiden, indem Sie den String wie folgt unterteilen und die einzelnen Teile miteinander verketteten:

```
"<' + '/SCRIPT>".
```

Dreamweaver führt den Code in der Ereignisprozedur `onLoad` aus (sofern eine solche im Tag `body` vorhanden ist), wenn der Benutzer den Befehl oder die Aktion in einem Menü für die Erweiterungstypen **Befehl** und **Verhalten** aufruft.

Dreamweaver führt den Code in der Ereignisprozedur `onLoad` aus (sofern eine solche im Tag `body` vorhanden ist), falls im Textkörper des Dokuments ein Formular für Objekterweiterungen enthalten ist.

In den folgenden Erweiterungen ignoriert Dreamweaver die Ereignisprozedur `onLoad` im Tag `body`:

- Datenübersetzer
- Eigenschafteninspektor
- Schwebendes Bedienfeld

Bei allen Erweiterungen führt Dreamweaver Code in anderen Ereignisprozeduren aus (zum Beispiel `onBlur="alert('Dieses Feld ist erforderlich.')"`), wenn der Benutzer mit den Formularfeldern arbeitet, an die sie angehängt sind.

Dreamweaver unterstützt die Verwendung von Ereignisprozeduren in Hyperlinks. Die Syntax der Ereignisprozedur in Hyperlinks muss wie folgt verwendet werden:

```
<a href="#" onMouseDown=alert('hi')>link text</a>
```

Plug-Ins (permanent auf `play` eingestellt) werden im `BODY` von Erweiterungen unterstützt. Die Anweisung `document.write()`, Java-Applets und Microsoft ActiveX-Steuerelemente werden in Erweiterungen nicht unterstützt.

Hilfe anzeigen

Wenn Sie die Funktion `displayHelp()`, die Teil verschiedener Erweiterungs-APIs ist, in Ihre Erweiterung aufnehmen, führt Dreamweaver die beiden folgenden Aktionen aus:

- Eine Hilfeschatfläche wird zur Oberfläche hinzugefügt.
- Die Funktion `displayHelp()` wird aufgerufen, wenn der Benutzer auf die Hilfeschatfläche klickt.

Sie müssen den Textteil der Funktion `displayHelp()` schreiben, um die Hilfe anzuzeigen. Die Kodierung der Funktion `displayHelp()` legt fest, wie die Erweiterung die Hilfe anzeigt. Sie können die Funktion `dreamweaver.browseDocument()` aufrufen, um eine Datei in einem Browser zu öffnen, oder ein benutzerdefiniertes Verfahren zum Anzeigen der Hilfe anzugeben, wie beispielsweise die Anzeige von Meldungen in einem anderen absolut positionierten Element in Warnfeldern.

Im folgenden Beispiel ist die Verwendung der Funktion `displayHelp()` zum Anzeigen der Hilfe durch den Aufruf von `dreamweaver.browseDocument()` gezeigt:

```
// Die folgende Instanz von displayHelp() öffnet einen Browser, um eine Datei anzuzeigen,
// in der die Verwendung der Datei erläutert wird.
function displayHelp() {

    var myHelpFile = dw.getConfigurationPath() + "ExtensionsHelp/myExtHelp.htm";
    dw.browseDocument(myHelpFile);
}
```

Erweiterungen lokalisieren

Verwenden Sie die folgenden Techniken, um die Übersetzung der Erweiterungen in andere Sprachen zu vereinfachen.

- Teilen Sie Erweiterungen in HTML- und JavaScript-Dateien auf. Die HTML-Dateien können repliziert und lokalisiert werden, die JavaScript-Dateien dagegen werden nicht lokalisiert.
- Definieren Sie keine Anzeigestrings in den JavaScript-Dateien (überprüfen Sie Warnungen und den UI-Code). Extrahieren Sie alle lokalisierbaren Strings in separate XML-Dateien im Dreamweaver-Ordner **Configuration\Strings**.
- Fügen Sie, mit Ausnahme der erforderlichen Ereignisprozeduren, keinen JavaScript-Code in die HTML-Dateien ein. Dadurch wird verhindert, dass ein Fehler mehrmals für mehrere Übersetzungen behoben werden muss, nachdem die HTML-Dateien repliziert und in andere Sprachen übersetzt wurden.

XML-String-Dateien

Speichern Sie alle Strings in XML-Dateien im Dreamweaver-Ordner **Configuration\Strings**. Wenn Sie viele verwandte Erweiterungsdateien installieren, können Sie auf diese Weise die Strings in einer einzelnen XML-Datei gemeinsam nutzen. Falls zutreffend, können Sie auch aus C++- und JavaScript-Erweiterungen auf denselben String Bezug nehmen.

Sie können beispielsweise eine Datei mit dem Namen `myExtensionStrings.xml` erstellen. Das folgende Beispiel stellt das Format der Datei dar:

```
<strings>
  <!-- Fehler für Funktion X -->
  <string id="featureX/subProblemY" value="There was a with X when you did Y. Try not to
    do Y!"/>
  <string id="featureX/subProblemZ" value="There was another problem with X, regarding Z.
    Don't ever do Z!"/>
</strings>
```

Die JavaScript-Dateien können jetzt auch auf diese übersetzbaren Strings verweisen, indem die Funktion `dw.loadString()` aufgerufen wird, wie im folgenden Beispiel dargestellt:

```
function initializeUI()
{
    ...
    if (problemYhasOccured)
    {
        alert(dw.loadString("featureX/subProblemY"));
    }
}
```

Sie können im Stringbezeichner Schrägstriche (/) verwenden, jedoch keine Leerzeichen. Durch die Verwendung von Schrägstrichen können Sie eine Ihren Anforderungen entsprechende Hierarchie erstellen und alle Strings in einer einzigen XML-Datei unterbringen.

Hinweis: Dateien, die im Ordner **Configuration\Strings** mit „cc“ beginnen, sind *Contribute-Dateien*. So handelt es sich bei der Datei **ccSiteStrings.xml** zum Beispiel um eine *Contribute-Datei*.

Lokalisierbare Strings mit eingebetteten Werten

In einige Anzeigestrings sind Werte eingebettet. Diese Strings können Sie mithilfe der Funktion `errMsg()` anzeigen. Die Funktion `errMsg()`, die der Funktion `printf()` in C ähnelt, befindet sich in der Datei `string.js` im Ordner **Configuration\Shared\MM\Scripts\CMN**. Verwenden Sie als Platzhalterzeichen das Prozentzeichen (%) und `s`, um anzugeben, wo Werte im String auftreten, und übergeben Sie die String- und Variablennamen dann als Argumente an `errMsg()`. Beispiel:

```
<string id="featureX/fileNotFoundInFolder" value="File %s could not be found in folder %s."/>
```

Im folgenden Beispiel ist dargestellt, wie der String zusammen mit den eingebetteten Variablen an die Funktion `alert()` übergeben wird.

```
if (fileMissing)
{
    alert( errMsg(dw.loadString("featureX/fileNotFoundInFolder"), fileName,
        folderName) );
}
```

Mit dem Extension Manager arbeiten

Wenn Sie Erweiterungen für andere Benutzer erstellen, müssen Sie sie nach den Richtlinien komprimieren, die Sie auf der Macromedia Exchange-Website (www.adobe.com/go/exchange_de) im Abschnitt **Hilfe > Eine Erweiterung erstellen** finden. Nachdem Sie eine Erweiterung im Extension Manager geschrieben und getestet haben, wählen Sie den Befehl **Datei > Erweiterung erstellen**. Sobald die Erweiterung komprimiert ist, können Sie sie aus dem Extension Manager an Exchange senden, indem Sie den Befehl **Datei > Erweiterung senden** wählen.

Der Adobe Extension Manager ist im Lieferumfang von Dreamweaver enthalten. Genauere Informationen über seine Verwendung finden Sie in den zugehörigen Hilfedateien und auf der Adobe Exchange-Website.

Kapitel 5: Benutzeroberflächen für Erweiterungen

Die meisten Erweiterungen dienen dazu, dass der Benutzer auf einer Benutzeroberfläche (User Interface = UI) Informationen eingeben kann. Wenn Sie beispielsweise eine Eigenschafteninspektor-Erweiterung für das Tag `marquee` erstellen, müssen Sie den Benutzern eine Möglichkeit einräumen, Attribute wie die Richtung oder Höhe anzugeben. Wenn Sie Ihre Erweiterung zur Zertifizierung durch Adobe einsenden möchten, müssen Sie die Richtlinien befolgen, die in den Extension Manager-Dateien auf der Adobe Exchange-Website (www.adobe.com/go/exchange_de) festgelegt sind. Der Zweck dieser Richtlinien ist es nicht, Ihre Kreativität einzuschränken. Sie sollen nur gewährleisten, dass zertifizierte Erweiterungen in der Benutzeroberfläche von Adobe Dreamweaver CS3 effektiv funktionieren und die Funktionalität der Benutzeroberfläche für Erweiterungen nicht beeinträchtigt wird.

Benutzeroberflächen von Erweiterungen gestalten

In der Regel wird mit einer Erweiterung ein von Benutzern häufig verwendeter Arbeitsschritt ausgeführt. Bestimmte Teile von Arbeitsschritten wiederholen sich häufig. Mithilfe von Erweiterungen lassen sich solche wiederholten Aktionen automatisieren. Manche Teile eines Arbeitsschritts oder bestimmte Attribute des Codes sind jedoch variabel, sodass die Abläufe der Erweiterung unterschiedlich sein müssen. Sie erstellen eine Benutzeroberfläche, damit die Benutzer die variablen Werte eingeben können.

So könnten Sie beispielsweise eine Erweiterung zum Aktualisieren eines Webkatalogs erstellen. Die Benutzer müssen Werte für die Bildquellen, die Beschreibung der Objekte und die Preise regelmäßig ändern. Die Werte ändern sich zwar, doch die Abläufe zum Abrufen dieser Werte und zum Formatieren dieser Informationen für die Anzeige auf der Website bleiben immer gleich. Eine einfache Erweiterung kann das Formatieren automatisieren, während die Benutzer die neuen, aktualisierten Werte für die Bildquellen, Objektbeschreibungen und Preise weiterhin manuell eingeben. Eine etwas komplexere Erweiterung kann diese Werte regelmäßig aus einer Datenbank abrufen.

Ihre Erweiterungs-UI soll also die Informationen abrufen, die von den Benutzern eingegeben werden. Diese Informationen sind zum Regeln der variablen Aspekte eines wiederholten Arbeitsschritts erforderlich, den die Erweiterung übernimmt. Dreamweaver unterstützt HTML- und JavaScript-Formularelemente als grundlegende Bausteine bei der Erstellung von Steuerelementen für die Benutzeroberfläche und stellt die Benutzeroberfläche mithilfe einer eigenen HTML-Anzeigefunktion dar. Eine einfache Benutzeroberfläche für eine Erweiterung besteht z. B. aus einer HTML-Datei, die eine zweiseitige Tabelle mit Textbeschreibungen und Formulareingabefeldern beschreibt.

Beim Entwurf einer Erweiterung sollten Sie ermitteln, welche Variablen erforderlich sind und welche Formularelemente diese am besten behandeln können.

Beim Entwerfen der Benutzeroberfläche für eine Erweiterung sollten Sie die folgenden Richtlinien beachten:

- Fügen Sie den gewünschten Namen für die Erweiterung in das `title`-Tag Ihrer HTML-Datei ein. Dreamweaver zeigt den Namen in der Titelleiste der Erweiterung an.
- Ordnen Sie Bezeichnungen auf der linken Seite Ihrer Benutzeroberfläche rechtsbündig an, die Textfelder hingegen linksbündig auf der rechten Seite. Dadurch kann der Benutzer den Anfang der einzelnen Textfelder leichter erkennen. Hinter dem Textfeld kann minimaler Text folgen, z. B. eine Erklärung oder eine Maßeinheit.
- Ordnen Sie Bezeichnungen von Kontrollkästchen und Optionsschaltern auf der rechten Seite linksbündig an.
- Bei lesbarem Code sollten Sie Ihren Textfeldern logische Namen zuweisen. Wenn Sie mit Dreamweaver die Benutzeroberfläche Ihrer Erweiterung erstellen, können Sie den Eigenschafteninspektor oder den Quick Tag-Editor verwenden, um den Feldern Namen zuzuweisen.

In der Regel testen Sie nach dem Erstellen der Benutzeroberfläche den Erweiterungscode, um festzustellen, ob er die folgenden Schritte wie vorgesehen erfüllt:

- Abrufen der Werte aus den Textfeldern
- Festlegen der Standardwerte für die Textfelder oder Ermitteln der Werte aus der Auswahl
- Vornehmen von Änderungen am Benutzerdokument

Steuerung der HTML-Darstellung in Dreamweaver

In den Versionen bis Dreamweaver 4 wurden die Steuerelemente in Formularen mit mehr Raum umgeben als bei Microsoft Internet Explorer und Netscape Navigator. Die Formularsteuerelemente in den Benutzeroberflächen von Erweiterungen wurden mit zusätzlichem Raum dargestellt, da Dreamweaver zur Darstellung dieser Benutzeroberflächen die integrierte HTML-Rendering-Engine verwendet.

In späteren Dreamweaver-Versionen stimmt die Darstellung von Formularsteuerelementen nun besser mit der Browseranzeige überein. Um die verbesserte Darstellung zu nutzen, müssen Sie in Ihren Erweiterungsdateien eine der drei neuen DOCTYPE-Anweisungen wie folgt verwenden:

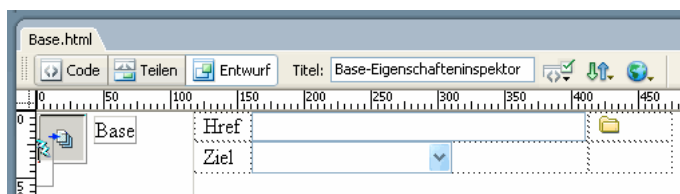
```
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//dialog">  
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//floater">  
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//pi">
```

In der Regel müssen sich die DOCTYPE-Anweisungen in der ersten Zeile eines Dokuments befinden. Um jedoch Konflikte mit erweiterungsspezifischen Direktiven zu vermeiden, die sich in früheren Versionen in der ersten Zeile einer Datei befinden mussten (z. B. der Kommentar am Anfang einer Eigenschafteninspektor-Datei oder die Direktive MENU-LOCATION=NONE in einem Befehl), können DOCTYPE-Anweisungen und -Direktiven nun in beliebiger Reihenfolge verwendet werden, sofern sie sich vor dem einleitenden HTML-Tag befinden.

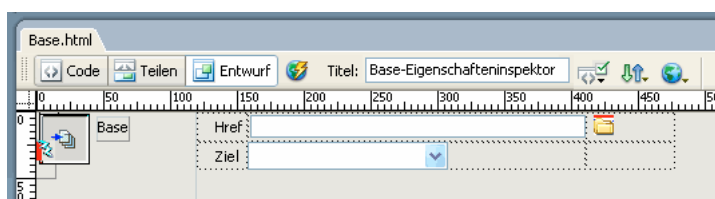
Neben der besseren Übereinstimmung der Benutzeroberflächen von Erweiterungen mit integrierten Dialog- und Bedienelementen können Sie mithilfe der neuen DOCTYPE-Anweisungen Ihre Erweiterungen in der Dreamweaver-Entwurfsansicht genau so sehen, wie sie den Benutzern angezeigt werden.

Sie können drei Beispiele sehen, in denen die Schreibweise des Dialogfelds verwendet wird, und zwar in den folgenden Dateien im Ordner **Configuration\Commands**: **CFLoginWizard.htm**, **TeamAdminDlgDates.html** und **TeamAdminDlgDays.html**

Im Folgenden ist der Basis-Eigenschafteninspektor *ohne* die DOCTYPE-Anweisung (verbesserte Darstellung der Formularsteuerelemente) und *mit* der DOCTYPE-Anweisung aufgeführt.



Der Basis-Eigenschafteninspektor, wie er in der Entwurfsansicht ohne die DOCTYPE-Anweisung angezeigt wird.



Der Basis-Eigenschafteninspektor, wie er in der Entwurfsansicht mit der DOCTYPE-Anweisung (und nach einigen Anpassungen für die neue Darstellungsform) angezeigt wird.

Benutzerdefinierte UI-Elemente in Erweiterungen verwenden

Über die standardmäßigen HTML-Formularelemente hinaus unterstützt Dreamweaver benutzerdefinierte Steuerelemente für die Benutzeroberfläche, so dass Sie flexible, professionell wirkende Oberflächen gestalten können:

- Bearbeitbare Auswahllisten (auch Kombinationsfelder genannt), durch die Sie die Funktionsweise von Auswahllisten mit der von Textfeldern kombinieren können.
- Datenbank-Steuerelemente, die die Anzeige von Datenhierarchien und Datenfeldern erleichtern.
- Struktursteuerelemente, die Informationen in erweiter- und reduzierbaren Knoten anordnen.
- Steuerelemente zur Farbauswahl, mit denen Sie Ihre Erweiterungen um Oberflächen zur Farbauswahl ergänzen können.

Bearbeitbare Auswahllisten

Benutzeroberflächen von Erweiterungen enthalten häufig Popuplisten, die mithilfe des Tags `select` definiert sind. In Dreamweaver können die Popuplisten in Erweiterungen zur Bearbeitung verfügbar gemacht werden, indem `editable="true"` zum Tag `select` hinzugefügt wird. Um einen Standardwert anzugeben, legen Sie das Attribut `editText` und den Wert fest, den die Auswahlliste anzeigen soll.

Im folgenden Beispiel werden die Einstellungen für eine bearbeitbare Auswahlliste dargestellt:

```
<select name="travelOptions" style="width:250px" editable="true" editText="other  
  (please specify)">  
<option value="plane">plane</option>  
<option value="car">car</option>  
<option value="bus">bus</option>  
</select>
```

Wenn Sie Auswahllisten in Ihren Erweiterungen verwenden, überprüfen Sie, ob ein bearbeitbares Attribut vorhanden ist und welchen Wert es hat. Wenn kein Wert vorhanden ist, gibt die Auswahlliste den Standardwert `false` zurück, was anzeigt, dass die Auswahlliste nicht bearbeitbar ist.

Wie standardmäßige, nicht bearbeitbare Listen haben bearbeitbare Listen die Eigenschaft `selectedIndex` (siehe „Objekte, Eigenschaften und Methoden des Dreamweaver-DOM“ auf Seite 88). Diese Eigenschaft gibt den Wert 1 zurück, wenn das Textfeld ausgewählt ist.

Damit der Wert eines aktiven bearbeitbaren Textfelds in eine Erweiterung gelesen wird, muss der Wert der Eigenschaft `editText` gelesen werden. Die Eigenschaft `editText` gibt den vom Benutzer in das bearbeitbare Textfeld eingegebenen String zurück, den Wert des Attributs `editText` oder einen leeren String, falls kein Text eingegeben und kein Wert für die Eigenschaft `editText` angegeben wurde.

Dreamweaver fügt dem Tag `select` die folgenden benutzerdefinierten Attribute zum Steuern bearbeitbarer Popuplisten- hinzu:

Attributname	Beschreibung	Akzeptierte Werte
<code>editable</code>	Deklariert, dass die Popupliste über einen bearbeitbaren Textbereich verfügt.	Ein Boolescher Wert (<code>true</code> oder <code>false</code>)
<code>editText</code>	Beinhaltet Text in einem bearbeitbaren Textbereich oder legt ihn fest.	Ein String mit beliebigem Wert.

Hinweis: Bearbeitbare Auswahllisten stehen in Dreamweaver zur Verfügung.

Im folgenden Beispiel wird mithilfe gängiger JavaScript-Funktionen eine Befehlserweiterung erstellt, die eine bearbeitbare Auswahlliste enthält:

So erstellen Sie ein Beispiel:

1 Erstellen Sie in einem Texteditor eine neue, leere Datei.

2 Geben Sie den folgenden Code ein:

```
<html>
<head>
  <title>Editable Dropdown Test</title>
  <script language="javascript">
    function getAlert()
    {

      var i=document.myForm.mySelect.selectedIndex;
      if (i>=0)
      {
        alert("Selected index: " + i + "\n" + "Selected text " +
          document.myForm.mySelect.options[i].text);
      }
      else
      {
        alert("Nothing is selected" + "\n" + "or you entered a value");
      }
    }
    function commandButtons()
    {
      return new Array("OK", "getAlert()", "Cancel", "window.close()");
    }
  </script>
</head>

<body>
<div name="test">
<form name="myForm">
<table>
  <tr>
    <td colspan="2">
      <h4>Select your favorite</h4>
    </td>
  </tr>
  <tr>
    <td>Sport:</td>
    <td>
      <select name="mySelect" editable="true" style="width:150px"
        editText="Editable Text">
        <option> Baseball</option>
        <option> Football </option>
        <option> Soccer </option>
      </select>
    </td>
  </tr>
</table>
</form>
</div>
</body>
</html>
```

3 Speichern Sie die Datei unter dem Namen **EditableSelectTest.htm** im Dreamweaver-Ordner **Configuration/Commands**.

So testen Sie das Beispiel:

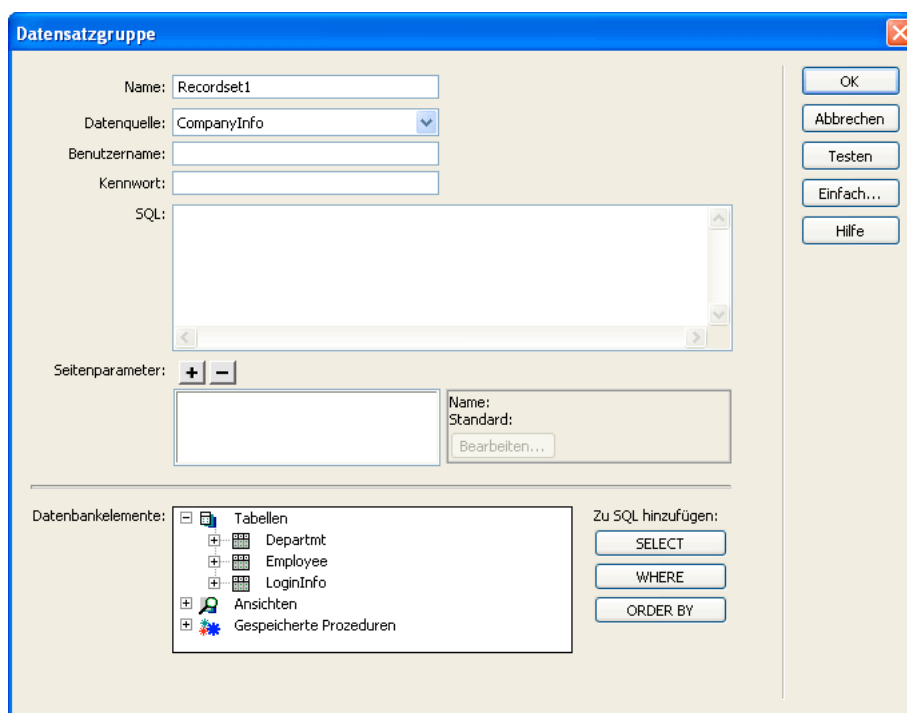
- 1 Starten Sie Dreamweaver neu.
- 2 Wählen Sie **Befehle > EditableSelectTest**.

Wenn Sie einen Wert in der Liste auswählen, wird eine Meldung mit dem Index des Werts und dem Text angezeigt. Wenn Sie einen Wert eingeben, wird gemeldet, dass keine Auswahl getroffen wurde.

Datenbank-Steuerelemente

In Dreamweaver können Sie das HTML-Tag `select` verwenden, um Datenbank-Struktursteuerelemente zu erstellen. Sie können auch ein variables Tabellen-Steuerelement hinzufügen. Das Datenbank-Struktursteuerelement zeigt Datenbankschemas an, das variable Tabellen-Steuerelement dagegen tabellarische Informationen.

In der folgenden Abbildung sehen Sie ein erweitertes Dialogfeld **Datensatzgruppe**, das ein Datenbank-Struktursteuerelement und ein variables Tabellen-Steuerelement verwendet:

**Datenbank-Struktursteuerelemente hinzufügen**

Das Datenbank-Struktursteuerelement hat folgende Attribute:

Attributname	Beschreibung
name	Name des Datenbank-Struktursteuerelements
control.style	Breite und Höhe in Pixel
type	Steuerelementtyp
connection	Name der im Connection Manager definierten Datenbankverbindung; falls leer, ist auch das Steuerelement leer.
noexpandbuttons	Wenn dieses Attribut angegeben ist, enthält das Struktursteuerelement die Plus- (+) und Minuszeichen (-) bzw. auf dem Macintosh die entsprechenden Pfeile als Symbole für das Erweitern bzw. Reduzieren nicht. Dieses Attribut ist beim Erstellen von Steuerelementen für Listen mit mehreren Spalten nützlich.
showheaders	Wenn dieses Attribut angegeben ist, zeigt das Datenbank-Steuerelement eine Kopfzeile mit den Namen aller Spalten an.

Option-Tags innerhalb des Tags `select` werden ignoriert.

Um einem Dialogfeld ein Datenbank-Struktursteuerelement hinzuzufügen, können Sie den folgenden Beispielcode verwenden und die entsprechenden Ersetzungen vornehmen:

```
<select name="DBTree" style="width:400px;height:110px" ~  
type="mmdatabasetree" connection="ConnectionName" noexpandbuttons showHeaders></select>
```

Sie können das Attribut `connection` ändern, um ausgewählte Daten abzurufen und in der Struktur darzustellen. Sie können das Attribut `DBTreeControl` als JavaScript-Wrapper-Objekt für das neue Tag verwenden. Weitere Beispiele finden Sie in der Datei `DBTreeControlClass.js`, die sich im Ordner **Configuration\Shared\Scripts** befindet.

Variable Tabellen-Steuerelemente hinzufügen

Das variable Tabellen-Steuerelement hat folgende Attribute:

Attributname	Beschreibung
name	Name des variablen Tabellen-Steuerelements
style	Breite des Steuerelements in Pixel
Typ	Steuerelementtyp
columns	Jede Spalte muss einen Namen besitzen, getrennt durch Kommas.
columnWidth	Breite jeder Spalte, getrennt durch Kommas. Wenn die Breite nicht angegeben wird, haben alle Spalten die gleiche Breite.

Im folgenden Beispiel wird einem Dialogfeld ein einfaches variables Tabellen-Steuerelement hinzugefügt:

```
<select name="ParamList" style="width:515px;" ~  
type="mmparameterlist" columns="Name,SQL Data ~  
Type,Direction,Default Value,Run-time Value" size=6></select>
```

Im folgenden Beispiel wird ein variables Tabellen-Steuerelement erstellt, das 500 Pixel breit ist und fünf Spalten unterschiedlicher Breite besitzt:

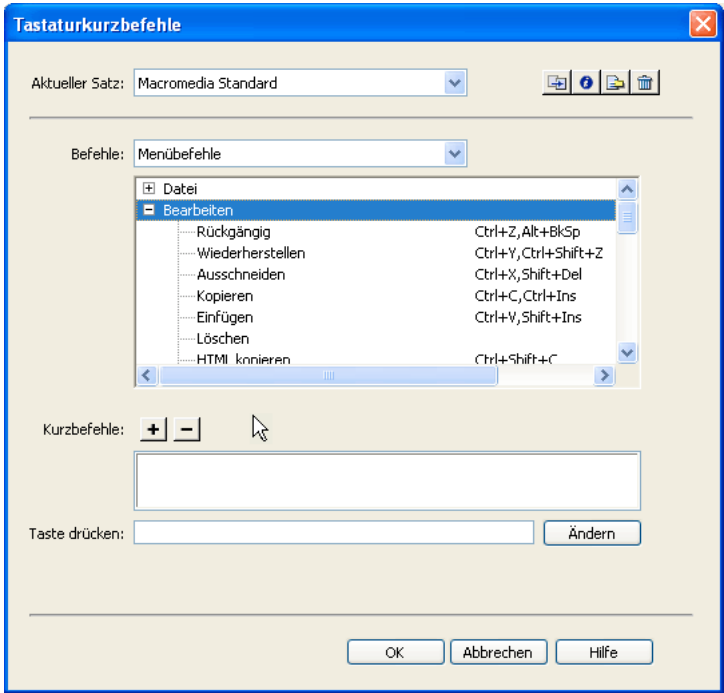
```
<select  
  name="ParamList"  
  style="width:500px;"  
  type="mmparameterlist"  
  columns="Name,SQL Data Type,Direction, Default Value,Run-time Value"  
  columnWidth="100,25,11,"  
  size=6>
```

In diesem Beispiel werden zwei leere Spalten mit einer Breite von 182 Pixel erstellt. (Die festgelegten Spalten ergeben zusammen 136. Die Gesamtbreite des variablen Tabellen-Steuerelements beträgt 500. Der verbleibende Raum, nachdem die ersten drei Spalten positioniert wurden, beträgt 364. Es bleiben zwei Spalten übrig: 364 geteilt durch 2 ergibt 182.)

Dieses variable Tabellen-Steuerelement verfügt zudem über ein JavaScript-Wrapper-Objekt, das verwendet wird, um auf die Daten des Tabellen-Steuerelements zuzugreifen und diese zu verändern. Sie finden die Implementierung in der Datei `GridControlClass.js`, die sich im Ordner **Configuration\Shared\MM\Scripts\Class** befindet.

Struktursteuerelemente hinzufügen

Struktursteuerelemente zeigen Daten als Hierarchie an, sodass der Benutzer Knoten in der Struktur erweitern und reduzieren kann. Mithilfe des Tags `MM: TREECONTROL` können Sie Steuerelemente für alle Arten von Informationen erstellen. Im Gegensatz zu dem im Abschnitt „Datenbank-Struktursteuerelemente hinzufügen“ auf Seite 79 beschriebenen Datenbank-Struktursteuerelement ist keine Verknüpfung mit einer Datenbank erforderlich. In Dreamweaver verwendet der Tastaturbefehl-Editor das Struktursteuerelement wie folgt:



Struktursteuerelemente erstellen

Das Tag `MM: TREECONTROL` erstellt ein Struktursteuerelement und kann mithilfe zusätzlicher Tags Strukturen wie folgt hinzufügen:

- `MM: TREECOLUMN` ist ein leeres, optionales Tag, das eine Spalte in dem Struktursteuerelement definiert.
- `MM: TREENODE` ist ein optionales Tag, das einen Knoten in der Struktur definiert. Dieses Tag ist nicht leer und kann nur andere Tags `M: TREENODE` enthalten.

Die Tags `MM: TREECONTROL` haben die folgenden Attribute:

Attributname	Beschreibung
Name	Name des Struktursteuerelements
size	Optional. Anzahl der im Steuerelement angezeigten Spalten; die Standardeinstellung ist 5 Spalten.
theControl	Optional. Wenn die Anzahl der Knoten des Attributs <code>theControl</code> den Wert des <code>size</code> -Attributs überschreitet, werden Bildlaufleisten angezeigt.
multiple	Optional. Ermöglicht Mehrfachauswahl; die Standardeinstellung ist die Einfachauswahl.
Stil	Optional. Stildefinition für Höhe und Breite des Struktursteuerelements. Wenn sie angegeben ist, hat sie Vorrang vor dem Attribut <code>size</code> .
noheaders	Optional. Legt fest, dass die Kopfzeilen der Spalten nicht angezeigt werden.

Die Tags `MM: TREECOLUMN` haben die folgenden Attribute:

Attributname	Beschreibung
Name	Name der Spalte
value	String, der in der Kopfzeile der Spalte angezeigt werden soll.
width	Breite der Spalte in Pixel (keine Prozentangabe möglich); die Standardeinstellung ist 100.
align	Optional. Legt fest, ob der Text in der Spalte linksbündig, rechtsbündig oder zentriert ausgerichtet werden soll; die Standardeinstellung ist linksbündig.
state	Legt fest, ob die Spalte ein- oder ausgeblendet ist.

Zur besseren Lesbarkeit sollten die Tags `TREECOLUMN` immer direkt im Anschluss an das Tag `MM: TREECONTROL` folgen, wie im folgenden Beispiel zu sehen:

```
<MM: TREECONTROL name="tree1">
<MM: TREECOLUMN name="Column1" width="100" state="visible">
<MM: TREECOLUMN name="Column2" width="80" state="visible">
...
</MM: TREECONTROL>
```

Die `MM: TREENODE`-Attribute werden in der folgenden Tabelle erläutert:

Attributname	Beschreibung
Name	Name des Knotens.
Wert	Enthält den Inhalt des angegebenen Knotens. Wenn mehrere Spalten enthalten sind, wird dieser String durch senkrechte Striche getrennt. Platzieren Sie einen einzelnen Leerschritt vor dem senkrechten Strich (), wenn Sie eine leere Spalte angeben möchten.
Zustand	Legt fest, dass der Knoten mithilfe der Strings "expanded" oder "collapsed" erweitert oder reduziert wird.
selected	Wenn die Struktur das Attribut <code>MULTIPLE</code> enthält, können Sie mehrere Knoten auswählen, indem Sie das Attribut für mehrere Knoten der Struktur festlegen.
icon	Optional. Index des zu verwendenden integrierten Symbols, mit 0 beginnend: 0 = kein Symbol; 1 = Dreamweaver-Dokumentsymbol; 2 = Mehrfachdokumentsymbol

Bei dem folgenden Struktursteuerelement sind alle Knoten erweitert.

```
<mm:treecontrol name="test" style="height:300px;width:300px">

<mm:treenode value="rootnode1" state="expanded">
<mm:treenode value="node1" state="expanded"></mm:treenode>
<mm:treenode value="node3" state="expanded"></mm:treenode>
</mm:treenode>

<mm:treenode value="rootnode2" state="expanded">
<mm:treenode value="node2" state="expanded"></mm:treenode>
<mm:treenode value="node4" state="expanded"></mm:treenode>
</mm:treenode>

</mm:treecontrol>
```


Inhalt von Struktursteuerelementen bearbeiten

Struktursteuerelemente und die darin enthaltenen Knoten werden als HTML-Tags implementiert. Sie werden von Dreamweaver analysiert und in der Dokumentstruktur gespeichert. Diese Tags können genauso wie alle anderen Dokumentknoten bearbeitet werden. Weitere Informationen über DOM-Funktionen und -Methoden finden Sie in „Das Dreamweaver-Dokumentobjektmodell“ auf Seite 87.

Knoten hinzufügen. Um einem vorhandenen Struktursteuerelement im Programm einen Knoten hinzuzufügen, legen Sie die Eigenschaft `innerHTML` des Tags `MM:TREECONTROL` oder eines der vorhandenen `MM:TREENODE`-Tags fest. Durch das Festlegen der Eigenschaft `innerHTML` eines Strukturknotens wird ein verschachtelter Knoten erstellt.

Im folgenden Beispiel wird der obersten Ebene einer Struktur ein Knoten hinzugefügt:

```
var tree = document.myTreeControl;
//unten in der Struktur einen Knoten der obersten Ebene hinzufügen
tree.innerHTML = tree.innerHTML + '<mm:treenode name="node3" value="node3">';
```

Untergeordnete Knoten hinzufügen Um zum aktuell ausgewählten Knoten einen untergeordneten Knoten hinzuzufügen, legen Sie die Eigenschaft `innerHTML` des ausgewählten Knotens fest.

Im folgenden Beispiel wird zum aktuell ausgewählten Knoten ein untergeordneter Knoten hinzugefügt:

```
var tree = document.myTreeControl;
var selNode = tree.selectedNodes[0];
//die Auswahl des Knotens aufheben, damit der neue ausgewählt werden kann
selNode.removeAttribute("selected");
//oben bei den untergeordneten Knoten des ausgewählten Knotens den neuen Knoten hinzufügen
selNode.innerHTML = '<mm:treenode name="item10" value="New item11" expanded selected>' + ~
selNode.innerHTML;
```

Knoten löschen Um den aktuell ausgewählten Knoten aus der Dokumentstruktur zu löschen, verwenden Sie die Eigenschaft `innerHTML` oder `outerHTML`.

Im folgenden Beispiel wird der ausgewählte Knoten samt aller untergeordneten Knoten gelöscht:

```
var tree = document.myTreeControl;
var selNode = tree.selectedNodes[0];
selNode.outerHTML = "";
```

Steuerelemente zur Farbauswahl

Neben den standardmäßigen Eingabetypen wie Text, Kontrollkästchen und Schaltfläche unterstützt Dreamweaver in Erweiterungen den Dokumenttyp `mmcolorbutton`.

Durch die Einstellung `<input type="mmcolorbutton">` im Code wird ein Farbwähler auf der Benutzeroberfläche angezeigt. Sie können die Standardfarbe für den Farbwähler festlegen, indem Sie ein Wertattribut im Input-Tag angeben. Wenn kein Wert angegeben ist, wird der Farbwähler standardmäßig grau angezeigt und die Werteigenschaft des Eingabeobjekts gibt einen leeren String zurück.

Das folgende Beispiel zeigt einen gültigen `mmcolorbutton`-Tag:

```
<input type="mmcolorbutton" name="colorbutton" value="#FF0000">
<input type="mmcolorbutton" name="colorbutton" value="teal">
```

Ein Steuerelement für die Farbauswahl besitzt ein `onChange`-Ereignis. Dieses wird durch das Ändern der Farbe ausgelöst.

Es kann sinnvoll sein, ein Textfeld und eine Farbauswahl zu synchronisieren. Im folgenden Beispiel wird ein Textfeld erstellt, wobei die Farbe des Textfelds und die Farbe des Farbwählers synchronisiert sind.

```
<input type="mmcolorbutton" name="fgcolorPicker" onChange="document.fgcolorText.value=this.value">
<input type="text" name="fgcolorText" onBlur="document.fgcolorPicker.value=this.value">
```

Sobald der Benutzer in diesem Beispiel den Wert des Textfelds ändert und mit der Maus oder der Tabulatortaste an eine andere Stelle wechselt, wird der Farbwähler aktualisiert und zeigt die im Textfeld angegebene Farbe an. Jedes Mal, wenn der Benutzer mit dem Farbwähler eine neue Farbe wählt, wird das Textfeld aktualisiert und zeigt den Hexadezimalwert dieser Farbe an.

Flash-Inhalt zu Dreamweaver hinzufügen

Flash-Inhalt (SWF-Dateien) kann in der Dreamweaver-Benutzeroberfläche als Teil eines Objekts oder Befehls angezeigt werden. Diese Flash-Unterstützung ist besonders nützlich, wenn Sie Erweiterungen erstellen, die Flash-Formulare, Animationen, ActionScript oder anderen Flash-Inhalt verwenden.

In der Regel bestimmen Sie selbst, wie Dialogfelder von Dreamweaver-Objekten und -Befehlen angezeigt werden (in „Objekte der Einfügleiste“ auf Seite 95 finden Sie weitere Informationen zum Erstellen von Objekten und in „Befehle“ auf Seite 123 Informationen zu Befehlen). Verwenden Sie dazu das Tag `form` mit dem Tag `object`, um den Flash-Inhalt in einem Dreamweaver-Dialogfeld einzubetten.

Bespiel für ein einfaches Flash-Dialogfeld

In diesem Beispiel erstellen Sie mithilfe von Dreamweaver einen neuen Befehl, mit dem eine SWF-Datei mit dem Namen „myFlash.swf“ angezeigt wird, wenn der Benutzer im Menü **Befehle** auf den Befehl klickt. Wenn Sie vorher weitere Angaben zum Erstellen von Befehlen benötigen, lesen Sie „Befehle“ auf Seite 123.

Hinweis: In diesem Beispiel wird vorausgesetzt, dass die SWF-Datei *myFlash.swf* sich bereits im Ordner **Configuration\Commands** des Installationsordners von Dreamweaver befindet. Um das Beispiel mit Ihrer SWF-Datei durchzuführen, speichern Sie die SWF-Datei im Anwendungsordner **Commands**, und ersetzen Sie den Namen Ihrer Datei durch „myFlash.swf“.

Öffnen Sie in Dreamweaver eine neue einfache HTML-Datei (es handelt sich dabei um Ihre Befehlsdefinitionsdatei). Geben Sie zwischen dem öffnenden und schließenden `title`-Tag den Namen **My Flash Movie** ein, sodass der Anfang der Seite folgendermaßen aussieht:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>My Flash Movie</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
```

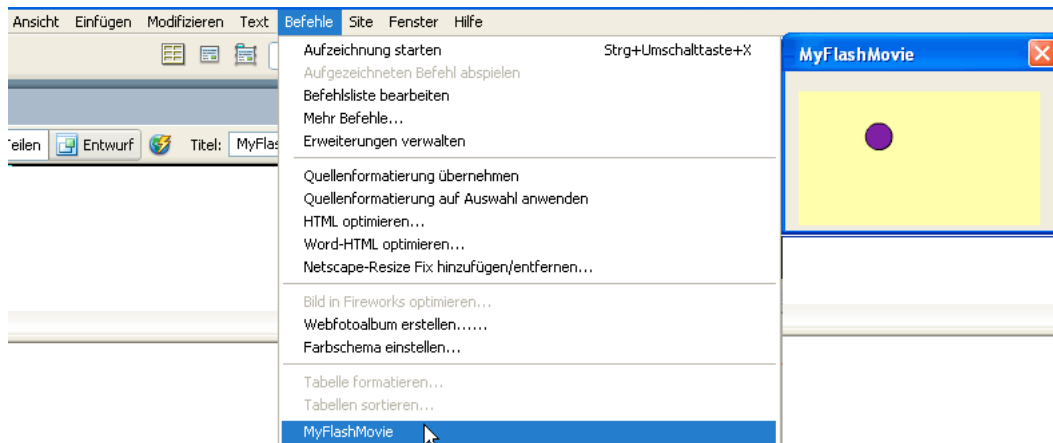
Speichern Sie nun die Datei unter dem Namen **MyFlashMovie.htm** im Anwendungsordner **Configuration\Commands**. Schließen Sie die Datei jedoch noch nicht. Da Sie die Datei nun gespeichert haben, können Sie den Flash-Inhalt in einem relativen Pfad einbetten. Andernfalls verwendet Dreamweaver einen absoluten Pfad.

Fügen Sie im HTML-Dokument zwischen dem öffnenden und schließenden `body`-Tag einen öffnenden und schließenden `form`-Tag ein. Fügen Sie anschließend innerhalb der `form`-Tags mithilfe der Befehle **Einfügen > Medien > Flash** den Flash-Inhalt in die Befehlsdefinitionsdatei ein. Wählen Sie bei entsprechender Aufforderung die SWF-Datei im Ordner **Commands** aus, und klicken Sie auf **OK**. Ihre Befehlsdefinitionsdatei sollte nun dem folgenden Beispiel entsprechen (die Attribute `width` und `height` können entsprechend den Eigenschaften der SWF-Datei hiervon abweichen):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>My Flash Movie</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
```

```
<body>
<body>
<form>
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase="http:
//download.Macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,29,0"
width="200" height="100">
<param name="movie" value="myFlash.swf">
<param name="quality" value="high">
<embed src="myFlash.swf" quality="high" pluginspage="http://www.Macromedia.com/go/
getflashplayer" type="application/x-shockwave-flash" width="200" height="100"></embed>
</object>
</form>
</body>
</html>
```

Speichern Sie die Datei noch einmal. Beenden Sie anschließend Dreamweaver, und starten Sie das Programm neu. Wählen Sie **Befehle > MyFlashMovie**. Der Flash-Inhalt wird im Dreamweaver-Dialogfeld wie in der folgenden Abbildung angezeigt:



Dieses Beispiel zeigt eine einfache Implementierung der Dreamweaver-Unterstützung für Flash-Inhalt. Wenn Sie mit dem Erstellen von Objekten und Befehlen sowie mit anspruchsvollen Formularen vertraut sind, können Sie Flash-Inhalt in Dreamweaver-Erweiterungen einfügen, um die Anwendung für die Benutzer dynamischer zu gestalten. Weitere Informationen zum Erstellen einer `commandButtons()`-Funktion, mit der Schaltflächen zum Anzeigen des Flash-Inhalts in das Dialogfeld eingefügt werden, finden Sie in „Befehle“ auf Seite 123.

Kapitel 6: Das Dreamweaver-Dokumentobjektmodell

Das Dokumentobjektmodell (DOM) in Adobe Dreamweaver CS3 ist für die Entwickler von Erweiterungen von sehr großer Bedeutung. Es dient dazu, auf Elemente im Dokument des Benutzers und in der Erweiterungsdatei zuzugreifen und sie zu bearbeiten.

Ein DOM definiert den Aufbau von Dokumenten, die mithilfe einer Markup-Sprache erstellt werden. Indem es Tags und Attribute als Objekte und Eigenschaften darstellt, ermöglicht das DOM Programmiersprachen die Bearbeitung von Dokumenten und deren Komponenten sowie den Zugriff darauf.

Die Struktur eines HTML-Dokuments entspricht der eines Baumes. Der Stamm ist das Tag `html`, und die beiden wichtigsten Äste sind die Tags `head` und `body`. Zu den Ablegern von `head` gehören die Tags `title`, `style`, `script`, `isindex`, `base`, `meta` und `link`. Ableger des Tags `body` sind Überschriften (`h1`, `h2` usw.), Elemente auf Blockebene (`p`, `div`, `form` usw.), Inline-Elemente (`br`, `img` usw.) und weitere Elementtypen. Von hier ausgehend verzweigt sich die Struktur weiter in Attribute wie `width`, `height`, `alt` und sonstige.

In einem DOM bleibt die Baumstruktur gewahrt und wird als Hierarchie von über- und untergeordneten Knoten dargestellt. Der Stammknoten hat keinen übergeordneten Knoten, und die Blattknoten haben keine untergeordneten Knoten. Auf jeder Ebene innerhalb der HTML-Struktur kann das HTML-Element von JavaScript als Knoten verwendet werden. Mithilfe dieser Struktur können Sie auf das Dokument oder jedes beliebige Element darin zugreifen.

In JavaScript können Sie über den Namen oder Index auf jedes Objekt im Dokument verweisen. Wenn etwa eine Schaltfläche zum Absenden mit dem Namen oder Kennzeichen `"myButton"` das zweite Element im ersten Formular des Dokuments ist, sind die beiden folgenden Verweisformen zulässig:

- Nach Name, wie in `document.myForm.myButton`
- Nach Index, wie in `document.forms[0].elements[1]`

Objekte mit demselben Namen, z. B. eine Gruppe von Optionsschaltern, werden zu einem Array reduziert. Sie können auf ein bestimmtes Objekt im Array zugreifen, indem Sie den Index (von Null ausgehend) erhöhen (z. B. wird der erste Optionsschalter mit dem Namen „myRadioGroup“ in einem Formular mit dem Namen „myForm“ als `document.myForm.myRadioGroup[0]` referenziert).

Welches Dokument-DOM?

Es ist wichtig, das DOM des Benutzerdokuments und das DOM der Erweiterung zu unterscheiden. Die in diesem Abschnitt angegebenen Informationen gelten für beide Dreamweaver-Dokumenttypen; die DOMs werden jedoch unterschiedlich referenziert.

Wenn Sie mit der Verwendung von JavaScript in Browsern vertraut sind, können Sie Objekte im aktiven Dokument durch die Eingabe von `document` referenzieren (z. B. `document.forms[0]`). In Dreamweaver bezieht sich `document` auf die Erweiterungsdatei und `document.forms[0]` auf das erste Formular der Benutzeroberfläche der Erweiterung. Um Objekte im Dokument des Benutzers zu referenzieren, müssen Sie jedoch `dw.getDocumentDOM()`, `dw.createDocument()` oder eine andere Funktion aufrufen, die ein Benutzerdokumentobjekt zurückgibt.

Um beispielsweise die erste Abbildung im aktiven Dokument zu referenzieren, können Sie `dw.getDocumentDOM().images[0]` verwenden. Sie können auch das Dokumentobjekt in einer Variablen speichern und diese Variablen für zukünftige Verweise verwenden, wie im folgenden Beispiel zu erkennen:

```
var dom = dw.getDocumentDOM(); //ruft das DOM des aktuellen Dokuments ab
var firstImg = dom.images[0];
firstImg.src = "myImages.gif";
```

Diese Schreibweise wird in allen Dateien des Konfigurationsordners verwendet, vor allem in Befehlsdateien. Weitere Informationen zur Methode `dw.getDocumentDOM()` finden Sie in der Beschreibung der Funktion `dreamweaver.getDocumentDOM()` im *Dreamweaver API-Referenzhandbuch*.

Das Dreamweaver-DOM

Das Dreamweaver-DOM enthält eine Teilmenge von Objekten, Eigenschaften und Methoden der DOM-Level-1-Spezifikation des World Wide Web Consortiums (W3C), ergänzt um bestimmte Eigenschaften des DOM von Microsoft Internet Explorer 4.0.

Objekte, Eigenschaften und Methoden des Dreamweaver-DOM

In der folgenden Tabelle sind die Objekte, Eigenschaften, Methoden und Ereignisse aufgeführt, die das Dreamweaver-DOM unterstützt. Manche Eigenschaften sind schreibgeschützt, wenn auf sie als Eigenschaft eines bestimmten Objekts zugegriffen wird. Eigenschaften, die bei der Verwendung im aufgeführten Kontext schreibgeschützt sind, sind mit einem Punkt (•) gekennzeichnet.

Objekt	Eigenschaften	Methoden	Ereignisse
window	- navigator • - document • - innerWidth • - innerHeight • - screenX • - screenY •	- alert() - confirm() - escape() - unescape() - close() - setTimeout() - clearTimeout() - setInterval() - clearInterval() - resizeTo()	onResize
navigator	platform •	Kein	Keine
document	- forms • (Array von Formularobjekten) - images • (Array von Bildobjekten) - layers • (ein Array von LAYER, ILAYER und absolut positionierten Elementen) - child-Objekte nach Name • - nodeType • - parentNode • - childNodes • - previousSibling • - nextSibling • - documentElement • - body • - URL • - parentWindow •	- getElementsByTagName() - getElementsByName() - getElementById() - hasChildNodes()	onLoad
Alle Tags/Elemente	- nodeType • - parentNode • - childNodes • - tagName • - previousSibling • - nextSibling • - attributes by name - innerHTML - outerHTML	- getAttribute() - setAttribute() - removeAttribute() - getElementsByTagName() - getElementsByName() - hasChildNodes()	

Objekt	Eigenschaften	Methoden	Ereignisse
form	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: tags:elements • (ein Array von Objekten des Typs button, checkbox, password, radio, reset, select, submit, text, file, hidden, image und textarea) mmcolorbutton Untergeordnete Objekte nach Name •	Nur die für alle Tags verfügbaren Methoden.	Keine
layer	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: visibility left top width height zIndex	Nur die für alle Tags verfügbaren Methoden.	Kein
image	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: src	Nur die für alle Tags verfügbaren Methoden.	onMouseOver onMouseOut onMouseDown onMouseUp
button reset submit	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: form •	Zusätzlich zu den für alle Tags verfügbaren Methoden: blur() focus()	onClick
checkbox radio	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: checked form •	Zusätzlich zu den für alle Tags verfügbaren Methoden: blur() focus()	onClick
password text file hidden image (Feld) textarea	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: form • value	Zusätzlich zu den für alle Tags verfügbaren Methoden: blur() focus() select()	onBlur onFocus
select	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: form • options • (Array an option-Objekten) selectedIndex	Zusätzlich zu den für alle Tags verfügbaren Methoden: blur() (nur für Windows) focus() (nur für Windows)	onBlur (Nur für Windows) onChange onFocus (Nur für Windows)
option	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: text	Nur die für alle Tags verfügbaren Methoden.	Kein
mmcolorbutton	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: name value	Kein	onChange
array boolean date function math number object string regexp	Entspricht Netscape Navigator 4.0	Entspricht Netscape Navigator 4.0	Kein

Objekt	Eigenschaften	Methoden	Ereignisse
text	nodeType • parentNode • childNodes • previousSibling • nextSibling • data	hasChildNodes ()	Kein
comment	nodeType • parentNode • childNodes • previousSibling • nextSibling • data	hasChildNodes ()	Kein
NodeList	length •	item ()	Kein
NamedNodeMap	length •	item ()	Kein

Eigenschaften und Methoden des Dokumentobjekts

In der folgenden Tabelle sind Einzelheiten zu den in Dreamweaver unterstützten Eigenschaften und Methoden des Objekts `document` aufgeführt. Ein Punkt (•) kennzeichnet schreibgeschützte Eigenschaften.

Eigenschaft oder Methode	Rückgabewert
nodeType •	Node.DOCUMENT_NODE
parentNode •	null
parentWindow •	Das JavaScript-Objekt, das dem übergeordneten Fenster des Dokuments entspricht. (Diese Eigenschaft ist im Microsoft Internet Explorer 4.0-DOM definiert, gehört aber nicht zu DOM Level 1 oder 2.)
childNodes •	Eine Knotenliste (NodeList) mit allen unmittelbar untergeordneten Elementen des Objekts <code>document</code> . In der Regel besitzt ein Dokument ein einziges untergeordnetes Element: das Objekt <code>HTML</code> .
previousSibling •	null
nextSibling •	null
documentElement •	Das JavaScript-Objekt, das dem Tag <code>HTML</code> entspricht. Mit dieser Eigenschaft wird der Wert von <code>document.childNodes</code> umgehend abgefragt und das <code>HTML</code> -Tag aus der <code>NodeList</code> extrahiert.
body •	Das JavaScript-Objekt, das dem Tag <code>body</code> entspricht. Mit dieser Eigenschaft wird der Wert von <code>document.documentElement.childNodes</code> umgehend abgefragt und das Tag <code>body</code> aus der <code>NodeList</code> extrahiert. Bei Frameset-Dokumenten gibt diese Eigenschaft den Knoten für das äußerste Frameset zurück.
URL •	Die Pfadangabe für das Dokument in der Form <code>Datei://URL</code> bzw., falls das Dokument nicht gespeichert wurde, ein leerer String.
getElementsByTagName (tagName)	Eine <code>NodeList</code> zum schrittweisen Durchlaufen von Tags des Typs <code>tagName</code> (z. B. <code>img</code>, <code>div</code> usw.). Wenn der Wert des Arguments <code>tagName</code> „LAYER“ ist, gibt die Funktion alle Tags der Typen <code>LAYER</code> und <code>ILAYER</code> sowie alle absolut positionierten Elemente zurück. Wenn der Wert des Arguments <code>tagName</code> „INPUT“ ist, gibt die Funktion alle Formularelemente zurück. (Wenn ein <code>name</code>-Attribut für ein oder mehrere <code>tagName</code>-Objekte angegeben ist, muss es – wie in der HTML 4.01-Spezifikation angegeben – mit einem Buchstaben beginnen, ist die Länge des von dieser Funktion zurückgegebenen Arrays ungültig.)

Eigenschaft oder Methode	Rückgabewert
<code>getElementsById(id)</code>	Ruft den Elementknoten mit der angegebenen <i>id</i> ab. Dabei bezeichnet <i>id</i> einen String mit der ID des abzurufenden Elements. <pre>var dom = dw.getDocumentDOM(); var contObj = dom.getElementById('content'); alert("The element with the id 'content' is a " + contObj.tagName);</pre>
<code>getElementsByName(attrName)</code>	Eine <i>NodeList</i> zum schrittweisen Durchlaufen von Elementen mit einem <i>attrName</i> -Attribut (z. B. alle Elemente mit dem Attribut „for“). Gehört nicht zu DOM Level 1 oder 2.
<code>getElementById(id)</code>	Ruft das HTML-Element mit der angegebenen ID ab.
<code>hasChildNodes()</code>	true

Eigenschaften und Methoden von HTML-Objekten

In der folgenden Tabelle sind die Eigenschaften und Methoden von HTML-Elementen in Dreamweaver aufgeführt, zusammen mit ihren Rückgabewerten und ggf. entsprechenden Erläuterungen. Ein Punkt (•) kennzeichnet schreibgeschützte Eigenschaften.

Eigenschaft oder Methode	Rückgabewert
<code>nodeType</code> •	<code>Node.ELEMENT_NODE</code>
<code>parentNode</code> •	Das übergeordnete Tag. Wenn es sich dabei um das HTML-Tag handelt, wird das Dokumentobjekt zurückgegeben.
<code>childNodes</code> •	Eine <i>NodeList</i> mit allen unmittelbar untergeordneten Elementen des Tags.
<code>previousSibling</code> •	Der Parallelknoten unmittelbar vor diesem. In einem HTML-Dokument ist der <code>previousSibling</code> für das <code>body</code> -Element beispielsweise das Element <code>head</code> .
<code>nextSibling</code> •	Der Parallelknoten unmittelbar nach diesem. In einem HTML-Dokument ist der <code>nextSibling</code> für das <code>head</code> -Element beispielsweise das Element <code>body</code> . (Alle <code>script</code> -, <code>style</code> - und <code>meta</code> -Tags im <code>head</code> -Bereich sind untergeordnete Knoten des <code>head</code> -Elements.)
<code>tagName</code> •	Der HTML- <code>tagName</code> für das Element, z. B. <code>IMG</code> , <code>A</code> oder <code>DIV</code> . Dieser Wert wird immer in Großbuchstaben zurückgegeben.
<code>attrName</code>	String, der den Wert des angegebenen Tag-Attributs enthält. <code>tag.attrName</code> kann nicht verwendet werden, wenn das Attribut <code>attrName</code> in der JavaScript-Sprache ein reserviertes Wort darstellt (zum Beispiel <code>class</code>). Verwenden Sie in diesem Fall <code>getAttribute()</code> und <code>setAttribute()</code> .
<code>innerHTML</code>	Der Quellcode innerhalb des öffnenden und des schließenden Tags. Der Code <code><p>Hello, World!</p></code> , <code>p.innerHTML</code> gibt beispielsweise <code>Hello, World!</code> zurück. Wenn Sie in diese Eigenschaft schreiben, wird der DOM-Baum sofort aktualisiert, um die neue Dokumentstruktur wiederzugeben. (Diese Eigenschaft ist im Microsoft Internet Explorer 4.0-DOM definiert, gehört aber nicht zu DOM Level 1 oder 2.)
<code>outerHTML</code>	Der Quellcode für dieses Tag, einschließlich des Tags selbst. Im vorherigen Beispielscode wird von <code>p.outerHTML</code> der String <code><p>Hello, World!</p></code> zurückgegeben. Wenn Sie in diese Eigenschaft schreiben, wird der DOM-Baum sofort aktualisiert, um die neue Dokumentstruktur wiederzugeben. (Diese Eigenschaft ist im Microsoft Internet Explorer 4.0-DOM definiert, gehört aber nicht zu DOM Level 1 oder 2.)
<code>getAttribute(attrName)</code>	Wert des angegebenen Attributs, sofern ausdrücklich angegeben; andernfalls null.
<code>getTranslatedAttribute(attrName)</code>	Der übersetzte Wert des angegebenen Attributs bzw. der gleiche Wert, der von <code>getAttribute()</code> zurückgegeben wird, wenn der Wert des Attributs nicht übersetzt ist. (Diese Eigenschaft ist nicht Teil von DOM Level 1. Sie wurde in Dreamweaver 3 aufgenommen, um die Übersetzung von Attributen zu unterstützen.)

Eigenschaft oder Methode	Rückgabewert
<code>setAttribute(attrName, attrValue)</code>	Gibt keinen Wert zurück. Setzt das betreffende Attribut auf das angegebene Wertbeispiel, also <code>img.setAttribute("src", "image/roses.gif")</code> .
<code>removeAttribute(attrName)</code>	Gibt keinen Wert zurück. Entfernt das angegebene Attribut und seinen Wert aus dem HTML-Code für dieses Tag.
<code>getElementsByTagName(tagName)</code>	Eine <code>NodeList</code> zum schrittweisen Durchlaufen von untergeordneten Tags des Typs <code>tagName</code> (z. B. <code>IMG</code> , <code>DIV</code> usw.). Wenn der Wert des Arguments <code>tagName</code> „layer“ ist, gibt die Funktion alle Tags der Typen <code>LAYER</code> und <code>ILAYER</code> sowie alle absolut positionierten Elemente zurück. Wenn der Wert des Arguments <code>tagName</code> „input“ ist, gibt die Funktion alle Formularelemente zurück. (Wenn ein <code>name</code> -Attribut für ein oder mehrere <code>tagName</code> -Objekte angegeben ist, muss es – wie in der HTML 4.01-Spezifikation angegeben – mit einem Buchstaben beginnen, ist die Länge des von dieser Funktion zurückgegebenen Arrays ungültig.)
<code>getElementsByName(attrName)</code>	Eine <code>NodeList</code> zum schrittweisen Durchlaufen von Elementen mit einem <code>attrName</code> -Attribut (z. B. alle Elemente mit dem Attribut „for“). Gehört nicht zu DOM Level 1 oder 2.
<code>hasChildNodes()</code>	Ein Boolescher Wert, der angibt, ob das Tag untergeordnete Elemente besitzt.
<code>hasTranslatedAttributes()</code>	Ein Boolescher Wert, der angibt, ob das Tag übersetzte Attribute besitzt. (Diese Eigenschaft ist nicht Teil von DOM Level 1. Sie wurde in Dreamweaver 3 aufgenommen, um die Übersetzung von Attributen zu unterstützen.)

Eigenschaften und Methoden von Textobjekten

Jeder zusammenhängende Textblock in einem HTML-Dokument (zum Beispiel der Text innerhalb eines `p`-Tags) wird durch ein JavaScript-Objekt repräsentiert. Textobjekte können keine untergeordneten Elemente haben. In der folgenden Tabelle sind Einzelheiten zu den vom DOM Level 1 übernommenen und in Dreamweaver verwendeten Eigenschaften und Methoden der Textobjekte aufgeführt. Ein Punkt (•) kennzeichnet schreibgeschützte Eigenschaften.

Eigenschaft oder Methode	Rückgabewert
<code>nodeType</code> •	<code>Node.TEXT_NODE</code>
<code>parentNode</code> •	Das übergeordnete Tag.
<code>childNodes</code> •	Eine leere <code>NodeList</code>
<code>previousSibling</code> •	Der Parallelknoten unmittelbar vor diesem. Im Code <code><p>blah
blah</p></code> , beispielsweise besitzt das Tag <code><p></code> drei untergeordnete Knoten (Textknoten, Elementknoten, Textknoten). Der <code>previousSibling</code> des dritten untergeordneten Elements ist das Tag <code>
</code> und der <code>previousSibling</code> des ersten untergeordneten Elements ist <code>null</code> .
<code>nextSibling</code> •	Der Parallelknoten unmittelbar nach diesem. Im Code <code><p>blah
blah</p></code> ist beispielsweise der <code>nextSibling</code> des ersten untergeordneten Elements des Tags <code>p</code> das <code>
</code> -Tag und der <code>nextSibling</code> des dritten untergeordneten Elements ist <code>null</code> .
<code>data</code>	Der eigentliche Textstring. Entities im Text werden als einzelne Zeichen dargestellt (der Text <code>Johann & ich</code> wird zum Beispiel als <code>Johann & ich</code> zurückgegeben).
<code>hasChildNodes()</code>	<code>false</code>

Eigenschaften und Methoden von Kommentarobjekten

Ein JavaScript-Objekt, dass für jeden HTML-Kommentar steht. In der folgenden Tabelle sind Einzelheiten zu den vom DOM Level 1 übernommenen und in Dreamweaver verwendeten Eigenschaften und Methoden der Kommentarobjekte aufgeführt. Ein Punkt (•) kennzeichnet schreibgeschützte Eigenschaften.

Eigenschaft oder Methode	Rückgabewert
nodeType •	Node.COMMENT_NODE
parentNode •	Das übergeordnete Tag.
childNodes •	Ein leeres NodeList-Array mit untergeordneten Knoten.
previousSibling •	Der Parallelknoten unmittelbar vor diesem.
nextSibling •	Der Parallelknoten unmittelbar nach diesem.
data	Der Textstring zwischen den Kommentarmarkierungen (<!-- und -->)
hasChildNodes()	false

Die Objekte „dreamweaver“ und „site“

Dreamweaver implementiert die über das DOM verfügbaren Standardobjekte und fügt zwei benutzerdefinierte Objekte hinzu: `dreamweaver` und `site`. Diese benutzerdefinierten Objekte werden in den APIs und beim Schreiben von Erweiterungen häufig verwendet. Weitere Informationen zu den Methoden der Objekte `Dreamweaver` und `Site` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Eigenschaften des Objekts `dreamweaver`

Das Objekt `Dreamweaver` hat, wie in der folgenden Liste beschrieben, zwei schreibgeschützte Eigenschaften:

- Die Eigenschaft `appName` hat den Wert "Dreamweaver".
- Der Wert der Eigenschaft `appVersion` wird in der Form "Versionsnummer.Releasenummer.Buildnummer [Sprachcode] (Plattform)" ausgegeben.

Der Wert der Eigenschaft `appVersion` für die schwedische Windows-Version von Dreamweaver CS3 lautet beispielsweise "8.0.XXXX [se] (Win32)" und der Wert für die englische Macintosh-Version lautet "8.0.XXXX [en] (MacPPC)".

Hinweis: Die Versions- und Buildnummer wird Ihnen unter *Hilfe > Über Dreamweaver* angezeigt.

Die Eigenschaften `appName` und `appVersion` wurden in Dreamweaver 3 implementiert und sind in früheren Versionen des Programms nicht verfügbar.

Um die Version von Dreamweaver zu ermitteln, überprüfen Sie zunächst die Existenz von `appVersion` und dann wie folgt die Versionsnummer:

```
if (dreamweaver.appVersion && dreamweaver.appVersion.indexOf('3.01') != -1) {~
    // Ausführungscode
}
```

Das Objekt `dreamweaver` hat eine Eigenschaft namens `systemScript`, mit der Sie die Sprache des verwendeten Betriebssystems ermitteln können. Verwenden Sie diese Eigenschaft wie folgt, um Sonderfälle in Ihrem Erweiterungscode für lokalisierte Betriebssysteme berücksichtigen zu können:

```
if (dreamweaver.systemScript && (dreamweaver.systemScript.indexOf('ja') != -1) {~
    SpecialCase
}
```

Die Eigenschaft `systemScript` gibt für lokalisierte Betriebssysteme die folgenden Werte zurück:

Programmiersprache	Wert
Japanisch	ja
Koreanisch	ko
Traditionelles Chinesisch	zh_tw
Vereinfachtes Chinesisch	zh_cn

Die Betriebssysteme aller europäischen Sprachen geben 'en' zurück.

Das Objekt „site“

Das Objekt `site` hat keine Eigenschaften. Informationen über die Methoden des Objekts `site` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Kapitel 7: Objekte der Einfügleiste

In Adobe Dreamweaver CS3 können mithilfe von Objekten bestimmte Strings in ein Benutzerdokument eingefügt werden. Objekte befinden sich in der Regel in der Einfügleiste, im Menü **Einfügen** oder auch in beiden. Mithilfe von Objekten können die Benutzer Inhalte wie Bilder, absolut positionierte Elemente (AP-Elemente) und Tabellen hinzufügen, indem Sie auf Symbole oder Menüoptionen klicken. Sie können Elemente in die Einfügleiste aufnehmen, um häufig wiederkehrende Aufgaben für Ihre Benutzer zu automatisieren. Zudem können Sie Dialogfelder erstellen, in denen die Benutzer bestimmte Attribute angeben können.



Objekte werden im Ordner **Configuration\Objects** gespeichert, der sich wiederum im Dreamweaver-Anwendungsordner befindet. Die Unterordner für die Objekte sind entsprechend ihrer Anordnung auf der Einfügleiste zusammengefasst. Sie können diese Dateien öffnen, um den Aufbau bereits vorhandener Objekte nachzuvollziehen. Sie können z. B. die Datei **Configuration\Objects\Common\Hyperlink.htm** öffnen, um den Code zu überprüfen, mit dem auf der Einfügleiste eine Schaltfläche für ein Hypertextverknüpfungsobjekt dargestellt wird.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Objekten:

Pfad	Datei	Beschreibung
Configuration\Objects\Objektyp\	Objektnamen.htm	Gibt an, was in das Dokument eingefügt werden soll.
Configuration\Objects\Objektyp\	Objektnamen.js	Enthält die auszuführenden Funktionen.
Configuration\Objects\Objektyp\	Objektnamen.gif	Enthält das Bild, das in der Einfügleiste angezeigt wird.
Configuration\Objects	insertbar.xml	Gibt die in der Einfügleiste angezeigten Objekte und deren Reihenfolge an.

So funktionieren Objektdateien

Objekte umfassen die folgenden Komponenten:

- Die HTML-Datei, die angibt, was in ein Dokument eingefügt wird.

Der Abschnitt `head` einer Objektdatei enthält JavaScript-Funktionen (oder verweist auf externe JavaScript-Dateien), die Formulareingaben aus dem Abschnitt `body` verarbeiten und steuern, welcher Inhalt dem Dokument eines Benutzers hinzugefügt wird. Der Abschnitt `body` einer Objektdatei kann ein HTML-Formular enthalten, das Parameter für das Objekt akzeptiert (z. B. die Anzahl der in eine Tabelle einzufügenden Zeilen und Spalten) und ein Dialogfeld aktiviert, in dem die Benutzer Attribute eingeben können.

Hinweis: Die einfachsten Objekte enthalten lediglich den einzufügenden HTML-Code, ohne die Tags `body` und `head`. Weitere Informationen finden Sie (in englischer Sprache) im Adobe Support Center im Abschnitt „Dreamweaver anpassen“.

- Das auf der Einfügleiste angezeigte Bild im Format 18 x 18 Pixel.
- Neue Passagen in der Datei **insertbar.xml**. In der Datei **insertbar.xml** ist festgelegt, an welcher Stelle der Einfügleiste das Objekt angezeigt wird.

Wenn ein Benutzer auf ein Symbol in der Einfügleiste klickt oder ein Element im Menü **Einfügen** aufruft, um ein Objekt auszuwählen, läuft folgende Ereigniskette ab:

1 Dreamweaver ruft die Funktion `canInsertObject()` auf, um festzustellen, ob ein Dialogfeld angezeigt werden soll.

Die Objektdaten werden nach dem Tag `form` durchsucht. Wenn ein Formular vorhanden ist und im Dialogfeld **Voreinstellungen** die Option **Beim Einfügen von Objekten Dialogfeld anzeigen** ausgewählt ist, ruft Dreamweaver die Funktion `windowDimensions()` auf (sofern definiert), um die Größe des Dialogfelds für das Formular festzulegen. Wenn in der Objektdaten kein Formular existiert, ruft Dreamweaver kein Dialogfeld auf. Schritt 2 wird in diesem Fall übersprungen.

2 Wenn Dreamweaver in Schritt 1 ein Dialogfeld angezeigt hat, gibt der Benutzer die Parameter für das Objekt in das Dialogfeld ein (zum Beispiel die Anzahl der Zeilen und Spalten in einer Tabelle) und klickt auf **OK**.

3 Dreamweaver ruft die Funktion `objectTag()` auf und fügt ihren Rückgabewert nach der aktuellen Auswahl in das Dokument ein. (Die aktuelle Auswahl wird dabei nicht ersetzt.)

4 Wenn Dreamweaver die Funktion `objectTag()` nicht findet, wird stattdessen die Funktion `insertObject()` gesucht und aufgerufen.

Die Definitionsdatei für die Einfügleiste

In der Datei **Configuration\Objects\insertbar.xml** werden die Eigenschaften der Einfügleiste festgelegt. Diese XML-Datei enthält Definitionen für jedes einzelne Objekt, und zwar in der Reihenfolge, in der die Objekte angezeigt werden.

Wenn der Benutzer Dreamweaver zum ersten Mal startet, wird die Einfügleiste waagrecht über dem Dokument angezeigt. Anschließend werden Anzeige und Position in der Registrierung gespeichert.

Die Tag-Hierarchie der Datei insertbar.xml

Das folgende Beispiel illustriert das Format und die Hierarchie verschachtelter Tags in der Datei `insertbar.xml`:

```
<?xml version="1.0" ?>
<!DOCTYPE insertbarset SYSTEM "-//Macromedia//DWEExtension insertbar 5.0">

<insertbar xmlns:MMString="http://www.Macromedia.com/schemes/data/string/">

<category id="DW_Insertbar_Common" MMString:name="insertbar/categorycommon" folder="Common">
    <button id="DW_Hyperlink" image="Common\Hyperlink.png"
        MMString:name="insertbar/hyperlink" file="Common\Hyperlink.htm" />
    <button id="DW_Email" image="Common\E-Mail Link.png"
        MMString:name="insertbar/email" file="Common\E-Mail Link.htm" />
    <separator />

    <menubutton id="DW_Images" MMString:name="insertbar/images"
        image="Common\Image.png">
        <button id="DW_Image" image="Common\Image.png"
            MMString:name="insertbar/image" file="Common\Image.htm" />
        ...
    </menubutton>
    <separator />
    <button id="DW_TagChooser" MMString:name="insertbar/tagChooser"
        image="Common\Tag Chooser.gif" command="dw.showTagChooser()"
        codeOnly="TRUE"/>
</category>
...
</insertbar>
```

Hinweis: Während für die Tags `insertbar` und `category` die jeweiligen schließenden Tags `</insertbar>` und `</category>` verwendet werden müssen, gibt es für die Tags `button`, `checkboxbutton` und `separator` keine entsprechenden schließenden Tags. Stattdessen wird bei den Tags `button`, `checkboxbutton` und `separator` das Ende ihrer Attribute und ihres Inhalts mit einem Schrägstrich (/) vor der schließenden Klammer gekennzeichnet.

Tags zur Definition der Einfügeleiste

Die Datei **insertbar.xml** enthält die folgenden Tags und Attribute:

<insertbar>

Beschreibung

Dieses Tag signalisiert den Anfang des Inhalts der Definitionsdatei für die Einfügeleiste. Das schließende `</insertbar>`-Tag gibt das Ende des Inhalts an.

Attribute

Keine

Beispiel

```
<insertbar>
  <category id="DW_Insertbar_Common" folder="Common">
    <button id="DW_Hyperlink" image="Common\Hyperlink.gif"
      file="Common\Hyperlink.htm" />0
  ...
</insertbar>
```

<category>

Beschreibung

Dieses Tag definiert eine Kategorie auf der Einfügeleiste (wie etwa **Allgemein**, **Formulare** oder **HTML**). Das Ende des Inhalts der Kategorie wird mit dem schließenden `</insertbar>`-Tag angegeben.

***Hinweis:** Die Einfügeleiste wird standardmäßig in Verwendungskategorien untergliedert (z. B. **Allgemein**, **Formulare** oder **HTML**). In früheren Versionen von Dreamweaver war diese Leiste auf ähnliche Weise durch Registerkarten aufgeteilt. Die Benutzer können selbst festlegen, wie die Objekte der Einfügeleiste angeordnet sein sollen (in Kategorien oder Registerkarten). Wenn der Benutzer sich für Registerkarten entscheidet, werden die einzelnen Registerkarten mithilfe des `category`-Tags definiert.*

Attribute

`id`, `{folder}`, `{showIf}`

Beispiel

```
<category id="DW_Insertbar_Common" folder="Common">
  <button id="DW_Hyperlink" image="Common\Hyperlink.gif"
    file="Common\Hyperlink.htm" />
</category>
```

<menubutton>

Beschreibung

Dieses Tag definiert ein Popupmenü für die Einfügeleiste.

Attribute

`id`, `image`, `{showIf}`, `{name}`, `{folder}`

Beispiel

```
<menubutton
  id="DW_ImageMenu"
  name="Images"
  image="Common\imagemenu.gif"
  folder="Images">

  <button id="DW_Image"
    image="Common\Image.gif"
    enabled=" "
    showIf=" "
    file="Common\Image.htm" />
</menubutton>
```

<button />

Beschreibung

Dieses Tag definiert eine Schaltfläche auf der Einfügleiste, mit deren Hilfe der Benutzer den durch die Attribute `command` oder `file` angegebenen Code ausführen kann.

Attribute

`id`, `image`, `name`, `{canDrag}`, `{showIf}`, `{enabled}`, `{command}`, `{file}`, `{tag}`, `{codeOnly}`

Beispiel

```
<button id="DW_Object"
image="Common\Object.gif"
name="Object"
enabled="true"
showIf=" "
file="Common\Obect.htm"
/>
```

<checkbox />

Beschreibung

Bei einem Kontrollkästchen handelt es sich um eine Schaltfläche, die aktiviert oder deaktiviert sein kann. Wenn ein Benutzer auf sie klickt, wird es in gedrückter Form dargestellt und ist hervorgehoben. Wenn das Kontrollkästchen deaktiviert ist, wird es in flacher Form dargestellt. In Dreamweaver sind die folgenden Zustände möglich: Mauszeiger über dem Kontrollkästchen, gedrücktes Kontrollkästchen, Mauszeiger über dem gedrückten Kontrollkästchen und deaktiviertes gedrücktes Kontrollkästchen. Der Befehl muss gewährleisten, dass sich der Status des Kontrollkästchens ändert, wenn darauf geklickt wird.

Attribute

`id`, `image`, `checked`, `{showIf}`, `{enabled}`, `{command}`, `{file}`, `{tag}`, `{name}`, `{codeOnly}`

Beispiel

```
<checkbox id="DW_StandardView"
name = "Standard View"
image="Tools\Standard View.gif"
checked="_View_Standard"
command="dw.getDocumentDOM().setShowLayoutView(false)"/>
```

<separator />

Beschreibung

Dieses Tag zeigt in der Einfügleiste eine vertikale Linie an.

Attribute

`{showIf}`

Beispiel

```
<separator showIf="_VIEW_CODE"/>
```

Attribute für die Tags zur Definition der Einfügeleiste

Die Attribute für die Tags zur Definition der Einfügeleiste haben die folgenden Bedeutungen:

id="unique id"

Beschreibung

Das Attribut `id` ist ein Bezeichner für die Schaltflächen, die auf der Einfügeleiste angezeigt werden. Das Attribut `id` muss in der Datei ein eindeutiger Bezeichner für das Element sein.

Beispiel

```
id="DW_Anchor"
```

image="Bilddateipfad"

Beschreibung

Dieses Attribut kennzeichnet den Pfad zur Datei des Symbols, das in der Einfügeleiste angezeigt wird, relativ zum Dreamweaver-Ordner **Configuration**. Das Symbol kann jedes Format haben, das Dreamweaver anzeigen kann. In der Regel werden jedoch die Formate GIF oder JPEG und die Maße 18 x 18 Pixel verwendet.

Beispiel

```
image="Common/table.gif"
```

canDrag="Boolean"

Beschreibung

Dieses Attribut gibt an, ob der Benutzer das Symbol in den Code oder den Arbeitsbereich ziehen kann, um das Objekt in ein Dokument einzufügen. Bei fehlendem Argument gilt als Standardwert `true`.

Beispiel

```
canDrag="false"
```

showIf="Enabler"

Beschreibung

Dieses Attribut gibt an, dass diese Schaltfläche nur auf der Einfügeleiste angezeigt werden soll, wenn der vorgegebene Dreamweaver-Enabler den Wert `true` aufweist. Falls Sie `showIf` nicht angeben, wird die Schaltfläche immer eingeblendet. Mögliche Enabler: `_SERVERMODEL_ASP`, `_SERVERMODEL_ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von ColdFusion), `_SERVERMODEL_CFML_UD4` (für UltraDev Version 4 von ColdFusion), `_SERVERMODEL_PHP`, `_FILE_TEMPLATE`, `_VIEW_CODE`, `_VIEW_DESIGN`, `_VIEW_LAYOUT`, `_VIEW_EXPANDED_TABLES`, und `_VIEW_STANDARD`.

Sie können mehrere Enabler angeben, indem Sie sie durch ein Komma (gleichbedeutend mit AND) trennen. Zur Angabe von NOT geben Sie ein Ausrufezeichen (!) ein.

Beispiel

Wenn eine Schaltfläche nur in der Codeansicht für eine ASP-Seite angezeigt werden soll, geben Sie die Enabler wie folgt an:

```
showIf="_VIEW_CODE, _SERVERMODEL_ASP"
```

enabled="Enabler"**Beschreibung**

Dieses Attribut gibt an, dass das Element für den Benutzer verfügbar sein soll, wenn der Wert von *DW_enabler* *true* lautet. Wenn Sie die Funktion *enabled* nicht angeben, ist das Element standardmäßig immer aktiviert. Mögliche Enabler:

_SERVERMODEL_ASP, *_SERVERMODEL_ASPNET*, *_SERVERMODEL_JSP*, *_SERVERMODEL_CFML* (für alle Versionen von ColdFusion), *_SERVERMODEL_CFML_UD4* (für UltraDev Version 4 von ColdFusion), *_SERVERMODEL_PHP*, *_FILE_TEMPLATE*, *_VIEW_CODE*, *_VIEW_DESIGN*, *_VIEW_LAYOUT*, *_VIEW_EXPANDED_TABLES*, und *_VIEW_STANDARD*.

Sie können mehrere Enabler angeben, indem Sie sie durch ein Komma (gleichbedeutend mit AND) trennen. Zur Angabe von NOT geben Sie ein Ausrufezeichen (!) ein.

Beispiel

Wenn die Schaltfläche nur in der Codeansicht verfügbar sein soll, geben Sie Folgendes an:

```
enabled="_VIEW_CODE"
```

Damit wird die Schaltfläche in anderen Ansichten abgeblendet.

checked="enabler"**Beschreibung**

Das *checked*-Attribut ist erforderlich, wenn Sie das *checkboxbutton*-Tag verwenden.

Das Element ist markiert, wenn *DW_enabler* auf *true* gesetzt ist. Mögliche Enabler: *_SERVERMODEL_ASP*, *_SERVERMODEL_ASPNET*, *_SERVERMODEL_JSP*, *_SERVERMODEL_CFML* (für alle Versionen von ColdFusion), *_SERVERMODEL_CFML_UD4* (nur für UltraDev Version 4 von ColdFusion), *_SERVERMODEL_PHP*, *_FILE_TEMPLATE*, *_VIEW_CODE*, *_VIEW_DESIGN*, *_VIEW_LAYOUT*, *_VIEW_EXPANDED_TABLES*, und *_VIEW_STANDARD*.

Sie können mehrere Enabler angeben, indem Sie sie durch ein Komma (gleichbedeutend mit AND) trennen. Zur Angabe von NOT geben Sie ein Ausrufezeichen (!) ein.

Beispiel

```
checked="_View_Layout"
```

command="API-Funktion"**Beschreibung**

Anstatt Dreamweaver auf eine HTML-Datei mit dem einzufügenden Code zu verweisen, können Sie mithilfe dieses Tags einen Befehl angeben, den Dreamweaver ausführt, sobald der Benutzer auf die Schaltfläche klickt.

Beispiel

```
command="dw.showTagChooser()"
```

file="file_path"**Beschreibung**

Das Attribut *file* kennzeichnet den Pfad einer Objektdatei (relativ zum Dreamweaver-Ordner **Configuration**). Dreamweaver übernimmt die QuickInfo für das Objektsymbol vom Titel der Objektdatei, sofern Sie nicht das Attribut *name* angeben.

Beispiel

```
file="Templates/Editable.htm"
```

tag="Editor"

Beschreibung

Mit diesem Attribut wird Dreamweaver aufgefordert, einen Tag-Editor aufzurufen. Wenn in der Codeansicht das Attribut `tag` definiert ist und der Benutzer auf das Objekt klickt, ruft Dreamweaver das Tag-Dialogfeld auf. Wenn Sie in der Codeansicht die Attribute `tag` und `command` angeben, ruft Dreamweaver den Tag-Editor auf. Wenn in der Entwurfsansicht `codeOnly="TRUE"` angegeben ist und das Attribut `file` nicht festgelegt ist, ruft Dreamweaver MX die Code- und Entwurfsansicht auf, legt den Fokus auf den Code und ruft den Tag-Editor auf.

Beispiel

```
tag = "form"
```

name="QuickInfo-Text"

Beschreibung

Das Attribut `name` gibt die QuickInfo an, die angezeigt wird, wenn sich der Mauszeiger über dem Objekt befindet. Wenn Sie eine Objektdatenbank angeben, aber das Attribut `name` nicht festlegen, verwendet Dreamweaver den Namen der Objektdatenbank für die QuickInfo.

Hinweis: Wenn das `name`-Attribut nicht angegeben ist, kann das Objekt nicht in der Favoritenkategorie auf der Einfügleiste gruppiert werden.

Manche Objekte der Einfügleiste verwenden eine Variante des `name`-Attributs mit dem Präfix `MMString`. `MMString` kennzeichnet einen lokalisierten String. Eine Erläuterung dieser Werte finden Sie unter „Erweiterungen lokalisieren“ auf Seite 73.

Beispiel

```
name = "cfoutput"
```

Einfügleiste ändern

Sie können Objekte aus einer Kategorie in eine andere verschieben, Kategorien umbenennen und Objekte vollständig aus dem Bedienfeld entfernen. Damit die Änderungen in der Einfügleiste angezeigt werden, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

So verschieben oder kopieren Sie ein Objekt von einer Kategorie der Einfügleiste in eine andere Kategorie oder innerhalb einer Kategorie an eine andere Stelle:

- 1 Speichern Sie eine Sicherungskopie der Datei `insertbar.xml` (z. B. unter dem Namen `insertbar.backup.xml`).
- 2 Öffnen Sie die Originaldatei `insertbar.xml`.
- 3 Suchen Sie das Tag `button`, das für das zu verschiebende oder zu kopierende Objekt steht. Wenn Sie z. B. das Bildobjekt von seiner ursprünglichen Stelle in der Kategorie **Allgemein** verschieben möchten, müssen Sie das Tag `button` suchen, dessen Attribut `id` den Wert `"DW_Image"` besitzt.
- 4 Kopieren Sie das gesamte Tag `button`, oder schneiden Sie es aus.
- 5 Suchen Sie das Tag `category`, das für die Kategorie steht, in die Sie das Objekt verschieben oder kopieren möchten.
- 6 Gehen Sie an die Stelle in der Kategorie, wo das Objekt angezeigt werden soll.
- 7 Fügen Sie das kopierte Tag `button` ein.
- 8 Speichern Sie die Datei `insertbar.xml`.
- 9 Laden Sie die Erweiterungen neu.

So entfernen Sie ein Objekt aus der Einfügleiste:

- 1 Speichern Sie eine Sicherungskopie der Datei **insertbar.xml** (z. B. unter dem Namen **insertbar.backup.xml**).
- 2 Öffnen Sie die Originaldatei **insertbar.xml**.
- 3 Suchen Sie das Tag **button**, das für das zu verschiebende oder zu kopierende Objekt steht.
- 4 Löschen Sie das gesamte Tag **button**.
- 5 Speichern Sie die Datei **insertbar.xml**.
- 6 Verschieben Sie auf Ihrer Festplatte die HTML-, GIF- und JavaScript-Dateien des Objekts aus ihrem aktuellen Ordner in einen Ordner, der nicht in der Datei **insertbar.xml** aufgeführt ist. Sie können z. B. unter **Configuration\Objects** einen neuen Ordner namens **Unused** erstellen und die Dateien des Objekts dorthin verschieben. (Wenn Sie sicher sind, dass Sie das Objekt löschen möchten, können Sie diese Dateien vollständig löschen. Es ist aber stets empfehlenswert, Sicherungskopien dieser Dateien zu behalten, falls Sie das Objekt später wiederherstellen müssen.)
- 7 Laden Sie die Erweiterungen neu.

So ändern Sie die Reihenfolge der Kategorien in der Einfügleiste:

- 1 Speichern Sie eine Sicherungskopie der Datei **insertbar.xml** (z. B. unter dem Namen **insertbar.backup.xml**).
- 2 Öffnen Sie die Originaldatei **insertbar.xml**.
- 3 Suchen Sie das Tag **category**, das der zu verschiebenden Kategorie entspricht, und wählen Sie dieses Tag einschließlich aller darin enthaltenen Tags aus.
- 4 Schneiden Sie das Tag aus.
- 5 Fügen Sie das Tag an der neuen Stelle ein. Achten Sie darauf, das Tag nicht innerhalb eines anderen **category**-Tags einzufügen.
- 6 Speichern Sie die Datei **insertbar.xml**.
- 7 Laden Sie die Erweiterungen neu.

So erstellen Sie eine neue Kategorie:

- 1 Speichern Sie eine Sicherungskopie der Datei **insertbar.xml** (z. B. unter dem Namen **insertbar.backup.xml**).
- 2 Öffnen Sie die Originaldatei **insertbar.xml**.
- 3 Erstellen Sie ein neues **category**-Tag, und geben Sie dabei den Standardordner für die Kategorie und eine Reihe von Objekten an, die in der Kategorie angezeigt werden sollen.
- 4 Weitere Informationen über die Syntax der Tags in der Datei **insertbar.xml** finden Sie unter „Tags zur Definition der Einfügleiste“ auf Seite 97.
- 5 Speichern Sie die Datei **insertbar.xml**.
- 6 Laden Sie die Erweiterungen neu.

Objekte in die Einfügleiste aufnehmen

Sie können der Einfügleiste Objekte hinzufügen. Damit die Änderungen in der Einfügleiste angezeigt werden, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

Gehen Sie wie folgt vor, um der Einfügleiste ein neues Objekt hinzuzufügen:

- 1 Definieren Sie die entsprechende Codezeichenfolge für das Dokument des Benutzers mithilfe von HTML und nach Belieben von JavaScript.
- 2 Geben Sie ein Bild für die Schaltfläche in der Benutzeroberfläche von Dreamweaver an, oder erstellen Sie ein neues Bild (18 x 18 Pixel).

Wenn Sie ein größeres Bild erstellen, wird es automatisch auf 18 x 18 Pixel verkleinert. Falls Sie keine Grafik für Ihr Objekt erstellen, wird auf der Einfügleiste ein Standardsymbol für das Objekt angezeigt. Dabei handelt es sich um ein Fragezeichen (?).

- 3 Fügen Sie die neuen Dateien dem Ordner **Configuration\Objects** hinzu.
- 4 Geben Sie in der Datei **insertbar.xml** den Speicherort dieser neuen Dateien an, und legen Sie Attribute für das Erscheinungsbild der Schaltfläche fest (siehe „Die Definitionsdatei für die Einfügleiste“ auf Seite 96).
- 5 Starten Sie Dreamweaver neu, oder laden Sie die Erweiterungen neu.

Das neue Objekt erscheint an der angegebenen Position auf der Einfügleiste.

Hinweis: Auch wenn Objektdateien in eigenen Ordnern gespeichert werden können, dürfen die Dateinamen jeweils nur einmal vorkommen. Die Funktion `dom.insertObject()` sucht nach Dateien im gesamten Ordner **Objects**, ohne nach Unterordnern zu unterscheiden. (Weitere Informationen über die Funktion `dom.insertObject()` finden Sie im Dreamweaver API-Referenzhandbuch.) Wenn sich sowohl im Ordner **Forms** als auch im Ordner **MyObjects** eine Datei mit dem Namen **Button.htm** befindet, können diese beiden Dateien nicht unterschieden werden. Falls zwei verschiedene Instanzen von **Button.htm** vorhanden sind, zeigt `dom.insertObject()` zwei Objekte namens „Button“ an, ohne dass der Benutzer einen Unterschied erkennt.

Objekte zum Menü „Einfügen“ hinzufügen

Um dem Menü **Einfügen** (oder einem anderen Menü) ein Objekt hinzuzufügen oder die Position eines Objekts festzulegen, bearbeiten Sie die Datei **menus.xml**. Diese Datei steuert die gesamte Menüstruktur von Dreamweaver. Weitere Informationen über das Bearbeiten der Datei **menus.xml** finden Sie in „Menüs und Menübefehle“ auf Seite 133.

Wenn Sie die Erweiterung an andere Dreamweaver-Benutzer weitergeben möchten, finden Sie unter „Mit dem Extension Manager arbeiten“ auf Seite 74 weitere Informationen über das Packen von Erweiterungen.

Einfaches Beispiel für das Einfügen eines Objekts

In diesem Beispiel wird zur Einfügleiste ein Objekt hinzugefügt, mit dem die Benutzer anhand einer Schaltfläche einen ausgewählten Text durchgestrichen formatieren können. Dieses Objekt ist z. B. hilfreich, wenn ein Benutzer ein fremdes Dokument redigiert.

Da in diesem Beispiel ein Text bearbeitet wird, bietet es sich an, einige der Objekte des Popupmenüs **Text** in der Kategorie **HTML** der Einfügleiste als Modelle zu betrachten. In den Objektdateien für die Formatierungen **Fett**, **Hervorhebung** und **Überschrift** finden Sie eine vergleichbare Funktionsweise: Dreamweaver umschließt einen ausgewählten Text mit Tags.

Zum Erstellen des Objekts für das Einfügen der durchgestrichenen Textformatierung nehmen Sie folgende Schritte vor:

- „Die HTML-Datei erstellen“ auf Seite 104
- „Die JavaScript-Funktionen hinzufügen“ auf Seite 104
- „Das Bild erstellen“ auf Seite 105
- „Die Datei insertbar.xml bearbeiten“ auf Seite 106
- „Dialogfelder hinzufügen“ auf Seite 106
- „Popupmenüs für die Einfügleiste erstellen“ auf Seite 108

Die HTML-Datei erstellen

Der Titel des Objekts wird zwischen dem öffnenden und dem schließenden `title`-Tag angegeben. Legen Sie außerdem JavaScript als Skriptsprache fest.

So erstellen Sie die HTML-Datei:

- 1 Erstellen Sie eine neue leere Datei.
- 2 Fügen Sie den folgenden Code hinzu:

```
<html>
<head>
<title>Strikethrough</title>
<script language="javascript">
</script>
</head>
<body>
</body>
</html>
```

- 3 Speichern Sie die Datei unter dem Namen **Strikethrough.htm** im Ordner **Configuration\Objects\Text**.

Die JavaScript-Funktionen hinzufügen

In diesem Beispiel definieren die JavaScript-Funktionen das Verhalten des Objekts „Strikethrough“ und den einzufügenden Code. Sie müssen alle API-Funktionen in den head-Abschnitt der Datei einfügen. Die vorhandenen Objektdateien, wie z. B. **Configuration\Objects\Text\Em.htm**, bestehen aus einer einfachen Abfolge von Funktionen und Kommentaren.

Die erste von der Objektdefinitionsdatei verwendete Funktion ist `isDOMRequired()`. Sie gibt an, ob die Entwurfsansicht mit der vorhandenen Codeansicht synchronisiert sein muss, bevor die Ausführung fortgesetzt wird. Da dieses Objekt jedoch zusammen mit zahlreichen anderen Objekten in der Codeansicht verwendet werden kann, ist keine erzwungene Synchronisierung erforderlich.

So fügen Sie die `isDOMRequired()`-Funktion hinzu:

- 1 Fügen Sie in den head-Abschnitt der Datei **Strikethrough.htm** die folgende Funktion zwischen das öffnende und das schließende `script`-Tag ein:

```
<script language="javascript">
    function isDOMRequired() {
        // Gibt false zurück, d.h. das Objekt ist in der Codeansicht verfügbar.
        return false;
    }
</script>
```

- 2 Speichern Sie die Datei.

Legen Sie nun fest, ob für die nächste Funktion `objectTag()` oder `insertObject()` verwendet werden soll. Das Objekt „Strikethrough“ umschließt lediglich den ausgewählten Text mit dem Tag `s` und erfüllt deshalb nicht die Kriterien zur Verwendung der Funktion `insertObject()` (siehe „insertObject()“ auf Seite 113).

Innerhalb der Funktion `objectTag()` können Sie `dw.getFocus()` verwenden, um festzustellen, ob es sich bei der aktuellen Ansicht um die Codeansicht handelt. Befindet sich der Eingabe-Fokus auf der Codeansicht, so sollte die Funktion den ausgewählten Text mit dem entsprechenden Tag (Groß- oder Kleinbuchstaben) umschließen. Liegt der Eingabe-Fokus auf der Entwurfsansicht, dann kann der ausgewählte Text mithilfe von `dom.applyCharacterMarkup()` formatiert werden. Beachten Sie, dass diese Funktion nur für unterstützte Tags möglich ist (siehe hierzu `dom.applyCharacterMarkup()` im *Dreamweaver API-Referenzhandbuch*). Für andere Tags oder Operationen sind möglicherweise andere API-Funktionen erforderlich. Nachdem Dreamweaver die Formatierung angewendet hat, sollte das Programm die Einfügemarke (Cursor) ohne weitere Meldung oder Eingabeaufforderung wieder in das Dokument setzen. Im folgenden Verfahren sehen Sie den neuen Aufbau der Funktion `objectTag()`.

So fügen Sie die objectTag()-Funktion hinzu:

1 Fügen Sie in den head-Abschnitt der Datei **Strikethrough.htm** die folgende Funktion hinter die Funktion `isDOMRequired()` ein:

```
function objectTag() {
    // Es wird bestimmt, ob der Benutzer die Codeansicht aktiviert hat.
    var dom = dw.getDocumentDOM();
    if (dw.getFocus() == 'textView' || dw.getFocus(true) == 'html'){
        var upCaseTag = (dw.getPreferenceString("Source Format", "Tags Upper Case", "") ==
            'TRUE');
        // die Auswahl manuell in Tags einschließen
        if (upCaseTag){
            dom.source.wrapSelection('<S>', '</S>');
        }else{
            dom.source.wrapSelection('<s>', '</s>');
        }
    }
    // Hat der Benutzer die Codeansicht nicht aktiviert, wird die Formatierung in der Entwurfsansicht
    angewendet.
    }else if (dw.getFocus() == 'document'){
        dom.applyCharacterMarkup("s");
    }
    // Einfach zurückkehren--sonst nichts machen
    return;
}
```

2 Speichern Sie die Datei unter dem Namen **Strikethrough.htm** im Ordner **Configuration\Objects\Text**.

Als Alternative zum Einfügen der JavaScript-Funktionen in den head-Abschnitt der HTML-Datei können Sie eine separate JavaScript-Datei erstellen. Diese getrennte Organisation ist bei Objekten nützlich, die mehrere Funktionen enthalten, oder bei Funktionen, die auch von anderen Objekten verwendet werden können.

So trennen Sie die HTML-Objektdefinitionsdatei von den unterstützenden JavaScript-Funktionen:

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie per Kopieren und Einfügen alle JavaScript-Funktionen in die Datei ein.

3 Löschen Sie die Funktionen aus der Datei **Strikethrough.htm**, und fügen Sie dem Attribut `src` des script-Tags den Namen der JavaScript-Datei hinzu, wie im folgenden Beispiel dargestellt:

```
<html>
<head>
<title>Strikethrough</title>
<script language="javascript" src="Strikethrough.js">
</script>
</head>
<body>
</body>
</html>
```

4 Speichern Sie die Datei **Strikethrough.htm**.

5 Speichern Sie die Datei, die nun die JavaScript-Funktionen enthält, unter dem Namen **Strikethrough.js** im Ordner **Configuration\Objects\Text**.

Das Bild erstellen**So erstellen Sie das Bild für die Einfügeleiste:**

1 Erstellen Sie eine GIF-Grafik (18 x 18 Pixel), wie in der folgenden Abbildung gezeigt:



2 Speichern Sie die Datei unter dem Namen **Strikethrough.gif** im Ordner **Configuration\Objects\Text**.

Die Datei insertbar.xml bearbeiten

Als Nächstes müssen Sie die Datei **insertbar.xml** bearbeiten, damit Dreamweaver diese beiden Elemente mit der Oberfläche für die Einfügleiste assoziieren kann.

Hinweis: Bevor Sie die Datei **insertbar.xml** bearbeiten, sollten Sie eine Sicherungskopie mit dem Namen **insertbar.xml.bak** erstellen.

Der Code in der Datei **insertbar.xml** kennzeichnet sämtliche Objekte, die auf der Einfügleiste vorhanden sind.

- Jedes Tag `category` in der XML-Datei erstellt eine Kategorie auf der Oberfläche.
- Jedes Tag `menubutton` erstellt ein Popupmenü auf der Einfügleiste.
- Jedes Tag `button` in der XML-Datei fügt ein Symbol auf der Einfügleiste ein und verknüpft dieses mit der entsprechenden HTML-Datei oder -funktion.

So fügen Sie der Einfügleiste das neue Objekt hinzu:

1 Suchen Sie im Anfangsbereich der Datei **insertbar.xml** die folgende Zeile:

```
<category id="DW_Insertbar_Common" MMString:name="insertbar/categorycommon" folder="Common">
```

Diese Zeile gibt den Anfang der Kategorie **Allgemein** in der Einfügleiste an.

2 Beginnen Sie nach dem `category`-Tag mit einer neuen Zeile. Fügen Sie dann das Tag `button` ein, und weisen Sie ihm die Attribute `id`, `image` und `file` für das Objekt „Strikethrough“ zu.

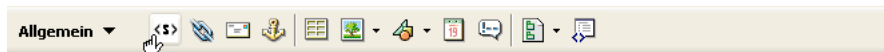
Bei der Kennung muss es sich um einen eindeutigen Namen für die Schaltfläche handeln (verwenden Sie entsprechend den Konventionen zur Namensgebung für dieses Objekt die Bezeichnung `DW_Text_Strikethrough`). Die Attribute `image` und `file` geben Dreamweaver, wie hier gezeigt, lediglich den Speicherort der unterstützenden Dateien an:

```
<button id="DW_Text_Strikethrough"
image="Text\Strikethrough.gif"
file="Text\Strikethrough.htm"/>
```

3 Speichern Sie die Datei **insertbar.xml**.

4 Laden Sie die Erweiterungen neu (siehe „Erweiterungen neu laden“ auf Seite 71).

Das neue Objekt wird, wie in der folgenden Abbildung dargestellt, am Anfang der Kategorie **Allgemein** auf der Einfügleiste angezeigt:



Dialogfelder hinzufügen

Sie können Ihrem Objekt ein Formular hinzufügen, in das der Benutzer Parameter eingeben kann, bevor Dreamweaver den angegebenen Code einfügt (z. B. benötigt das Hyperlink-Objekt vom Benutzer die Werte für Text, Verknüpfung, Kategorie, Index, Titel und Zugriffstaste). In diesem Beispiel fügen Sie dem Objekt „Strikethrough“ aus dem vorhergehenden Beispiel ein Formular hinzu. Das Formular öffnet ein Dialogfeld, in dem der Benutzer angeben kann, ob die Textfarbe in Rot geändert und das Tag **Strikethrough** hinzugefügt werden soll.

Bei diesem Beispiel wird vorausgesetzt, dass Sie bereits eine getrennte JavaScript-Datei namens **Strikethrough.js** erstellt haben.

Fügen Sie zunächst der Datei **Strikethrough.js** die Funktion hinzu, die vom Formular aufgerufen wird, sobald der Benutzer die Textfarbe ändert. Diese Funktion gleicht der Funktion `objectTag()` für das Objekt „Strikethrough“, ist jedoch optional.

So erstellen Sie die Funktion:

1 Erstellen Sie nach der Funktion `objectTag()` in **Strikethrough.js** eine Funktion namens `fontColorRed()`, indem Sie den folgenden Code eingeben:

```
function fontColorRed(){
    var dom = dw.getDocumentDOM();
    if (dw.getFocus() == 'textView' || dw.getFocus(true) == 'html'){
        var upCaseTag = (dw.getPreferenceString("Source Format", "Tags Upper Case", "")
            == 'TRUE');

        // die Auswahl manuell in Tags einschließen
        if (upCaseTag){
            dom.source.wrapSelection('<FONT COLOR="#FF0000">', '</FONT>');
        }else{
            dom.source.wrapSelection('<font color="#FF0000">', '</font>');
        }
    }else if (dw.getFocus() == 'document'){
        dom.applyFontMarkup("color", "#FF0000");
    }
    // Einfach zurückkehren--sonst nichts machen
    return;
}
```

Hinweis: Da `dom.applyCharacterMarkup()` keine Änderungen der Schriftfarbe unterstützt, müssen Sie die entsprechende API-Funktion zum Ändern der Schriftfarbe ermitteln. (Weitere Informationen finden Sie unter `dom.applyFontMarkup()` im Dreamweaver API-Referenzhandbuch).

2 Speichern Sie die Datei unter dem Namen **Strikethrough.js**.

Fügen Sie dann der Datei **Strikethrough.htm** das Formular hinzu. Beim Formular dieses Beispiels handelt es sich um ein einfaches Kontrollkästchen, das die Funktion `fontColorRed()` aufruft, sobald der Benutzer daraufklickt. Definieren Sie das Formular mit dem Tag `form`. Mit dem `table`-Tag können Sie das Layout festlegen (anderenfalls kann es vorkommen, dass im Dialogfeld Wörter umbrochen werden oder das Dialogfeld eine ungewöhnliche Größe hat).

So fügen Sie das Formular hinzu:

1 Fügen Sie nach dem Tag `body` den folgenden Code ein:

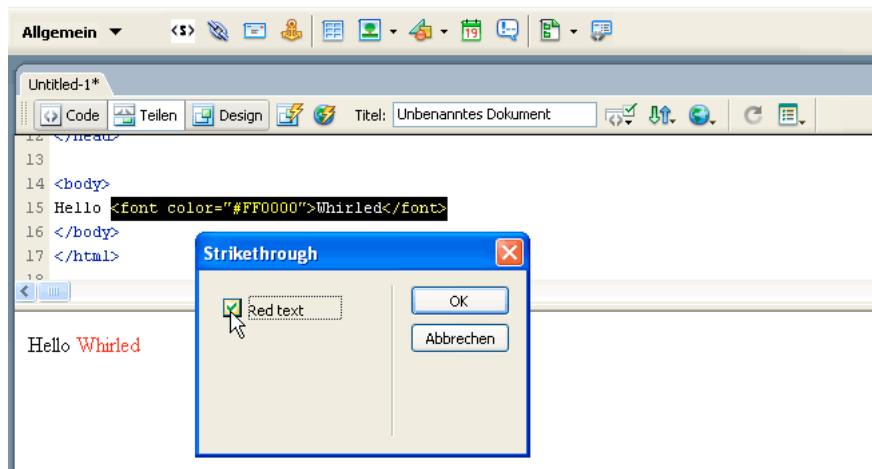
```
<form>
<table border="0" height="100" width="100">
    <tr valign="baseline">
        <td align="left" nowrap>
            <input type="checkbox" name="red" onClick=fontColorRed()>Red text</input>
        </td>
    </tr>
</table>
</form>
```

2 Speichern Sie die Datei unter dem Namen **Strikethrough.htm**.

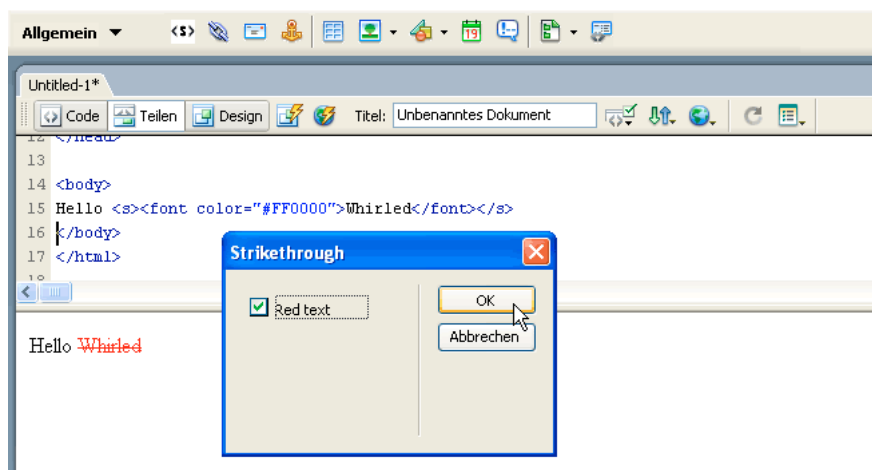
3 Laden Sie die Erweiterungen neu (siehe „Erweiterungen neu laden“ auf Seite 71).

So testen Sie das Dialogfeld:

- 1 Wählen Sie die Option **Red Text** aus.

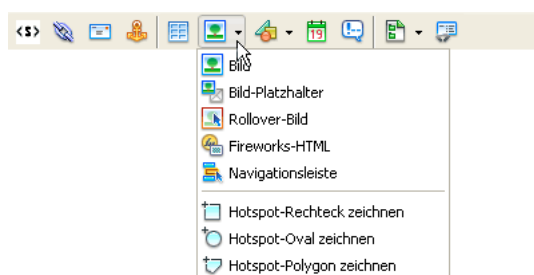


- 2 Klicken Sie auf **OK**, um die Funktion `objectTag()` auszuführen, die den Text durchstreicht:



Popupmenüs für die Einfügeleiste erstellen

In der Einfügeleiste von Dreamweaver sind Objekte jetzt neu angeordnet. Außerdem werden Popupmenüs unterstützt, mit denen sich Objekte in kleineren Gruppen anordnen lassen, wie in der folgenden Abbildung gezeigt:



Im folgenden Beispiel wird in der Einfügeleiste eine neue Kategorie namens **Editorial** erstellt, der dann ein Popupmenü hinzugefügt wird. Dieses Popupmenü enthält bereits das erstellte Objekt „Strikethrough“. Nun erstellen Sie ein Objekt namens „Blue Text“ und gruppieren die beiden Objekte im Popupmenü. Mit den Objekten in der Kategorie **Editorial** können die Benutzer ein Dokument redigieren. Text, der gelöscht werden soll, wird durchgestrichen formatiert, neuer Text wird blau gekennzeichnet.

So organisieren Sie die Dateien:

- 1 Erstellen Sie im Installationsordner von Dreamweaver einen neuen Ordner namens **Configuration\Objects\Editorial**.
- 2 Kopieren Sie die Dateien für das von Ihnen erstellte Beispielobjekt „Strikethrough“ (**Strikethrough.htm**, **Strikethrough.js** und **Strikethrough.gif**) in den Ordner **Editorial**.

So erstellen Sie das Objekt „Blue Text“:

- 1 Erstellen Sie eine neue HTML-Datei.
- 2 Fügen Sie den folgenden Code hinzu:

```
<html>
<head>
<title>Blue Text</title>
<script language="javascript">

//----- API-FUNKTIONEN-----

function isDOMRequired() {
    // Gibt false zurück, d.h. das Objekt ist in der Codeansicht verfügbar.
    return false;
}

function objectTag() {
    // die Auswahl manuell in Tags einschließen
    var dom = dw.getDocumentDOM();
    if (dw.getFocus() == 'textView' || dw.getFocus(true) == 'html'){

        var upCaseTag = (dw.getPreferenceString("Source Format", "Tags Upper Case", "") ==
            'TRUE');

        // die Auswahl manuell in Tags einschließen
        if (upCaseTag){
            dom.source.wrapSelection('<FONT COLOR="#0000FF">', '</FONT>');
        }else{
            dom.source.wrapSelection('<font color="#0000FF">', '</font>');
        }
    }else if (dw.getFocus() == 'document'){
        dom.applyFontMarkup("color", "#0000FF");
    }

    // Einfach zurückkehren--sonst nichts machen
    return;
}

</script>
</head>
<body>
</body>
</html>
```

- 3 Speichern Sie die Datei unter dem Namen **AddBlue.htm** im Ordner **Editorial**.

Nun können Sie eine Grafik für das „Blue Text“-Objekt erstellen.

So erstellen Sie die Grafik:

- 1 Erstellen Sie eine GIF-Datei im Format 18 x 18 Pixel mit der folgenden Grafik:



- 2 Speichern Sie die Grafik unter dem Namen **AddBlue.gif** im Ordner **Editorial**.

Als Nächstes bearbeiten Sie die Datei **insertbar.xml**. Diese Datei definiert die Objekte auf der Einfügleiste und deren Positionen. Innerhalb der Tags `category` sind verschiedene `menubutton`-Tags samt Attributen vorhanden; diese `menubutton`-Tags definieren die einzelnen Popupmenüs der HTML-Kategorie. Innerhalb der `menubutton`-Tags definiert jedes `button`-Tag ein Element im Popupmenü.

So bearbeiten Sie die Datei insertbar.xml:

- 1 Suchen Sie im Anfangsbereich der Datei die folgende Codezeile:

```
<insertbar xmlns:MMString="http://www.Macromedia.com/schemes/data/string/">
```

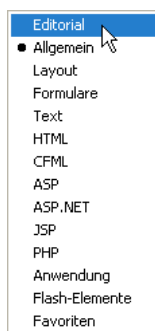
Das Tag `insertbar` definiert den Anfang des Bereichs für die Einfügleiste.

- 2 Fügen Sie im Anschluss an diese Zeile ein neues Tag `category` für die zu erstellende Kategorie **Editorial** ein. Weisen Sie ihm eine eindeutige ID, einen Namen und Ordnerattribute zu, und fügen Sie dann wie im folgenden Beispiel ein schließendes `category`-Tag ein.

```
<category id="DW_Insertbar_Editorial" name="Editorial" folder="Editorial">
</category>
```

- 3 Laden Sie die Erweiterungen neu. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

Auf der Einfügleiste wird die Kategorie **Editorial** angezeigt:



- 4 Fügen Sie innerhalb der öffnenden und schließenden `category`-Tags das Popupmenü mithilfe des Tags `menubutton` und der folgenden Attribute ein, einschließlich einer eindeutigen ID.

```
<menubutton id="DW_Insertbar_Markup" name="markup" image="Editorial\Strikethrough.gif"
folder="Editorial">
```

Weitere Informationen zu Attributen finden Sie unter „Attribute für die Tags zur Definition der Einfügleiste“ auf Seite 99.

- 5 Fügen Sie mit dem Tag `button` wie folgt die Objekte für das neue Popupmenü ein.

```
<button id="DW_Editorial_Strikethrough" image="Editorial\Strikethrough.gif"
file="Editorial\Strikethrough.htm"/>
```

- 6 Fügen Sie nach dem Tag für die Schaltfläche des Strikethrough-Objekts wie folgt das Hypertext-Objekt ein:

```
<button id="DW_Blue_Text" image="Editorial\AddBlue.gif name="Blue Text" file="Editorial\AddBlue.htm"/>
```

Hinweis: Das `button`-Tag verfügt über kein eigenes schließendes Tag, sondern endet lediglich mit `</>`.

7 Beenden Sie das Popupmenü mit dem schließenden Tag `</menubutton>`.

Der folgende Code umfasst Ihre gesamte Kategorie samt dem Popupmenü und den beiden Objekten:

```
<category id="DW_Insertbar_Editorial" name="Editorial" folder="Editorial">
  <menubutton id="DW_Insertbar_Markup" name="markup"
    image="Editorial\Strikethrough.gif" folder="Editorial">
    <button id="DW_Editorial_Strikethrough"
      image="Editorial\Strikethrough.gif" file="Editorial\Strikethrough.htm"/>
    <button id="DW_Blue_Text" image="Editorial\AddBlue.gif" name="Blue Text"
      file="Editorial\AddBlue.htm"/>
    </menubutton>
  </category>
```

So testen Sie das neue Popupmenü:

1 Laden Sie die Erweiterungen neu. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

2 Klicken Sie auf das Menü **Editorial**.

Das folgende Popupmenü wird angezeigt:



Die Funktionen der Objekt-API

In diesem Abschnitt werden die Funktionen der API für Objekte beschrieben. Sie müssen entweder die Funktion `insertObject()` oder die Funktion `objectTag()` definieren. Weitere Informationen über diese Funktionen finden Sie unter „insertObject()“ auf Seite 113. Die übrigen Funktionen sind optional.

canInsertObject()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Diese Funktion legt fest, ob das Dialogfeld für das Objekt angezeigt wird.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert.

Beispiel

Mit dem folgenden Code wird festgelegt, dass Dreamweaver einen bestimmten String im Dokument sucht, bevor der Benutzer das ausgewählte Objekt einfügen kann:

```
function canInsertObject(){
    var docStr = dw.getDocumentDOM().documentElement.outerHTML;
    var patt = /hava/;
    var found = ( docStr.search(patt) != -1 );
    var insertionIsValid = true;

    if (!found){
        insertionIsValid = false;
        alert("the document must contain a 'hava' string to use this object.");
    }
    return insertionIsValid;}
```

displayHelp()

Beschreibung

Wenn diese Funktion definiert ist, wird im Dialogfeld **Parameter** unter den Schaltflächen **OK** und **Abbrechen** die Schaltfläche **Hilfe** angezeigt. Diese Funktion wird aufgerufen, sobald der Benutzer auf die Schaltfläche **Hilfe** klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Im folgenden Beispiel wird die Datei **myObjectHelp.htm** in einem Browser geöffnet. Darin wird die Verwendung der Datei erläutert:

```
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/myObjectHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

isDOMRequired()

Beschreibung

Diese Funktion ermittelt, ob das Objekt zur Funktion ein gültiges DOM benötigt. Wenn diese Funktion den Wert `true` zurückgibt oder nicht definiert ist, geht Dreamweaver davon aus, dass für den Befehl ein gültiges DOM erforderlich ist, und synchronisiert die Code- und Entwurfsansicht des Dokuments vor der Ausführung. Bei der Synchronisierung wird die Entwurfsansicht mit allen in der Codeansicht vorgenommenen Änderungen aktualisiert.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet `true`, falls für die Funktionsfähigkeit eines Befehls ein gültiges DOM erforderlich ist; anderenfalls `false`.

insertObject()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion ist erforderlich, wenn die Funktion `objectTag()` nicht definiert ist. Sie wird aufgerufen, wenn der Benutzer auf **OK** klickt. Entweder fügt Sie Code in das Dokument des Benutzers ein und schließt das Dialogfeld oder sie zeigt eine Fehlermeldung an und lässt das Dialogfeld geöffnet. Diese Funktion kann in Objekten alternativ zu `objectTag()` verwendet werden. Dabei wird nicht davon ausgegangen, dass der Benutzer an der aktuellen Einfügemarke Text einfügt. Die Überprüfung der Daten erfolgt, sobald der Benutzer auf **OK** klickt. Sie sollten die Funktion `insertObject()` verwenden, wenn eine der folgenden Bedingungen erfüllt ist:

- Sie müssen Code an mehreren Stellen einfügen.
- Sie müssen Code an einer anderen Stelle als der Einfügemarke einfügen.
- Sie müssen eine Eingabe vor dem Einfügen von Code überprüfen.

Wenn keine dieser Bedingungen erfüllt ist, verwenden Sie die Funktion `objectTag()`.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String mit einer Fehlermeldung oder einen leeren String. Im Fall eines leeren Strings wird das Dialogfeld für das Objekt geschlossen, sobald der Benutzer auf **OK** klickt. Wenn der String nicht leer ist, zeigt Dreamweaver die Fehlermeldung an und das Dialogfeld bleibt geöffnet.

Enabler

`canInsertObject()`

Beispiel

Im folgenden Beispiel wird die Funktion `insertObject()` verwendet, da vor dem Einfügen von Code die Eingabe überprüft werden muss:

```
function insertObject() {
    var theForm = document.forms[0];
    var nameVal = theForm.firstField.value;
    var passwordVal = theForm.secondField.value;
    var errMsg = "",
        isValid = true;

    // sicherstellen, dass Feldwerte vollständig und gültig sind
    if (nameVal == "" || passwordVal == "") {
        errMsg = "Complete all values or click Cancel."
    } else if (nameVal.length < 4 || passwordVal.length < 6) {
        errMsg = "Your name must be at least four characters, and your password at least six";
    }

    if (!errMsg) {
        // hier lässt sich das Dokument bearbeiten Eine Übung für den Leser.
    }
    return errMsg;
}
```

objectTag()

Beschreibung

Die Funktionen `objectTag()` und `insertObject()` schließen sich gegenseitig aus: Wenn in einem Dokument beide definiert sind, wird die Funktion `objectTag()` verwendet. Weitere Informationen finden Sie unter „insertObject()“ auf Seite 113.

Diese Funktion fügt in das Benutzerdokument einen Codestring ein. In Dreamweaver MX ist die Rückgabe eines leeren Strings oder von `null` (auch als „return;“ bezeichnet) eine Aufforderung an Dreamweaver, nichts zu tun.

Hinweis: Dabei wird davon ausgegangen, dass die Änderungen bereits manuell vor der Anweisung `return` erfolgt sind. Die Tatsache, dass nichts geschieht, ist in diesem Fall nicht gleichbedeutend mit einem Klicken auf **Abbrechen**.

Wenn sich in Dreamweaver 4 der Fokus auf der Codeansicht befindet und es sich bei der Auswahl um einen Bereich handelt (also nicht um eine Einfügemarke), wird dieser Bereich durch den von der Funktion `objectTag()` zurückgegebenen String ersetzt. Dabei handelt es sich um den Wert `true`, auch wenn die Funktion `objectTag()` einen leeren String oder nichts zurückgibt. Die Funktion `objectTag()` gibt einen leeren String oder `null` zurück, da das Dokument bereits von Hand bearbeitet wurde. Anderenfalls kann die Bearbeitung durch doppelte Anführungszeichen (") gelöscht und die Auswahl ersetzt werden.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet den String, der in das Dokument des Benutzers eingefügt werden soll.

Beispiel

Das folgende Beispiel der Funktion `objectTag()` fügt eine Kombination des Typs `OBJECT/EMBED` für ein spezielles ActiveX-Steuerelement und -Plug-In ein:

```
function objectTag() {
return '\n' +
'<OBJECT CLASSID="clsid:166F100B-3A9R-11FB-8075444553540000" \n'-
+ 'CODEBASE="http://www.mysite.com/product/cabs/-
myproduct.cab#version=1,0,0,0" \n' + 'NAME="MyProductName"> \n' -
+ '<PARAM NAME="SRC" VALUE=""> \n' + '<EMBED SRC="" HEIGHT="" -
WIDTH="" NAME="MyProductName"> \n' + '</OBJECT>'
}
```

windowDimensions()

Beschreibung

Diese Funktion legt die Abmessungen für das Dialogfeld **Optionen** fest. Wenn diese Funktion nicht definiert ist, werden die Abmessungen automatisch berechnet.

Hinweis: Verwenden Sie diese Funktion nur dann, wenn das Dialogfeld **Optionen** größer als 640 x 480 Pixel sein soll.

Argumente

platform

- Der Wert des Arguments *platform* ist entweder "macintosh" oder "windows", abhängig von der verwendeten Plattform.

Rückgabewerte

Dreamweaver erwartet einen String der Form "widthInPixels,heightInPixels".

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da die Schaltflächen für **OK** und **Abbrechen** nicht mit einbezogen sind. Wenn mit den zurückgegebenen Abmessungen nicht für alle Optionen Platz vorhanden ist, werden Bildlaufleisten angezeigt.

Beispiel

Das folgende Beispiel der Funktion `windowDimensions()` legt die Abmessungen des Dialogfelds **Parameter** auf 648 x 520 Pixel (Windows) bzw. 660 x 580 Pixel (Macintosh) fest:

```
function windowDimensions(platform) {  
  
    var retval = ""  
    if (platform == "windows") {  
        retval = "648, 520";  
    } else {  
        retval = "660, 580";  
    }  
    return retval;  
}
```


Kapitel 8: API für Probleme bei der Browserkompatibilitätsprüfung

In Adobe Dreamweaver CS3 unterstützt Sie die Browserkompatibilitätsprüfung (BCC f. engl. Browser Compatibility Check) beim Erstellen von Seitenlayouts, die in mehreren Browsern gut funktionieren (also gleich aussehen und gleich funktionieren). Dabei wird nach HTML/CSS-Kombinationen gesucht, die Fehler in der Browserwiedergabe auslösen können. Diese Funktion verwendet JavaScript, um das Dokument des Benutzers nach problematischen Kombinationen von HTML- und CSS-Code zu durchsuchen. Der JavaScript-Code wird in HTML-Dateien gespeichert, so genannten Problemerkennungsdateien. Damit diese korrekt funktionieren, müssen sie im Ordner **Configuration/BrowserProfiles/Issues/** gespeichert werden.

Funktionsweise der Ermittlung

Wenn ein Benutzer zum ersten Mal eine Browserkompatibilitätsprüfung ausführt (und jedes Mal, wenn er im Dialogfeld **Zielbrowser** auf **OK** klickt), läuft die folgende Ereigniskette ab:

- 1 Dreamweaver liest die Profile für die gewählten Browser aus dem Ordner **Configuration\BrowserProfiles** ein.
- 2 Dreamweaver ruft in jeder Datei im Ordner **Configuration\BrowserProfiles\Issues** die Funktion `getIssueID()` auf, um die eindeutige ID der einzelnen Probleme abzurufen.
- 3 Dreamweaver ruft für jedes Problem die Funktion `getAffectedBrowserDisplayNames()` auf, falls sie definiert worden ist.
- 4 Dreamweaver ruft für jedes Problem die Funktion `getAffectedBrowserProfiles()` auf, um festzustellen, ob sich das Problem auf einen oder mehrere der gewählten Browser auswirkt.
- 5 Dreamweaver ruft für jedes Problem die Funktion `getIssueName()` auf, um den Namen zu ermitteln, der im Bedienfeld **Ergebnisse** angezeigt wird, wenn das Problem festgestellt worden ist.
- 6 Dreamweaver ruft für jedes Problem die Funktion `getIssueDescription` auf, um den Text zu ermitteln, der rechts im Bedienfeld **Ergebnisse** und in der Codeansicht in der QuickInfo zum Problemsymbol angezeigt wird, wenn das Problem festgestellt worden ist.

Die folgende Ereigniskette läuft nach Schritt 6 in der obigen Anleitung und für alle danach ausgeführten Browserkompatibilitätsprüfungen ab, und zwar für jeden im Dialogfeld **BCC-Einstellungen** gewählten Browser:

Reihenfolge der Ereignisse

- 1 Dreamweaver analysiert die Stile, die für das aktuelle Dokument gelten, so wie sie der jeweilige Browser lesen würde. Dabei werden sowohl inline definierte Stile als auch im head-Bereich oder in einem externen Stylesheet definierte Stile geprüft.
- 2 Dreamweaver ruft für jede Problemdatei im betroffenen Browser die Funktion `findIssue()` auf.

Ein Beispiel für ein Problem

Die folgenden Beispiele sind in den Dateien **ColAndColgroupCapturedByCaption.htm** und **ColAndColgroupCapturedByCaption.js** im Ordner **Configuration/BrowserProfiles/Issues/** zu finden.

ColAndColgroupCapturedByCaption.htm

```
<!DOCTYPE HTML SYSTEM "-//
//DWExtension layout-engine 5.0//dialog">
<html>
<head>
<title>Col and Colgroup Captured by Caption</title>

<script src="../../Shared/Common/Scripts/dwscripts.js"></script>
<script src="issue_utils.js"></script>
<script src="ColAndColgroupCapturedByCaption.js"></script>
<script>
//----- LOCALISIERBARE GLOBALS-----
var ISSUE_NAME = "Col und Colgroup/Caption-Konflikt";
var ISSUE_DESC = "Wenn das caption-Tag gemäß HTML 4.01-Spezifikation direkt nach dem öffnenden table-Tag
eingefügt wird, werden alle Stile, die auf die Tags col und colgroup in derselben Tabelle angewandt wurden,
ignoriert.";

//----- ENDE LOCALISIERBARE GLOBALS-----
</script>
</head>

<body>
</body>
</html>
```

ColAndColgroupCapturedByCaption.js

```
function findIssue(){
    var DOM = dw.getDocumentDOM();
    var issueNodes = new Array();

    if (DOM){
        // erst feststellen, ob das Dokument caption-Tags enthält.
        var captions = DOM.getElementsByTagName('caption');

        // eine Reihe von Variablen deklarieren, die in der
        // nachfolgenden Schleife benötigt werden.
        var currCap = null, props = null, parentTable = null;
        var colgroups = null, cols = null, allcol = null;
        var property = "", definedStyles = new Array();

        // OK, jetzt alle etwaigen Captions in einer Schleife durcharbeiten.
        for (var i=0; i < captions.length; i++){
            currCap = captions[i];
            parentTable = currCap.parentNode;

            // Die Caption ist nur ein Problem, wenn sie sich am gültigen
            // Ort befindet (also das erste Element, das der Tabelle untergeordnet ist)
            if (currCap == parentTable.childNodes[0]){

                // nach allen colgroup- und col-Tags suchen, die sich in
                // derselben Tabelle wie die Caption befinden.
                colgroups = parentTable.getElementsByTagName('colgroup');
                cols = parentTable.getElementsByTagName('col');
                allcol = colgroups.concat(cols);

                for (var x=0; x < allcol.length; x++){
                    // sind für ein colgroup- oder col-Tag in dieser Tabelle Stile
                    // deklariert, haben wir einen Problemknoten. Sie
                    // brauchen nicht weiterzusuchen
```

```

        props = window.getDeclaredStyle(allcol[x]);
        property = "";
        definedStyles.length = 0;
        for (property in props) {
            definedStyles.push(property);
        }
        if (definedStyles.length > 0) {
            issueNodes.push(currCap);
            break;
        }
    }
}
}
return issueNodes;
}
function getAffectedBrowserDisplayNames() {
    return new Array("Safari 2.0");
}
function getAffectedBrowserProfiles() {
    return new Array("Safari 2.0");
}
function getIssueID() {
    return "COL_AND_COLGROUP_CAPTURED_BY_CAPTION";
}
function getIssueName() {
    return ISSUE_NAME;
}
function getIssueDescription() {
    return ISSUE_DESC;
}
function getConfidenceLevel() {
    //DETCN 4
    return issueUtils.CONFIDENCE_HIGH;
}

```

Die Funktionen der Problem-API

Alle Funktionen der Problem-API, außer `getAffectedBrowserDisplayNames()`, sind obligatorisch. Wie bei Erweiterungs-APIs müssen Sie den Hauptteil jeder Funktion selbst schreiben und den richtigen Wert an Dreamweaver übergeben. Informationen über Funktionen für die Browserkompatibilitätsprüfung finden Sie im Abschnitt „Seiteninhalt“ des *Dreamweaver API-Referenzhandbuchs*.

findIssue()

Verfügbarkeit

Dreamweaver CS3.

Beschreibung

Durchsucht das Dokument nach der CSS/HTML-Kombination, die ein bestimmtes Problem in der Browserwiedergabe auslöst.

Argumente

Keine

Rückgabewerte

Ein Array an Elementknoten, die das Problem belegen (oder repräsentieren). Dreamweaver wählt diese Knoten aus, wenn der Benutzer die verschiedenen Browserkompatibilitätsprobleme auswählt.

Beispiel

Die folgende `findIssue()`-Funktion gibt ein Array an `<button>`-Tags zurück, auf die `float: left` oder `float: right` angewendet worden ist:

```
function findIssue(){
    var DOM = dw.getDocumentDOM();
    var issueNodes = new Array();
    var buttons = DOM.getElementsByTagName('button');
    var props = null;
    for (var i=0; i < buttons.length; i++){
        props = window.getComputedStyle(buttons[i]);
        if (props.cssFloat == "left" || props.cssFloat == "right"){
            issueNodes.push(buttons[i]);
        }
    }
    return issueNodes;
}
```

getAffectedBrowserProfiles()

Verfügbarkeit

Dreamweaver CS3.

Beschreibung

Versorgt Dreamweaver mit einer Liste an Browsern, für die das Problem relevant ist.

Argumente

Keine

Rückgabewerte

Ein Array an Browsernamen, von denen jeder mit der ersten Zeile in einem gültigen Browserprofil genau übereinstimmen muss (siehe die TXT-Dateien im Ordner **Configuration\BrowserProfiles**).

Beispiel

```
function getAffectedBrowsers(){
    return new Array("Microsoft Internet Explorer 5.0",
        "Microsoft Internet Explorer 5.5",
        "Microsoft Internet Explorer 6.0");
}
```

getAffectedBrowserDisplayNames()

Verfügbarkeit

Dreamweaver CS3.

Beschreibung

Versorgt Dreamweaver mit einer Liste an für den Benutzer sichtbaren Browsernamen, für die das Problem relevant ist. Diese Funktion ist optional. Wenn sie nicht angegeben wird, werden stattdessen die von `getAffectedBrowserProfiles()` zurückgegebenen Profilnamen verwendet.

Argumente

Keine

Rückgabewerte

Ein Array an Browsernamen. Dieses Array muss dem von `getAffectedBrowserProfiles()` zurückgegebenen Array entsprechen.

Beispiel

```
function getAffectedBrowsers(){  
    return new Array("IE/Win 5.0",  
        "IE/Win 5.5",  
        "IE/Win 6.0");  
}
```

getIssueID()**Verfügbarkeit**

Dreamweaver CS3.

Beschreibung

Versorgt Dreamweaver mit einer eindeutigen ID zu dem Problem.

Argumente

Keine

Rückgabewerte

Ein String mit einem eindeutigen Bezeichner zu dem Problem.

Beispiel

```
function getIssueID() {  
    return "EXPANDING_BOX_PROBLEM";  
}
```

getIssueName()**Verfügbarkeit**

Dreamweaver CS3.

Beschreibung

Versorgt Dreamweaver mit dem Namen oder einer kurzen Beschreibung des Problems.

Argumente

Keine

Rückgabewerte

Ein String mit einer ausführlichen Beschreibung des Problems.

Beispiel

```
function getIssueName(){  
    return "The Expanding Box Problem";  
}
```

getIssueDescription()**Verfügbarkeit**

Dreamweaver CS3.

Beschreibung

Versorgt Dreamweaver mit einer ausführlichen Beschreibung des Problems.

Argumente

Keine

Rückgabewerte

Ein String mit einer ausführlichen Beschreibung des Problems.

Beispiel

```
function getIssueDescription(){  
    return "Fixed-dimension boxes will incorrectly expand to fit their  
           content instead of clipping content at the specified width  
           or height.";  
}
```


Kapitel 9: Befehle

Mit den Befehlen in Adobe Dreamweaver CS3 können praktisch alle gewünschten Bearbeitungsvorgänge am aktuellen Dokument des Benutzers, an anderen geöffneten Dokumenten oder an beliebigen HTML-Dokumenten auf dem lokalen Laufwerk durchgeführt werden. Befehle dienen dazu, HTML-Tags und -Attribute sowie Kommentare und andere Texte einzufügen, zu entfernen oder anzupassen.

Befehle sind HTML-Dateien. Der body-Bereich einer Befehlsdatei kann ein HTML-Formular enthalten, in dem Optionen für den Befehl eingegeben werden können (beispielsweise wie oder nach welcher Spalte eine Tabelle sortiert werden soll). Der head-Bereich einer Befehlsdatei enthält JavaScript-Funktionen, die Formulareingaben aus dem body-Bereich verarbeiten und steuern, welche Änderungen am Benutzerdokument vorgenommen werden.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Befehlen:

Pfad	Datei	Beschreibung
Configuration\Commands\	<i>Befehlsname.htm</i>	Gibt die Benutzeroberfläche an.
Configuration\Commands\	<i>Befehlsname.js</i>	Enthält die auszuführenden Funktionen.

Funktionsweise von Befehlen

Wenn der Benutzer auf ein Menü klickt, das einen Befehl enthält, laufen folgende Vorgänge ab:

- 1 Dreamweaver ruft die Funktion `canAcceptCommand()` auf, um zu prüfen, ob das Menüelement deaktiviert werden sollte. Wenn `canAcceptCommand()` den Wert `false` zurückgibt, wird der Befehl im Menü abgeblendet und das Verfahren angehalten. Wenn `canAcceptCommand()` den Wert `true` zurückgibt, wird das Verfahren fortgesetzt.
- 2 Der Benutzer wählt einen Befehl im Menü aus.
- 3 Falls definiert, wird in der ausgewählten Befehlsdatei die Funktion `receiveArguments()` aufgerufen, damit der Befehl die vom Menüelement oder von der Funktion `dreamweaver.runCommand()` übergebenen Argumente verarbeiten kann. Weitere Informationen zur Funktion `dreamweaver.runCommand()` finden Sie im *Dreamweaver API-Referenzhandbuch*.
- 4 Falls definiert, wird die Funktion `commandButtons()` aufgerufen, um zu bestimmen, welche Schaltflächen auf der rechten Seite des Optionsdialogfelds angezeigt werden sollen und welcher Code beim Klicken auf die Schaltflächen jeweils ausgeführt werden soll.
- 5 Dreamweaver durchsucht die Befehlsdatei nach dem Tag `form`. Wenn ein Formular vorhanden ist, ruft Dreamweaver die Funktion `windowDimensions()` auf, mit der die Größe des Optionsdialogfelds geändert wird, in dem sich die body-Elemente der Datei befinden. Wenn Funktion `windowDimensions()` nicht definiert ist, passt Dreamweaver die Größe des Dialogfelds automatisch an.
- 6 Ist im `body`-Tag der Befehlsdatei eine Ereignisprozedur des Typs `onLoad` enthalten, wird diese von Dreamweaver ausgeführt (unabhängig davon, ob ein Dialogfeld angezeigt wird). Wenn kein Dialogfeld angezeigt wird, entfallen die restlichen Schritte.
- 7 Der Benutzer wählt die Optionen für den Befehl aus. An die einzelnen Felder angehängte Ereignisprozeduren werden von Dreamweaver ausgeführt, sobald sie vom Benutzer aktiviert werden.
- 8 Der Benutzer klickt auf eine der durch `commandButtons()` definierten Schaltflächen.
- 9 Dreamweaver führt den damit verknüpften Code aus. Das Dialogfeld wird so lange angezeigt, bis eines der Skripts im Befehl die Funktion `window.close()` aufruft.

Befehle in das Menü Befehle aufnehmen

Dreamweaver fügt Dateien, die sich im Unterordner **Configuration\Commands** befinden, automatisch unten im Menü **Befehle** hinzu. Wenn ein Befehl nicht im Menü **Befehle** aufgeführt werden soll, müssen Sie den nachfolgenden Kommentar in die erste Zeile der Datei schreiben:

```
<!-- MENU-LOCATION=NONE -->
```

Wenn diese Zeile vorhanden ist, erstellt Dreamweaver keine Menüoption für die Datei. Sie müssen den Befehl über `dw.runCommand()` aufrufen.

Einfaches Befehlsbeispiel

Mit dieser einfachen Erweiterung wird zum Menü **Befehle** ein Element hinzugefügt, mit dem Sie ausgewählten Text in Ihrem Dokument entweder in Groß- oder Kleinbuchstaben darstellen können. Wenn Sie auf das Menüelement klicken, wird eine Benutzeroberfläche mit drei Schaltflächen angezeigt, über die Sie Ihre Auswahl festlegen können.

Führen Sie zum Erstellen der Erweiterung die folgenden Schritte aus:

- „Benutzeroberfläche erstellen“ auf Seite 124
- „JavaScript-Code schreiben“ auf Seite 125
- „Erweiterung testen“ auf Seite 129

In diesem Beispiel werden zwei Dateien im Ordner **Commands** erstellt: **ChangeCase.htm**, die den Code für die Benutzeroberfläche enthält, und **ChangeCase.js**, die den JavaScript-Code enthält. Sie können auch nur die Datei **ChangeCase.htm** erstellen und den JavaScript-Code in den head-Abschnitt einfügen.

Benutzeroberfläche erstellen

Die Benutzeroberfläche besteht aus einem Formular, das zwei Optionsschalter enthält, mit denen der Benutzer entweder Klein- oder Großschreibung auswählen kann.

So erstellen Sie die Benutzeroberfläche:

- 1 Erstellen Sie eine neue leere Datei.
- 2 Fügen Sie der Datei den folgenden Code zum Erstellen des Formulars hinzu:

```
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//dialog">
<HTML>
<HEAD>
<!-- Copyright 2001-2002 Macromedia, Inc. All rights reserved. -->
<Title>Make Uppercase or Lowercase</Title>
<SCRIPT SRC="Change Selection Case.js"></SCRIPT>
```

```

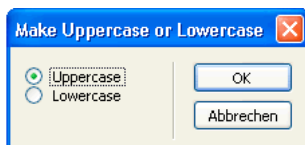
</HEAD>
<BODY>
<form name="uorl">
  <table border="0">
    <!--DWLayoutTable-->
    <tr>
      <td valign="top" nowrap> <p>
        <label>
          <input type="radio" name="RadioGroup1" value="uppercase" checked>
          Uppercase</label>
          <br>
          <label>
            <input type="radio" name="RadioGroup1" value="lowercase">
            Lowercase</label>
        </p></td>
      </tr>
    </table>
  </form>
</BODY>
</HTML>

```

3 Speichern Sie die Datei unter dem Namen **ChangeCase.htm** im Ordner **Configuration\Commands**.

Der Inhalt des Title-Tags, **Make Uppercase or Lowercase**, wird in der oberen Leiste des Dialogfeldes angezeigt. Innerhalb des Formulars wird das Layout der Elemente durch eine Tabelle mit zwei Zellen festgelegt. In den Tabellenzellen befinden sich die beiden Optionsschalter, **Uppercase** und **Lowercase**. Die Option **Uppercase** hat das Attribut `checked`. Dadurch ist er standardmäßig ausgewählt und es wird sichergestellt, dass der Benutzer entweder eine der beiden Optionen auswählt oder den Befehl abbricht.

Das Formular sieht wie in der folgenden Abbildung aus.



Die Funktion `commandButtons()` stellt die Schaltflächen **OK** und **Abbrechen** zur Verfügung, mit denen der Benutzer seine Auswahl eingeben oder den Vorgang abbrechen kann. Weitere Informationen finden Sie unter „`commandButtons()`“ auf Seite 130.

JavaScript-Code schreiben

Das folgende Beispiel besteht aus zwei erweiterten API-Funktionen, `canAcceptCommand()` und `commandButtons()`, die von Dreamweaver aufgerufen werden, sowie einer benutzerdefinierten Funktion, `changeCase()`, die über die Funktion `commandButtons()` aufgerufen wird.

In diesem Beispiel schreiben Sie JavaScript-Code für die folgenden Vorgänge:

- „Feststellen, ob der Befehl aktiviert oder abgeblendet sein soll“ auf Seite 125
- „Funktionen mit den Schaltflächen OK und Abbrechen verknüpfen“ auf Seite 127
- „Den Benutzer Groß- oder Kleinschreibung wählen lassen“ auf Seite 127

Feststellen, ob der Befehl aktiviert oder abgeblendet sein soll

Beim Erstellen eines Befehls müssen Sie zuerst feststellen, wann ein Element aktiviert und wann es abgeblendet werden soll. Wenn ein Benutzer auf das Menü **Befehle** klickt, ruft Dreamweaver die Funktion `canAcceptCommand()` für jedes Menüelement auf, um festzustellen, ob es aktiviert werden soll. Wenn `canAcceptCommand()` den Wert `true` zurückgibt, zeigt Dreamweaver den Text des Menüelements als aktiviert an. Wenn `canAcceptCommand()` den Wert `false` zurückgibt, blendet Dreamweaver das Menüelement ab. In diesem Beispiel ist das Menüelement aktiv, wenn der Benutzer im Dokument Text ausgewählt hat.

So stellen Sie fest, ob der Befehl aktiviert oder abgeblendet sein soll:

- 1 Erstellen Sie eine neue, leere Datei.
- 2 Fügen Sie den folgenden Code hinzu:

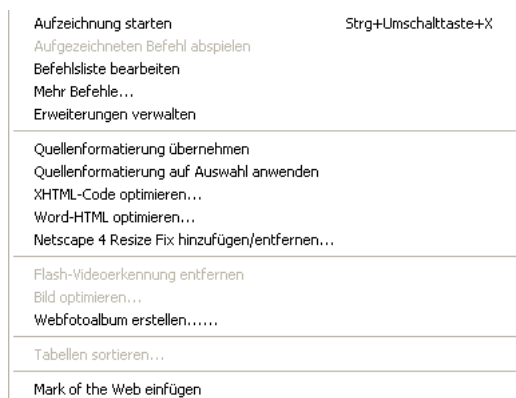
```
function canAcceptCommand(){
    var theDOM = dw.getDocumentDOM(); // das DOM des aktuellen Dokuments abrufen
    var theSel = theDOM.getSelection(); // Anfang und Ende der Auswahl abrufen
    var theSelNode = theDOM.getSelectedNode(); // den ausgewählten Knoten abrufen
    var theChildren = theSelNode.childNodes; // die dem aktuellen Knoten untergeordneten Elemente abrufen
    return (theSel[0] != theSel[1] && (theSelNode.nodeType == Node.TEXT_NODE ||
        theSelNode.hasChildNodes() && (theChildren[0].nodeType == Node.TEXT_NODE)));
}
```

- 3 Speichern Sie die Datei unter dem Namen **ChangeCase.js** im Ordner **Configuration\Commands**.

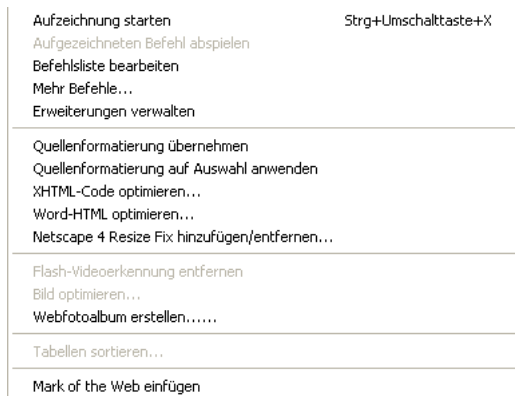
Die ersten Zeilen der Funktion `canAcceptCommand()` rufen den ausgewählten Text ab, indem sie das DOM des Benutzerdokuments abrufen und die Funktion `getSelection()` für das Dokumentobjekt aufrufen. Anschließend ruft die Funktion den Knoten ab, die den ausgewählten Text enthält, gefolgt von den untergeordneten Knoten, wie im folgenden Code dargestellt: Die letzte Zeile überprüft, ob es sich bei der Auswahl oder dem ersten untergeordneten Knoten um Text handelt, und gibt `true` oder `false` zurück.

Der erste Teil der `return`-Anweisung (`theSel[0] != theSel[1]`) überprüft, ob der Benutzer etwas im Dokument ausgewählt hat. Die Variable `theSel` ist ein Array, das die Anfang- und Ende-Offsets der Auswahl im Dokument enthält. Wenn die beiden Werte nicht identisch sind, wurde Inhalt ausgewählt. Wenn die beiden Werte im Array gleich sind, handelt es sich lediglich um die Einfügemarke, und es wurde nichts ausgewählt.

Der nächste Teil der `return`-Anweisung (`&& (theSelNode.nodeType == Node.TEXT_NODE)`) überprüft, ob der ausgewählte Knotentyp „Text“ ist. Falls ja, gibt die Funktion `canAcceptCommand()` den Wert `true` zurück. Ist der Knotentyp kein Text, wird der Test fortgesetzt, um zu überprüfen, ob es untergeordnete Knoten gibt (`|| theSelNode.hasChildNodes()`), und ob der erste untergeordnete Knotentyp „Text“ ist (`&& (theChildren[0].nodeType == Node.TEXT_NODE)`). Sind beide Bedingungen wahr, gibt `canAcceptCommand()` den Wert `true` zurück, und Dreamweaver aktiviert das Menüelement unten im Menü **Befehle**, wie in der Abbildung gezeigt:



Anderenfalls gibt `canAcceptCommand()` den Wert `false` zurück, und Dreamweaver blendet das Menüelement ab, wie in der folgenden Abbildung gezeigt:



Funktionen mit den Schaltflächen OK und Abbrechen verknüpfen

Wenn der Benutzer auf **OK** oder **Abbrechen** klickt, muss die Erweiterung die passende Aktion ausführen. Indem Sie angeben, welche JavaScript-Funktion bei Betätigung der Schaltflächen ausgeführt werden soll, bestimmen Sie die entsprechende Aktion.

So verknüpfen Sie Funktionen mit den Schaltflächen OK und Abbrechen:

- 1 Öffnen Sie die Datei `ChangeCase.js` im Ordner `Configuration\Commands`.
- 2 Fügen Sie am Ende der Datei den folgenden Code ein:

```
function commandButtons() {
    return new Array("OK", "changeCase()", "Cancel", "window.close()");
}
```

- 3 Speichern Sie die Datei.

Die Funktion `commandButtons()` weist Dreamweaver an, die Schaltflächen **OK** und **Abbrechen** anzuzeigen und legt fest, was Dreamweaver ausführt, wenn der Benutzer darauf klickt. Dreamweaver wird von der Funktion `commandButtons()` angewiesen, `changeCase()` aufzurufen, wenn der Benutzer auf **OK** klickt, und `window.close()`, wenn der Benutzer auf **Abbrechen** klickt:

Den Benutzer Groß- oder Kleinschreibung wählen lassen

Wenn der Benutzer auf ein Menüelement klickt, benötigt die Erweiterung einen Mechanismus, mit dem der Benutzer Groß- oder Kleinschreibung auswählen kann. Die Benutzeroberfläche stellt diesen Mechanismus in Form von zwei Optionsschaltern zur Verfügung, die den Benutzern die Auswahl ermöglichen.

So lassen Sie den Benutzer Groß- oder Kleinschreibung wählen:

- 1 Öffnen Sie die Datei `ChangeCase.js`.
- 2 Fügen Sie am Ende der Datei den folgenden Code ein:

```
function changeCase() {
    var uorl;

    // Prüfen, ob der Benutzer Groß- oder Kleinschreibung angefordert hat.
    if (document.forms[0].elements[0].checked)
        uorl = 'u';
    else
        uorl = 'l';

    // Die DOM abrufen.
    var theDOM = dw.getDocumentDOM();
```

```

// Das outerHTML des HTML-Tags abrufen (der
// gesamte Inhalt des Dokuments).
var theDocEl = theDOM.documentElement;
var theWholeDoc = theDocEl.outerHTML;

// Den Knoten mit der Auswahl abrufen.
var theSelNode = theDOM.getSelectedNode();

// Die dem ausgewählten Knoten untergeordneten Elemente abrufen
var theChildren = theSelNode.childNodes;
var i = 0;
if (theSelNode.hasChildNodes()) {
    while (i < theChildren.length) {
        if (theChildren[i].nodeType == Node.TEXT_NODE) {
            var selText = theChildren[i].data;
            var theSel = theDOM.nodeToOffsets(theChildren[0]);
            break;
        }
        ++i;
    }
} else {
    // Den Inhalt des Dokuments abrufen.
    var theSel = theDOM.getSelection();
    // Die Auswahl extrahieren
    var selText = theWholeDoc.substring(theSel[0], theSel[1]);
}
if (uorl == 'u') {
    theDocEl.outerHTML = theWholeDoc.substring(0, theSel[0]) +
        selText.toUpperCase() + theWholeDoc.substring(theSel[1]);
} else {
    theDocEl.outerHTML = theWholeDoc.substring(0, theSel[0]) +
        selText.toLowerCase() + theWholeDoc.substring(theSel[1]);
}

// Die Auswahl auf den Ausgangspunkt zurücksetzen
theDOM.setSelection(theSel[0], theSel[1]);
window.close(); // Erweiterungs-UI schließen
}

```

3 Speichern Sie die Datei unter dem Namen **ChangeCase.js** im Ordner **Configuration\Commands**.

`changeCase()` ist eine benutzerdefinierte Funktion, die von `commandButtons()` aufgerufen wird, wenn der Benutzer auf **OK** klickt. Diese Funktion ändert den ausgewählten Text in Klein- oder Großschreibung. Da in der Benutzeroberfläche Optionsschalter verwendet werden, ist sichergestellt, dass eine Option aktiviert wird. Der Code muss also nicht die Möglichkeit überprüfen, dass der Benutzer keine Auswahl eingibt.

Die Funktion `changeCase()` testet zuerst die Eigenschaft `document.forms[0].elements[0].checked`. Die Eigenschaft `document.forms[0].elements[0]` bezieht sich auf das erste Element im ersten Formular des aktuellen Dokumentobjekts, also in der Benutzeroberfläche der Erweiterung. Die Eigenschaft `checked` hat den Wert `true`, wenn das Element ausgewählt (bzw. aktiviert) wurde, und `false`, wenn es nicht ausgewählt wurde. In der Benutzeroberfläche bezieht sich `elements[0]` auf den ersten Optionsschalter, also auf **Uppercase**. Da einer der Optionsschalter auf jeden Fall ausgewählt sein muss, wenn der Benutzer auf **OK** klickt, wird davon ausgegangen, dass die Auswahl **Lowercase** (Kleinschreibung) lauten muss, wenn sie nicht **Uppercase** (Großschreibung) lautet. Die Funktion setzt die Variable `uorl` auf `u` oder `l`, um die Antwort des Benutzers zu speichern.

Der verbleibende Code in der Funktion ruft den ausgewählten Text ab, konvertiert ihn in Groß- oder Kleinschreibung und kopiert ihn zurück in das Dokument.

Zum Abrufen des ausgewählten Texts für das Dokument des Benutzers ruft die Funktion das DOM ab. Dann ruft es das Stammelement des Dokuments, das `html`-Tag, ab. Abschließend wird das gesamte Dokument in die Variable `theWholeDoc` extrahiert.

Dann ruft `changeCase()` mithilfe von `getSelectedNode()` den Knoten ab, der den ausgewählten Text enthält. Außerdem werden alle untergeordneten Knoten abgerufen (`theSelNode.childNodes`), falls ein Tag ausgewählt wurde, das Text enthält, wie z. B. `<b>text</b>`:

Sind untergeordnete Knoten vorhanden, gibt `hasChildNodes()` den Wert `true` zurück, und der Befehl führt eine Schleife durch den untergeordneten Knoten aus, um nach einem Textknoten zu suchen. Sobald einer gefunden wurde, werden der Text (`theChildren[i].data`) in `selText` und die Offsets des Textknotens in `theSel` gespeichert.

Wenn keine untergeordneten Knoten vorhanden sind, ruft der Befehl `getSelection()` auf und speichert die Anfang- und Ende-Offsets der Auswahl in `theSel`. Anschließend wird der String zwischen den beiden Offsets extrahiert und in `selText` gespeichert.

Die Funktion überprüft dann die Variable `uorl`, um festzustellen, ob der Benutzer „Uppercase“ ausgewählt hat: Ist dies der Fall, schreibt die Funktion den HTML-Code in mehreren Abschnitten in das Dokument zurück: Zuerst der Anfang des Dokuments bis zum Beginn der Auswahl, dann der ausgewählte Text in Großbuchstaben (`selText.toUpperCase()`); zuletzt das Ende des ausgewählten Textes bis zum Ende des Dokuments.

Wenn der Benutzer „Lowercase“ auswählt, führt die Funktion denselben Vorgang aus und ruft dann `selText.toLowerCase()` auf, um den ausgewählten Text in Kleinbuchstaben zu konvertieren.

Schließlich hebt `changeCase()` die Auswahl auf und ruft `window.close()` auf, um die Benutzeroberfläche zu schließen.

Erweiterung testen

Wenn Sie die Dateien im Ordner **Commands** abgelegt haben, können Sie die Erweiterung testen.

So testen Sie die Erweiterung:

1 Starten Sie Dreamweaver neu, oder laden Sie die Erweiterungen neu. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

Daraufhin wird im Menü **Befehle** der Eintrag **ChangeCase** angezeigt.

2 Geben Sie Text in ein Dokument ein.

3 Wählen Sie die Textstelle aus.

***Hinweis:** Der Eintrag **ChangeCase** bleibt so lange abgeblendet, bis Text in einem Dokument ausgewählt wird.*

4 Wählen Sie **ChangeCase** im Menü **Befehle**.

Die Schreibung des Textes ändert sich.

Die Funktionen der Befehls-API

Die benutzerdefinierten Funktionen der Befehls-API sind nicht erforderlich.

canAcceptCommand()

Beschreibung

Diese Funktion stellt fest, ob der Befehl für die aktuelle Auswahl zulässig ist.

***Hinweis:** Definieren Sie `canAcceptCommand()` nur, wenn mindestens in einem Fall `false` zurückgegeben wird. Ist die Funktion nicht definiert, gilt der Befehl als zulässig. Durch diese Annahme wird der Vorgang beschleunigt und die Leistung verbessert.*

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet `true`, wenn der Befehl zulässig ist, anderenfalls `false`. In diesem Fall wird der Befehl im Menü abgeblendet.

Beispiel

Das folgende Beispiel für `canAcceptCommand()` stellt den Befehl nur dann zur Verfügung, wenn eine Tabelle ausgewählt wurde:

```
function canAcceptCommand(){
    var retval=false;
    var selObj=dw.getDocumentDOM.getSelectedNode();
    return (selObj.nodeType == Node.ELEMENT_NODE && ~
        selObj.tagName=="TABLE");{
        retval=true;
    }
    return retval;
}
```

commandButtons()

Beschreibung

Diese Funktion definiert die Schaltflächen für die rechte Seite des Dialogfelds **Optionen** und deren Klickverhalten. Wenn diese Funktion nicht definiert ist, werden keine Schaltflächen angezeigt, und der body-Bereich der Befehlsdatei füllt das gesamte Dialogfeld aus.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array, in dem eine gerade Anzahl von Elementen gespeichert ist. Das erste Element ist ein String mit der Bezeichnung für die oberste Schaltfläche. Das zweite Element ist ein String mit JavaScript-Code, mit dem das Klickverhalten der obersten Schaltfläche bestimmt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel

Im folgenden Beispiel für `commandButtons()` werden drei Schaltflächen definiert: **OK**, **Abbrechen** und **Hilfe**.

```
function commandButtons(){
    return new Array("OK" , "doCommand()" , "Cancel" , ~
        "window.close()" , "Help" , "showHelp()");
}
```

isDOMRequired()

Beschreibung

Diese Funktion ermittelt, ob für die Funktionsfähigkeit des Befehls ein gültiges DOM benötigt wird. Wenn diese Funktion den Wert `true` zurückgibt oder nicht definiert ist, geht Dreamweaver davon aus, dass für den Befehl ein gültiges DOM erforderlich ist und synchronisiert vor der Ausführung die Entwurfsansicht und die Codeansicht des Dokuments. Bei der Synchronisierung werden alle in der Codeansicht vorgenommenen Änderungen in der Entwurfsansicht aktualisiert.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet `true`, falls für die Funktion eines Befehls ein gültiges DOM erforderlich ist, anderenfalls `false`.

receiveArguments()

Beschreibung

Diese Funktion verarbeitet Argumente, die von einem Menüelement oder von der Funktion `dw.runCommand()` übergeben werden.

Argumente

`{arg1}, {arg2}, ... {argN}`

- Wenn in einem `menuitem`-Tag das Attribut `arguments` definiert ist, wird der Wert dieses Attributs als ein bzw. mehrere Argumente an die Funktion `receiveArguments()` übergeben. Argumente können auch über die Funktion `dw.runCommand()` an einen Befehl übergeben werden.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

windowDimensions()

Beschreibung

Diese Funktion legt die Abmessungen für das Dialogfeld **Parameter** fest. Wenn diese Funktion nicht definiert ist, werden die Abmessungen automatisch berechnet.

***Hinweis:** Verwenden Sie diese Funktion nur dann, wenn das Dialogfeld **Optionen** größer als 640 x 480 Pixel sein soll.*

Argumente

`platform`

- Der Wert des Arguments `platform` ist entweder "macintosh" oder "windows", abhängig von der verwendeten Plattform.

Rückgabewerte

Ein String der Form "widthInPixels,heightInPixels".

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da die Schaltflächen für **OK** und **Abbrechen** nicht mit einbezogen sind. Wenn mit den zurückgegebenen Abmessungen nicht für alle Optionen Platz vorhanden ist, werden Bildlaufleisten angezeigt.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Parameter-Dialogfelds auf 648 x 520 Pixel festgelegt:

```
function windowDimensions(){  
    return "648,520";  
}
```


Kapitel 10: Menüs und Menübefehle

Die Menüs in Adobe Dreamweaver CS3 basieren auf der in der Datei `menus.xml` im Ordner **Configuration\Menus** definierten Struktur. Sie können Menübefehle neu anordnen, umbenennen und entfernen, indem Sie die Datei **menus.xml** bearbeiten. Außerdem haben Sie die Möglichkeit, Tastaturbefehle für Menübefehle hinzuzufügen, zu ändern oder zu entfernen. Dies kann in der Regel jedoch einfacher mithilfe des Editors für Tastaturbefehle durchgeführt werden (siehe Dreamweaver-Hilfe). Änderungen an Dreamweaver-Menüs werden nach dem Neustart oder Neuladen von Erweiterungen angezeigt.

Wenn Sie in einem Mehrbenutzer-Betriebssystem Änderungen in Dreamweaver vornehmen, die sich auf die Datei `menus.xml` auswirken (beispielsweise Änderungen an Tastaturbefehlen mithilfe des Editors für Tastaturbefehle), erstellt Dreamweaver eine neue Datei `menus.xml` in Ihrem Konfigurationsordner. Um die Datei `menus.xml` in einem Mehrbenutzer-Betriebssystem anzupassen, arbeiten Sie mit einer Kopie der Datei in Ihrem Konfigurationsordner (oder kopieren Sie die Masterdatei `menus.xml` in Ihren Konfigurationsordner, falls Dreamweaver dort nicht bereits eine Version abgelegt hat). Weitere Informationen finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 70.

Beim Öffnen der Datei `menus.xml` in einem XML-Editor werden unter Umständen Fehlermeldungen bezüglich der Et-Zeichen (&) in der Datei angezeigt. Öffnen Sie nach Möglichkeit die Datei `menus.xml` in einem Texteditor und nicht in Dreamweaver. Grundlegende Informationen zu XML finden Sie in der Dreamweaver-Hilfe.

Hinweis: Erstellen Sie immer eine Sicherungskopie der Datei `menus.xml` und von allen anderen Dreamweaver-Konfigurationsdateien, bevor Sie Änderungen an diesen Dateien vornehmen. Beim Bearbeiten der Menü-Konfigurationsdatei können einem schnell Fehler unterlaufen, und Sie können eine vorherige Gruppe von Menüs nur durch Ersetzen der Datei **menus.xml** wiederherstellen. Falls Sie vergessen sollten, eine Sicherungskopie anzulegen, befindet sich im Ordner **Configuration** eine Kopie der Standarddatei **menus.xml** mit der Bezeichnung `menus.bak`. Um die Standardmenüs wiederherzustellen, ersetzen Sie die Datei `menus.xml` durch eine Kopie der Datei **menus.bak**.

Die Datei `menus.xml`

In der Datei **menus.xml** sind Menüleisten, Menübefehle, Trennzeichen, eine Liste mit Kurzbefehlen und Tastaturbefehle enthalten. Diese Elemente werden durch XML-Tags beschrieben, die in einem Texteditor bearbeitet werden können.

Hinweis: Gehen Sie bei Menüänderungen sorgfältig vor. Dreamweaver verarbeitet keine Menüs oder Menübefehle, die XML-Syntaxfehler aufweisen.

Eine Menüleiste (durch öffnende und schließende `menubar`-Tags markiert) kann sich aus einem einzelnen Menü oder einer Gruppe von Menüs zusammensetzen. So ist beispielsweise für jedes Kontextmenü eine Hauptmenüleiste, eine Site-Bedienfeld-Menüleiste (nur unter Windows) und eine Menüleiste vorhanden. In jeder Menüleiste sind ein oder mehrere Menüs vorhanden. Ein Menü wird durch ein `menu`-Tag definiert. Jedes Menü enthält einen oder mehrere Menübefehle, die durch ein `menuitem`-Tag und die entsprechenden Attribute definiert sind. Ein Menü kann außerdem Trennzeichen (definiert durch `separator`-Tags) und Untermenüs enthalten.

Neben den Tastaturbefehlen für Menübefehle stellt Dreamweaver zahlreiche andere Tastaturbefehle zur Verfügung, darunter alternative Kurzbefehle oder solche, die nur in bestimmten Zusammenhängen verfügbar sind. `<Strg>+<Y>` (Windows) oder `<Befehl>+<Y>` (Macintosh) ist der Kurzbefehl für das Wiederherstellen eines Vorgangs. Bei `<Strg>+<Umschalt>+<Z>` oder `<Befehl>+<Umschalt>+<Z>` handelt es sich um den entsprechenden alternativen Kurzbefehl. Diese alternativen Kurzbefehle und andere Kurzbefehle, die nicht durch Tags für Menübefehle wiedergegeben werden können, sind in der Kurzbefehlsliste in der Datei **menus.xml** definiert. Jede dieser Listen (durch ein `shortcutlist`-Tag definiert) enthält einen oder mehrere Kurzbefehle, die durch das entsprechende `shortcut`-Tag beschrieben werden.

Im folgenden Abschnitt wird die Syntax der Tags der Datei `menus.xml` beschrieben. Optionale Attribute werden in den Attributlisten durch geschweifte Klammern ({}) markiert. Attribute, die nicht durch geschweifte Klammern markiert sind, sind erforderlich.

<menubar>

Beschreibung

Stellt Informationen zu einer Menüleiste in der Dreamweaver-Menüstruktur zur Verfügung.

Attribute

name, {app}, id, {platform}

- **name** Der Name der Menüleiste. Auch wenn es sich bei `name` um ein erforderliches Attribut handelt, kann ihm der Wert `""` zugeordnet werden.
- **app** Der Name der Anwendung, in der die Menüleiste angezeigt wird. Wird momentan nicht verwendet.
- **id** Die Menü-ID der Menüleiste. Jede Menü-ID in der Datei `menus.xml` muss eindeutig sein.
- **platform** Gibt an, dass die Menüleiste nur auf der angegebenen Plattform angezeigt wird. Gültige Werte sind `"win"` und `"mac"`.

Inhalt

Dieses Tag muss mindestens ein `menu`-Tag enthalten.

Container

Keine

Beispiel

Für die Hauptmenüleiste (Dokumentfenster) wird folgendes Menüleisten-Tag verwendet:

```
<menubar name="Main Window" id="DWMainWindow">
<!-- menu-Tags hier einfügen -->
</menubar>
```

<menu>

Beschreibung

Stellt Informationen zu einem Menü oder einem Untermenü in der Dreamweaver-Menüstruktur zur Verfügung.

Attribute

name, {app}, id, {platform}, {showIf}

- **name** Der Name des auf der Menüleiste angezeigten Menüs. Zum Festlegen der Zugriffstaste (Abkürzungstaste) für das Menü unter Windows verwenden Sie einen Unterstrich (`_`) vor dem entsprechenden Buchstaben. Der Unterstrich wird auf dem Macintosh automatisch entfernt.
- **app** Der Name der Anwendung, in der das Menü angezeigt wird. Wird momentan nicht verwendet.
- **id** Die Menü-ID des Menüs. Jede ID in der Datei muss eindeutig sein.
- **platform** Gibt an, dass das Menü nur auf der angegebenen Plattform angezeigt wird. Gültige Werte sind `"win"` und `"mac"`.
- **showIf** Gibt an, dass das Menü nur angezeigt wird, wenn der jeweilige Dreamweaver-Enabler auf `true` gesetzt ist. Mögliche Enabler: `_SERVERMODEL_ASP`, `_SERVERMODEL_ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von Macromedia ColdFusion von Adobe), `_SERVERMODEL_CFML_UD4` (für UltraDev Version 4 von ColdFusion), `_SERVERMODEL_PHP`, `_FILE_TEMPLATE`, `_VIEW_CODE`, `_VIEW_DESIGN`, `_VIEW_LAYOUT`, `_VIEW_EXPANDED_TABLES`, and `_VIEW_STANDARD`. Sie können mehrere Enabler angeben, indem Sie sie durch ein Komma (gleichbedeutend mit AND) trennen. Zur Angabe von NOT können Sie `"!"` verwenden. Wenn zum Beispiel ein Menü auf einer ASP-Seite nur in Codeansicht angezeigt werden soll, geben Sie das Attribut wie folgt an: `showIf="_VIEW_CODE, _SERVERMODEL_ASP"`.

Inhalt

Dieses Tag enthält mindestens ein `menuitem`-Tag und mindestens ein `separator`-Tag. Außerdem können andere `menu`-Tags (zum Erstellen von Untermenüs) und Standard-HTML-Kommentar-Tags enthalten sein.

Container

Dieses Tag muss in einem `menubar`-Tag enthalten sein.

Beispiel

```
<menu name="_File" id="DWMenu_File">
    <!-- menuitem-, separator-, menu- und comment-Tags hier einfügen -->
</menu>
```

<menuitem>

Beschreibung

Definiert einen Menübefehl für ein Dreamweaver-Menü.

Attribute

`name`, `id`, `{app}`, `{key}`, `{platform}`, `{enabled}`, `{arguments}`, `{command}`, `{file}`, `{checked}`, `{dynamic}`, `{isdomrequired}`, `{showIf}`

- **name** Der im Menü angezeigte Name des Menübefehls. Durch einen Unterstrich wird angezeigt, dass es sich beim nächsten Buchstaben um die Zugriffstaste (Abkürzungstaste) für den Befehl handelt. Diese Funktion steht nur für Windows zur Verfügung.
- **id** Von Dreamweaver verwendet, um das Element zu identifizieren. Diese ID muss in der Menüstruktur eindeutig sein. Wenn Sie der Datei `menus.xml` neue Menübefehle hinzufügen, müssen Sie die Eindeutigkeit sicherstellen, indem Sie für jede Menübefehls-ID Ihren Firmennamen oder einen anderen eindeutigen String als Präfix verwenden.
- **app** Der Name der Anwendung, in welcher der Menübefehl angezeigt wird. Wird momentan nicht verwendet.
- **key** Tastaturbefehl, falls vorhanden. Legen Sie Modifizierungstasten mithilfe folgender Strings fest:
 - `Cmd` gibt die Taste `<Strg>` (Windows) oder die Befehlstaste (Macintosh) an.
 - `Alt` und `Opt` geben die Taste `<Alt>` (Windows) oder die Wahltaste (Macintosh) an.
 - `Shift` gibt die Umschalttaste auf beiden Plattformen an.
 - `Ctrl` gibt die Steuerungstaste auf beiden Plattformen an.
 - Durch ein Pluszeichen (+) werden die Modifizierungstasten voneinander getrennt, wenn für einen Kurzbefehl mehrere Tasten gedrückt werden müssen. Beispielsweise bedeutet `Cmd+Opt+5` im Attribut `key`, dass der Menübefehl ausgeführt wird, wenn der Benutzer die Tasten `<Strg>+<Alt>+<5>` (Windows) bzw. `<Befehl>+<Option>+<5>` (Macintosh) drückt.
 - Sondertasten werden durch den Namen angegeben: `F1` bis `F12`, `PgDn`, `PgUp`, `Home`, `End`, `Ins`, `Del`, `Tab`, `Esc`, `BkSp` und `Space`. Den Sondertasten können auch Modifizierungstasten zugewiesen werden.
- **platform** Gibt die Plattform an, auf der das Element angezeigt wird. Gültige Werte sind `"win"` für Windows oder `"mac"` für Macintosh. Wenn Sie das Attribut `platform` nicht angeben, wird der Menübefehl auf beiden Plattformen angezeigt. Wenn die Verhaltensweise eines Menübefehls auf unterschiedlichen Plattformen unterschiedlich sein soll, müssen Sie zwei Menübefehle mit dem gleichen Namen (aber unterschiedlichen IDs) angeben: einen mit `platform="win"` und den anderen mit `platform="mac"`.
- **enabled** Stellt JavaScript-Code zur Verfügung (in der Regel einen JavaScript-Funktionsaufruf), der ermittelt, ob der Menübefehl momentan aktiviert ist. Wenn die Funktion den Wert `false` zurückgibt, ist der Menübefehl abgeblendet. Der Standardwert lautet `"true"`. Der Eindeutigkeit halber wäre es aber besser, immer einen Wert anzugeben, auch wenn der Wert `"true"` lautet.
- **arguments** Stellt Argumente zur Verfügung, die Dreamweaver an den Code in der mit dem Attribut `file` angegebenen JavaScript-Datei übergeben soll. Setzen Sie Argumente innerhalb der doppelten Anführungszeichen (") zur Begrenzung des Attributwerts in einfache Anführungszeichen (').
- **command** Gibt einen JavaScript-Ausdruck an, der ausgeführt wird, wenn der Benutzer dieses Element im Menü auswählt. Verwenden Sie für komplexen JavaScript-Code stattdessen eine JavaScript-Datei (angegeben im `file`-Attribut). Sie müssen für jeden Menübefehl entweder das Attribut `file` oder `command` angeben.

- **file** Der Name einer HTML-Datei mit JavaScript zur Steuerung des Menübefehls. Geben Sie einen Pfad zur Datei relativ zum Ordner **Configuration** an. (Beispielsweise wird der Menübefehl **Hilfe > Willkommen** durch `file="Commands/Welcome.htm"` angegeben.) Das Attribut `file` überschreibt die Attribute `command`, `enabled` und `checked`. Sie müssen für jeden Menübefehl entweder das Attribut `file` oder `command` angeben. Weitere Informationen zum Erstellen einer Befehlsdatei mithilfe des Bedienfelds **Verlauf** finden Sie in der Dreamweaver-Hilfe. Weitere Informationen zum Schreiben benutzerdefinierter JavaScript-Befehle finden Sie in „Befehle“ auf Seite 123.
- **checked** Ein JavaScript-Ausdruck, der angibt, ob sich neben dem Menübefehl ein Häkchen befindet. Wenn für den Ausdruck der Wert `true` gilt, wird neben dem Element ein Häkchen angezeigt.
- **dynamic** Wenn dieses Attribut vorhanden ist, wird angegeben, dass der Menübefehl dynamisch durch eine HTML-Datei bestimmt wird. Die Datei enthält JavaScript-Code, mit dem der Text und der Status des Menübefehls bestimmt werden. Wenn Sie ein Tag als `dynamic` festlegen, müssen Sie auch das Attribut `file` angeben.
- **isdomrequired** Gibt an, ob die Entwurfs- und Codeansicht vor dem Ausführen des Codes für diesen Menübefehl synchronisiert werden sollen. Gültige Werte sind `"true"` (Standard) und `"false"`. Wenn Sie dieses Attribut auf `"false"` setzen, wird das Dreamweaver-Dokumentobjektmodell (DOM) für Änderungen an der Datei, die mit diesem Menübefehl entsteht, nicht verwendet. Weitere Informationen zum DOM finden Sie in „Das Dreamweaver-Dokumentobjektmodell“ auf Seite 87).
- **showIf** Gibt an, dass das Menüelement nur angezeigt wird, wenn der jeweilige Dreamweaver-Enabler auf `true` gesetzt ist. Mögliche Enabler: `_SERVERMODEL_ASP`, `_SERVERMODEL ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von ColdFusion), `_SERVERMODEL_CFML_UD4` (für UltraDev Version 4 von ColdFusion), `_SERVERMODEL_PHP`, `_FILE_TEMPLATE`, `_VIEW_CODE`, `_VIEW_DESIGN`, `_VIEW_LAYOUT`, `_VIEW_EXPANDED_TABLES` und `VIEW_STANDARD`. Sie können mehrere Enabler angeben, indem Sie sie durch ein Komma (gleichbedeutend mit AND) trennen. Zur Angabe von NOT können Sie `"!"` verwenden. Wenn der Menübefehl beispielsweise in der Codeansicht aber nicht auf einer ColdFusion-Seite angezeigt werden soll, geben Sie für das Attribut `showIf=" _VIEW_CODE, !_SERVERMODEL_CFML"` an.

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `menu`-Tag enthalten sein.

Beispiel

```
<menuitem name="_New" key="Cmd+N" enabled="true" command="dw.createDocument()" id="DWMenu_File_New" />
```

<separator>**Beschreibung**

Gibt an, dass ein Trennzeichen an der entsprechenden Stelle im Menü angezeigt werden soll.

Attribute

{app}

`app` Der Name der Anwendung, in der das Trennzeichen angezeigt wird. Wird momentan nicht verwendet.

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `menu`-Tag enthalten sein.

Beispiel

```
<separator />
```

<shortcutlist>

Beschreibung

Gibt eine Kurzbefehlsliste in der Datei **menus.xml** an.

Attribute

{app}, id, {platform}

- **app** Der Name der Anwendung, in der die Kurzbefehlsliste angezeigt wird. Wird momentan nicht verwendet.
- **id** Die ID für die Kurzbefehlsliste. Diese ID muss mit der Menü-ID für die Menüleiste (oder für das Kontextmenü) in Dreamweaver übereinstimmen, die mit den Kurzbefehlen verknüpft ist. Gültige Werte sind "DWMainWindow", "DWMainSite", "DWTimelineContext" und "DWHTMLContext".
- **platform** Gibt an, dass die Kurzbefehlsliste nur auf der angegebenen Plattform angezeigt wird. Gültige Werte sind "win" und "mac".

Inhalt

Dieses Tag kann ein oder mehrere `shortcut`-Tags enthalten. Es kann außerdem ein oder mehrere Kommentartags enthalten (die die gleiche Syntax wie HTML-Kommentartags aufweisen).

Container

Keiner

Beispiel

```
<shortcutlist id="DWMainWindow">
<!-- shortcut- und comment-Tags hier einfügen -->
</shortcutlist>
```

<shortcut>

Beschreibung

Gibt einen Tastaturbefehl in der Datei **menus.xml** an.

Attribute

key, {app}, {platform}, {file}, {arguments}, {command}, id, {name}

- **key** Die Tastenkombination, über die der Tastaturbefehl ausgeführt wird. Weitere Informationen zur Syntax finden Sie unter „<menuitem>“ auf Seite 135.
- **app** Der Name der Anwendung, in der der Kurzbefehl verfügbar ist. Wird momentan nicht verwendet.
- **platform** Gibt an, dass der Kurzbefehl nur für die angegebene Plattform gültig ist. Gültige Werte sind "win" und "mac". Wenn Sie dieses Attribut nicht angeben, kann der Kurzbefehl auf beiden Plattformen verwendet werden.
- **file** Der Pfad zur einer Datei mit dem JavaScript-Code, den Dreamweaver bei Verwendung des Tastaturbefehls ausführt. Das Attribut `file` überschreibt das `command`-Attribut. Sie müssen für jeden Kurzbefehl entweder das Attribut `file` oder `command` angeben.
- **arguments** Stellt Argumente zur Verfügung, die Dreamweaver an den Code in der mit dem Attribut `file` angegebenen JavaScript-Datei übergeben soll. Setzen Sie Argumente innerhalb der doppelten Anführungszeichen (") zur Begrenzung des Attributwerts in einfache Anführungszeichen (').
- **command** Der JavaScript-Code, den Dreamweaver bei der Verwendung des Tastaturbefehls ausführt. Geben Sie für jeden Kurzbefehl entweder das Attribut `file` oder `command` an.
- **id** Ein eindeutiger Kennzeichner für einen Kurzbefehl.
- **name** Ein Name für den über einen Tastaturbefehl ausgeführten Befehl im Stile eines Menübefehlsnamens. Beispielsweise lautet das Attribut `name` für den F12-Kurzbefehl "Vorschau in Primärbrowser".

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `shortcutlist`-Tag enthalten sein.

Beispiel

```
<shortcut key="Cmd+Shift+Z" file="Menus/MM/Edit_Clipboard.htm"
arguments="'redo'" id="DWShortcuts_Edit_Redo" />
```

<tool>

Beschreibung

Stellt ein Tool dar. Es enthält sämtliche Kurzbefehle für das Tool in Form von Subtags in der Datei `menus.xml`.

Attribute

{name}, id

- **name** Eine lokalisierte Version des Toolnamens.
- **id** Die internationale Tool-Kennung des Tools, auf das sich die Kurzbefehle beziehen.

Inhalt

Dieses Tag kann eines oder mehrere der Tags `activate`, `override` und `action` enthalten.

Container

Dieses Tag muss in einem `menu`-Tag enthalten sein.

Beispiel

```
<tool name="Hand tool" id="com.Macromedia.dreamweaver.tools.hand">
  <!-- tool tags here -->
</tool>
```

<action>

Beschreibung

Enthält die Tastenkombination und den JavaScript-Code, der ausgeführt werden soll, wenn das Tool aktiv ist und die Tastenkombination betätigt wird.

Attribute

{name}, key, command, id

- **name** Eine lokalisierte Version der Aktion.
- **key** Die Tastenkombination für die Ausführung der Aktion. Weitere Informationen zur Syntax finden Sie unter „<menuitem>“ auf Seite 135.
- **command** Die auszuführenden JavaScript-Anweisungen. Dieses Attribut hat dasselbe Format wie das Attribut `command` von „<shortcut>“ auf Seite 137.
- **id** Eine eindeutige ID für Verweise auf die Aktion.

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `tool`-Tag enthalten sein.

Beispiel

```
<action name="Set magnification to 50%" key="5" command="dw.activeViewScale = 0.50" id="DWTools_Zoom_50"
/>
```


<activate>

Beschreibung

Enthält die Tastenkombination zum Aktivieren des Tools.

Attribute

{name}, key, id

- **name** Eine lokalisierte Version der Aktion.
- **key** Die Tastenkombination, mit der das Tool aktiviert wird. Weitere Informationen zur Syntax finden Sie unter „<menuitem>“ auf Seite 135.
- **id** Eine eindeutige ID für Verweise auf die Aktion.

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `tool`-Tag enthalten sein.

Beispiel

```
<activate name="Switch to Hand tool" key="H" id="DWTools_Hand_Active1" />
```

<override>

Beschreibung

Enthält die Tastenkombination zum kurzzeitigen Aktivieren des Tools. Während der Verwendung eines anderen Hilfsmittels kann der Benutzer diese Tastenkombination betätigen und gedrückt halten, um auf dieses Tool umzuschalten.

Attribute

{name}, key, id

- **name** Eine lokalisierte Version der Aktion.
- **key** Die Tastenkombination für die Schnellaktivierung des Tools. Weitere Informationen zur Syntax finden Sie unter „<menuitem>“ auf Seite 135.
- **id** Eine eindeutige ID für Verweise auf die Aktion.

Inhalt

Keiner (leeres Tag)

Container

Dieses Tag muss in einem `tool`-Tag enthalten sein.

Beispiel

```
<override name="Quick switch to Hand tool" key="Space" id="DWTools_Hand_Override" />
```

Ändern von Menüs und Menübefehlen

In der Datei **menus.xml** können Sie Menübefehle innerhalb eines Menüs oder zwischen Menüs verschieben, Trennzeichen hinzufügen oder entfernen und Menüs auf einer Menüleiste oder zwischen verschiedenen Menüleisten verschieben.

Sie können Elemente in oder aus Kontextmenüs auf die gleiche Art verschieben wie bei anderen Menüs.

Weitere Informationen hierzu finden Sie unter „Die Datei menus.xml“ auf Seite 133.

So verschieben Sie einen Menübefehl:

- 1 Beenden Sie Dreamweaver.
- 2 Erstellen Sie eine Sicherungskopie der Datei **menus.xml**.
- 3 Öffnen Sie die Datei **menus.xml** in einem Texteditor, z. B. in BBEdit, Macromedia® HomeSite® oder WordPad. (Öffnen Sie die Datei jedoch nicht in Dreamweaver.)
- 4 Schneiden Sie ein vollständiges `menuitem` Tag aus, angefangen bei `<menuitem` bis hin zu `>` am Ende.
- 5 Setzen Sie die Einfügemarke an die gewünschte Stelle für den Menübefehl. (Es muss sich zwischen einem `menu`-Tag und dem entsprechenden `/menu`-Tag befinden.)
- 6 Fügen Sie den Menübefehl an der neuen Stelle ein.

So erstellen Sie ein Untermenü und verschieben einen Menübefehl:

- 1 Setzen Sie die Einfügemarke in ein Menü (an eine beliebige Stelle zwischen einem `menu`-Tag und dem entsprechenden `/menu`-Tag).
- 2 Fügen Sie ein neues `menu`-Tag und `/menu`-Tag in das Menü ein.
- 3 Fügen Sie dem neuen Untermenü neue Menübefehle hinzu.

So fügen Sie ein Trennzeichen zwischen zwei Menübefehlen ein:

- Fügen Sie ein `separator/-` Tag zwischen die beiden `menuitem`-Tags ein.

So entfernen Sie ein vorhandenes Trennzeichen:

- Löschen Sie die entsprechende `separator/-` Zeile.

So verschieben Sie ein Menü:

- 1 Beenden Sie Dreamweaver.
- 2 Erstellen Sie eine Sicherungskopie der Datei **menus.xml**.
- 3 Öffnen Sie die Datei **menus.xml** in einem Texteditor, z. B. in BBEdit, HomeSite oder WordPad. (Öffnen Sie die Datei jedoch nicht in Dreamweaver.)
- 4 Schneiden Sie ein Menü und seinen Inhalt aus, beginnend beim öffnenden `menu` bis hin zum schließenden `/menu`-Tag.
- 5 Setzen Sie die Einfügemarke an die gewünschte Stelle für das Menü. (Es muss sich zwischen einem `menubar`-Tag und dem entsprechenden `/menubar`-Tag befinden.)
- 6 Fügen Sie das Menü an der neuen Stelle ein.

Name eines Menübefehls oder Menüs ändern

Sie können auf einfache Art den Namen eines Menübefehls oder Menüs in der Datei **menus.xml** ändern.

So ändern Sie den Namen eines Menübefehls oder Menüs:

- 1 Beenden Sie Dreamweaver.
- 2 Erstellen Sie eine Sicherungskopie der Datei **menus.xml**.
- 3 Öffnen Sie die Datei **menus.xml** in einem Texteditor, z. B. in HomeSite, BBEdit oder WordPad. (Öffnen Sie die Datei jedoch nicht in Dreamweaver.)
- 4 Wenn Sie einen Menübefehl ändern möchten, suchen Sie das entsprechende `menuitem`-Tag, und ändern Sie den Wert des `name`-Attributs. Wenn Sie ein Menü ändern möchten, suchen Sie das entsprechende `menu`-Tag, und ändern Sie den Wert des `name`-Attributs. Ändern Sie dabei auf keinen Fall das `id`-Attribut.
- 5 Speichern und schließen Sie die Datei **menus.xml**. Starten Sie anschließend Dreamweaver neu, um die Änderungen anzuzeigen.

Bearbeiten:Tastaturbefehle

Wenn die Standard-Tastaturbefehle nicht Ihren Erwartungen entsprechen, können Sie vorhandene Tastaturbefehle entfernen und neue hinzufügen. Dies können Sie auf einfache Art mit dem Editor für Tastaturbefehle durchführen. (Weitere Informationen hierzu finden Sie in der Dreamweaver-Hilfe). Sie haben auch die Möglichkeit, Tastaturbefehle direkt in der Datei **menus.xml** zu ändern. Hierbei können Ihnen aber leichter Fehler unterlaufen, als wenn Sie mit dem Editor für Tastaturbefehle arbeiten.

So ändern Sie einen Tastaturbefehl:

- 1 Beenden Sie Dreamweaver.
- 2 Erstellen Sie eine Sicherungskopie der Datei **menus.xml**.
- 3 Öffnen Sie die Datei **menus.xml** in einem Texteditor, z. B. in BBEdit, HomeSite oder WordPad. (Öffnen Sie die Datei jedoch nicht in Dreamweaver.)
- 4 Suchen Sie in der Tastaturbefehlstabelle (verfügbar über das Dreamweaver Support Center unter http://www.adobe.com/go/dreamweaver_support_de) nach einem Befehl, der noch nicht verwendet wird, oder den Sie neu zuweisen möchten.

Wenn Sie einen Befehl neu zuweisen, tragen Sie ihn in ein ausgedrucktes Exemplar der Tabelle ein, um ihn später nachschlagen zu können.

- 5 Suchen Sie den Menübefehl, dem der Tastaturbefehl zugeordnet ist, und entfernen Sie das Attribut `key="shortcut"` aus diesem Menübefehl.
- 6 Suchen Sie den Menübefehl, dem der Tastaturbefehl zugewiesen oder neu zugewiesen werden soll.
- 7 Ist für den Menübefehl bereits ein Tastaturbefehl vorhanden, suchen Sie das `key`-Attribut in der Zeile. Wenn kein Kurzbefehl vorhanden ist, fügen Sie `key=""` an einer beliebigen Stellen zwischen den Attributen im `menuitem`-Tag ein.
- 8 Geben Sie zwischen die Anführungszeichen (") des `key`-Attributs den neuen Tastaturbefehl ein.

Setzen Sie bei Eingabe einer Tastenkombination ein Pluszeichen (+) zwischen die einzelnen Tastennamen. Weitere Informationen zu Modifizierungstasten finden Sie in der Beschreibung zum `menuitem`-Tag unter „<menuitem>“ auf Seite 135.

Wenn der Kurzbefehl an einer anderen Stelle verwendet wird und Sie ihn nicht löschen, gilt der Befehl nur für den ersten Menübefehl in der Datei **menus.xml**, dem er zugeordnet wurde.

Hinweis: Sie können unter Windows und Macintosh den gleichen Kurzbefehl für einen Menübefehl verwenden.

- 9 Notieren Sie sich den neuen Kurzbefehl an der entsprechenden Stelle in der Tastaturbefehlstabelle.

Popup- und Kontextmenüs ändern

Dreamweaver stellt in vielen Bedien- und Dialogfeldern Popup- und Kontextmenüs zur Verfügung. Einige Kontextmenüs sind in der Datei `menus.xml` und andere in XML-Dateien definiert. Sie können Elemente in diesen Menüs hinzufügen, entfernen und ändern. In den meisten Fällen ist es jedoch besser, eine Erweiterung für diese Änderungen zu schreiben.

Folgende Popup- und Kontextmenüs in Dreamweaver sind in XML-Dateien definiert, dabei werden die gleichen Tags wie in der Datei `menus.xml` verwendet:

- Datenquellen (angezeigt im Popupmenü mit dem Pluszeichen (+) des Bedienfelds **Bindungen**) sind in `DataSources.xml`-Dateien in Unterordnern des Ordners **DataSources** definiert.
- Serververhalten (angezeigt im Popupmenü mit dem Pluszeichen (+) des Bedienfelds **Serververhalten**) sind in `ServerBehaviors.xml`-Dateien in Unterordnern des Ordners **ServerBehaviors** definiert.
- Serverformate (angezeigt im Popupmenü mit dem Pluszeichen (+) des Dialogfelds **Formatliste bearbeiten**) sind in `ServerFormats.xml`-Dateien in Unterordnern des Ordners **ServerFormats** definiert.
- Elemente im Popupmenü **Formate** für eine Bindung im Bedienfeld **Bindungen** sind in `Formats.xml`-Dateien in Unterordnern des Ordners **ServerFormats** definiert. Sie können diesem Menü Einträge in Dreamweaver im Dialogfeld **Format hinzufügen** hinzufügen.
- Die Menübefehle für das Dialogfeld **Tag-Bibliothek-Editor** sind in der Datei `TagLibraries\TagImporters\TagImporters.xml` definiert.
- Menübefehle für Parameter im Dialogfeld **Verhalten generieren**, das im Serververhaltensgenerator zu finden ist, sind in der Datei `Shared\Controls\String Menu\Controls.xml` definiert.
- Elemente für mit ColdFusion-Komponenten verknüpfte Kontextmenüs sind in der Datei `Components\ColdFusion\CFCs\CFCsMenus.xml` definiert.
- Elemente für mit ColdFusion-Datenquellen verknüpfte Kontextmenüs sind in der Datei `Components\ColdFusion\DataSources\DataSourcesMenus.xml` definiert.
- Elemente für mit JavaBeans verknüpfte Kontextmenüs sind in der Datei `Components\Jsp\JavaBeans\JavaBeanMenus.xml` definiert.
- Elemente für mit mehreren Serverkomponenten verknüpfte Kontextmenüs sind in XML-Dateien in Unterordnern des Ordners **Components** definiert.

Menübefehle

Mit Menübefehlen können Menüs flexibler und dynamischer gestaltet werden. Menübefehle können wie normale Befehle dazu verwendet werden, praktisch beliebige Bearbeitungsvorgänge am aktuellen Dokument, an anderen geöffneten Dokumenten oder an einem HTML-Dokument auf dem lokalen Laufwerk durchzuführen. Mit der API für Menübefehle wird die normale Befehls-API um verschiedene Merkmale im Zusammenhang mit der Anzeige und dem Aufruf der Befehle im Menüsystem erweitert.

Menübefehle sind HTML-Dateien, die über das Attribut `file` eines Tags vom Typ `menuitem` in der Datei `menus.xml` aufgerufen werden. Der `body`-Bereich einer Menübefehlsdatei kann ein HTML-Formular enthalten, in dem Optionen für den Befehl eingegeben werden können (beispielsweise wie und nach welcher Spalte eine Tabelle sortiert werden soll). Der `head`-Bereich einer Menübefehlsdatei enthält JavaScript-Funktionen, die Formulareingaben aus dem `body`-Bereich verarbeiten und die Änderungen am Benutzerdokument kontrollieren.

Menübefehle werden im Unterordner **Configuration\Menus** innerhalb des Anwendungsordners von Dreamweaver gespeichert.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Menübefehlen:

Pfad	Datei	Beschreibung
Configuration\Menus\	menus.xml	Enthält eine strukturierte Liste von Menüleisten, Menüs, Menübefehlen, Trennzeichen, Kurzbefehlen und Tastaturbefehlen. Bearbeiten Sie diese Datei, wenn Sie neue Menüs und Menübefehle hinzufügen möchten.
Configuration\Menus\	Befehlsname.htm	Enthält die für den Menübefehl erforderlichen Funktionen.

Hinweis: Wenn Sie in Dreamweaver Ihre eigenen Menübefehle hinzufügen, sollten Sie diese der obersten Ebene des Ordners **Menus** oder einem eigens dafür erstellten Unterordner hinzufügen. Der Ordner **MM** ist für die mit Dreamweaver gelieferten Menübefehle reserviert.

Das Menü Befehle ändern

Sie können dem Menü **Befehle** bestimmte Befehle hinzufügen und die Namen ändern, ohne dabei die Datei **menus.xml** zu bearbeiten. Weitere Informationen zur Datei **menus.xml** finden Sie unter „Ändern von Menüs und Menübefehlen“ auf Seite 140.

Hinweis: Der Begriff **Befehl** hat in Dreamweaver zwei Bedeutungen. Genau genommen ist ein **Befehl** eine besondere Art von Erweiterung. In machen Kontexten hingegen wird er synonym für **Menüelement** verwendet. Darunter fallen alle Elemente in einem Dreamweaver-Menü, ungeachtet der Funktions- und Implementierungsweise.

Verwenden Sie das Bedienfeld **Verlauf**, um Befehle zu erstellen, die automatisch in das Menü **Befehle** eingefügt werden sollen. Des Weiteren können Sie mit dem Extension Manager neue Erweiterungen und somit auch Befehle installieren. Weitere Informationen hierzu finden Sie in der Dreamweaver-Hilfe.

Um die Elemente im Menü **Befehle** neu anzuordnen oder Elemente zwischen Menüs zu verschieben, müssen Sie die Datei **menus.xml** bearbeiten.

So benennen Sie einen benutzerdefinierten Befehl um:

- 1 Wählen Sie **Befehle > Befehlsliste bearbeiten**.

Ein Dialogfeld mit allen Befehlen, deren Namen Sie ändern können, wird angezeigt. (Befehle des Standardbefehlsmenüs werden nicht angezeigt und können mit diesem Verfahren nicht geändert werden.)

- 2 Wählen Sie den neu zu benennenden Befehl aus.

- 3 Geben Sie einen neuen Namen ein.

- 4 Klicken Sie auf **Schließen**.

Dem Befehl im Menü **Befehle** wurde ein neuer Name zugewiesen.

So löschen Sie einen benutzerdefinierten Befehl:

- 1 Wählen Sie **Befehle > Befehlsliste bearbeiten**.

Ein Dialogfeld mit allen Befehlen, die gelöscht werden können, wird angezeigt. (Befehle des Standardbefehlsmenüs werden nicht angezeigt und können mit diesem Verfahren nicht gelöscht werden.)

- 2 Wählen Sie den zu löschenden Befehl aus.

- 3 Klicken Sie auf **Löschen**, und bestätigen Sie den Löschvorgang.

Der Befehl wird gelöscht. Die Datei mit dem Code für den Befehl wird ebenfalls gelöscht. Durch das Löschen eines Befehls wird also nicht nur der Menübefehl aus dem Menü entfernt. Vergewissern Sie sich, dass Sie den Befehl wirklich löschen möchten, bevor Sie dieses Verfahren verwenden. Wenn Sie den Befehl aus dem Menü **Befehle** entfernen möchten, ohne dabei die Datei zu löschen, suchen Sie die Datei im Ordner **Configuration\Commands**, und verschieben Sie diese in einen anderen Ordner.

- 4 Klicken Sie auf **Schließen**.

Funktionsweise von Befehlen

Wenn der Benutzer auf ein Menü klickt, das ein Menüelement mit einem verknüpften Menübefehl enthält, laufen folgende Vorgänge ab:

- 1 Falls ein `menuitem`-Tag im Menü das Attribut `dynamic` enthält, wird in der zugehörigen Menübefehlsdatei die Funktion `getDynamicContent()` aufgerufen, um das Menü mit Einträgen zu füllen.
- 2 In jeder mit dem Menü verknüpften Menübefehlsdatei wird die Funktion `canAcceptCommand()` aufgerufen, um zu prüfen, ob der Befehl für diese Auswahl geeignet ist.
 - Wenn die Funktion `canAcceptCommand()` den Wert `false` zurückgibt, wird das Menüelement abgeblendet dargestellt.
 - Falls die Funktion `canAcceptCommand()` den Wert `true` zurückgibt oder nicht definiert ist, wird die Funktion `isCommandChecked()` aufgerufen, um festzustellen, ob neben dem Menüelement ein Häkchen angezeigt werden soll. Wenn die Funktion `isCommandChecked()` nicht definiert ist, wird kein Häkchen angezeigt.
- 3 Die Funktion `setMenuText()` wird aufgerufen, um zu bestimmen, welcher Text im Menü angezeigt werden soll. Wenn die Funktion `setMenuText()` nicht definiert ist, wird der im Tag `menuitem` angegebene Text verwendet.
- 4 Der Benutzer wählt einen Eintrag im Menü aus.
- 5 Falls definiert, wird in der ausgewählten Menübefehlsdatei die Funktion `receiveArguments()` aufgerufen, damit der Befehl die vom Menüelement übergebenen Argumente verarbeiten kann.

Hinweis: Bei dynamischen Menüelementen wird die Kennung des Menüeintrags als einziges Argument übergeben.

- 6 Falls definiert, wird die Funktion `commandButtons()` aufgerufen, um zu bestimmen, welche Schaltflächen auf der rechten Seite des Optionsdialogfelds angezeigt werden sollen und welcher Code beim Klicken auf die Schaltflächen jeweils ausgeführt werden soll.
- 7 Die Menübefehlsdatei wird nach dem Tag `form` durchsucht. Falls ein Formular existiert, ruft Dreamweaver die Funktion `windowDimensions()` auf, um die Größe des Optionsdialogfelds mit den `BODY`-Elementen der Datei zu bestimmen. Wenn Funktion `windowDimensions()` nicht definiert ist, passt Dreamweaver die Größe des Dialogfelds automatisch an.
- 8 Wenn im `body`-Tag der Menübefehlsdatei eine Ereignisprozedur vom Typ `onLoad` enthalten ist, wird der entsprechende Code von Dreamweaver ausgeführt (unabhängig davon, ob ein Dialogfeld angezeigt wird). Wenn kein Dialogfeld angezeigt wird, entfallen die restlichen Schritte.
- 9 Der Benutzer wählt im Dialogfeld die gewünschten Optionen aus. An die einzelnen Felder angehängte Ereignisprozeduren werden von Dreamweaver ausgeführt, sobald sie vom Benutzer aktiviert werden.
- 10 Der Benutzer klickt auf eine der durch die Funktion `commandButtons()` definierten Schaltflächen.
- 11 Dreamweaver führt den mit der Schaltfläche verknüpften Code aus.
- 12 Das Dialogfeld bleibt so lange sichtbar, bis eines der Skripts in der Datei für das Menü „Befehle“ die Funktion `window.close()` aufruft.

Beispiel für einen Menübefehl

In diesem leicht nachzuvollziehenden Beispiel wird die Funktionsweise der Menübefehle **Rückgängig** und **Wiederherstellen** beschrieben. Mit dem Menübefehl **Rückgängig** wird der Bearbeitungsvorgang eines Benutzers rückgängig gemacht, und mit **Wiederherstellen** wird die Rückgängig-Aktion aufgehoben und der letzte Bearbeitungsvorgang wiederhergestellt.

Sie können dieses Beispiel anhand der folgenden Schritte nachvollziehen:

- „Menübefehle erstellen“ auf Seite 145
- „JavaScript-Code schreiben“ auf Seite 145
- „Befehlsdatei im Ordner Menus ablegen“ auf Seite 147

Menübefehle erstellen

Fügen Sie die folgenden HTML-Menü-Tags am Ende der Datei `menus.xml` ein, um ein Menü mit der Bezeichnung **MyMenu** zu erstellen, das die Menüelemente „Undo“ (Rückgängig) und „Redo“ (Wiederherstellen) enthält.

```
<menu name="MyMenu" id="MyMenu_Edit">

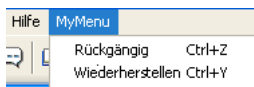
<menuitem name="MyUndo" key="Cmd+Z" file="Menus/MyMenu.htm" arguments="'undo'" id="MyMenu_Edit_Undo" />

<menuitem name="MyRedo" key="Cmd+Y" file="Menus/MyMenu.htm" arguments="'redo'" id="MyMenu_Edit_Redo" />

</menu>
```

Mit dem Attribut `key` werden Tastaturbefehle definiert, die der Benutzer eingeben kann, um ein Menüelement aufzurufen. Mit dem Attribut `file` wird der Name der Befehlsdatei definiert, die Dreamweaver beim Aufrufen des Menüelements ausführt. Der Wert des Attributs `arguments` definiert die Argumente, die Dreamweaver beim Aufrufen der Funktion `receiveArguments()` übergibt.

Die folgende Abbildung zeigt diese Menüelemente:



JavaScript-Code schreiben

Wenn der Benutzer im Menü **MyMenu** den Befehl **Undo** (Rückgängig) oder **Redo** (Wiederherstellen) wählt, ruft Dreamweaver die Befehlsdatei `MyMenu.htm` auf, die durch das Attribut `file` im Tag `menuitem` definiert ist. Erstellen Sie die Befehlsdatei **MyMenu.htm** im Dreamweaver-Ordner **Configuration\Menus**. Fügen Sie dann die drei API-Funktionen `canAcceptCommand()`, `receiveArguments()`, und `setMenuText()` für den Menübefehl hinzu, um den mit den Menüelementen verknüpften Code zu implementieren. Im Folgenden werden diese Funktionen beschrieben.

canAcceptCommand()

Dreamweaver ruft die Funktion `canAcceptCommand()` für jedes Menüelement im Menü **MyMenu** auf, um dessen Status (aktiviert oder deaktiviert) zu bestimmen. In der Datei **MyMenu.htm** überprüft die Funktion `canAcceptCommand()` den Wert des Arguments `arguments[0]`, um festzulegen, ob Dreamweaver ein Rückgängig- Wiederherstellen-Menüelement verarbeitet. Wenn das Argument `"undo"` lautet, ruft die Funktion `canAcceptCommand()` die Enabler-Funktion `dw.canUndo()` auf und zeigt den zurückgegebenen Wert `true` oder `false` an. Lautet das Argument `"redo"` ruft die Funktion `canAcceptCommand()` die Enabler-Funktion `dw.canRedo()` auf und gibt die ermittelten Werte an Dreamweaver zurück. Gibt die Funktion `canAcceptCommand()` den Wert `false` zurück, blendet Dreamweaver das für die Funktion aufgerufene Menüelement ab. Das folgende Beispiel zeigt den Code für die Funktion `canAcceptCommand()`:

```
function canAcceptCommand()
{
    var selarray;
    if (arguments.length != 1) return false;
    var bResult = false;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        bResult = dw.canUndo();
    }
    else if (whatToDo == "redo")
    {
        bResult = dw.canRedo();
    }
    return bResult;
}
```

receiveArguments()

Dreamweaver ruft die Funktion `receiveArguments()` auf, um ein beliebiges, für das `menuitem`-Tag definiertes Argument zu verarbeiten. Für die Wiederherstellen- und Rückgängig-Menüeinträge ruft die Funktion `receiveArguments()` die Funktion `dw.undo()` oder `dw.redo()` auf. Dies hängt davon ab, ob der Wert des Arguments `arguments[0]` "undo" oder "redo" lautet. Mit der Funktion `dw.undo()` wird der vorherige Schritt des Benutzers im Dokumentfenster, Dialogfeld oder Bedienfeld rückgängig gemacht. Mit der Funktion `dw.redo()` wird der zuletzt rückgängig gemachte Vorgang wiederhergestellt.

Hier ist ein Beispiel für den Code der Funktion `receiveArguments()`:

```
function receiveArguments()
{
    if (arguments.length != 1) return;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        dw.undo();
    }
    else if (whatToDo == "redo")
    {
        dw.redo();
    }
}
```

In diesem Befehl verarbeitet die Funktion `receiveArguments()` die Argumente und führt anschließend den Befehl aus. Komplexere Menübefehle können zum Ausführen des Befehls verschiedene Funktionen aufrufen. Im folgenden Code wird beispielsweise überprüft, ob das erste Argument "foo" lautet. Ist dies der Fall, wird die Funktion `doOperationX()` aufgerufen und das zweite Argument übergeben. Wenn das erste Argument "bar" lautet, wird die Funktion `doOperationY()` aufgerufen und das zweite Argument übergeben. Die Funktionen `doOperationX()` oder `doOperationY()` übernehmen die Ausführung des Befehls.

```
function receiveArguments(){
    if (arguments.length != 2) return;

    var whatToDo = arguments[0];

    if (whatToDo == "foo"){
        doOperationX(arguments[1]);
    }else if (whatToDo == "bar"){
        doOperationY(arguments[1]);
    }
}
```

setMenuText()

Dreamweaver ruft die Funktion `setMenuText()` auf, um den für das Menüelement angezeigten Text zu bestimmen. Wird die Funktion `setMenuText()` nicht definiert, verwendet Dreamweaver den im `name`-Attribut des `menuitem`-Tags angegebenen Text.

Die Funktion `setMenuText()` überprüft den von Dreamweaver übergebenen Wert des Arguments `arguments[0]`. Lautet der Wert des Arguments "undo", ruft Dreamweaver die Funktion `dw.getUndoText()` auf. Heißt der Wert "redo", ruft Dreamweaver die Funktion `dw.getRedoText()` auf. Mit der Funktion `dw.getUndoText()` wird der Text zu dem von Dreamweaver rückgängig gemachten Vorgang angezeigt. Wenn der Benutzer beispielsweise mehrere Wiederherstellen-Schritte ausführt, gibt die Funktion `dw.getUndoText()` beispielsweise den Text „Bearbeiten der Quelle rückgängig machen“ aus. Entsprechend gibt die Funktion `dw.getRedoText()` den Text zu dem vom Dreamweaver wiederhergestellten Schritt an. Wenn der Benutzer beispielsweise mehrere Rückgängig-Schritte ausführt, gibt die Funktion `dw.RedoText()` beispielsweise den Text „Bearbeiten der Quelle wiederherstellen“ aus.

Hier ist ein Beispiel für den Code der Funktion `setMenuText()`:

```
function setMenuText()
{
    if (arguments.length != 1) return "";

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
        return dw.getUndoText();
    else if (whatToDo == "redo")
        return dw.getRedoText();
    else return "";
}
```

Befehlsdatei im Ordner Menus ablegen

Zum Implementieren der Menüelemente **Undo** (Rückgängig) und **Redo** (Wiederherstellen) müssen Sie die Befehlsdatei **MyMenu.htm** im Dreamweaver-Ordner **Configuration\Menus** oder in einem benutzerdefinierten Unterordner speichern. Der Speicherort der Datei muss mit dem im `menuitem`-Tag angegebenen Speicherort übereinstimmen. Um auf sie zugreifen zu können, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erweiterungen neu laden“ auf Seite 71.

Wählen Sie zum Ausführen der Menübefehle das aktivierte Menüelement aus. Dreamweaver ruft diese Funktionen wie unter „Funktionsweise von Befehlen“ auf Seite 144 beschrieben in der Befehlsdatei auf.

Ein Beispiel für ein dynamisches Menü

In diesem Beispiel wird das Dreamweaver-Untermenü **Vorschau in Browser** implementiert, in dem die verfügbaren Browser aufgelistet werden. Außerdem werden im benutzerdefinierten Browser die aktuelle Datei oder die im Bedienfeld **Dateien** ausgewählten Dateien angezeigt. Die Implementierung dieses dynamischen Menüs setzt sich aus den folgenden Schritten zusammen:

- „Dynamische Menüelemente erstellen“ auf Seite 147
- „JavaScript-Code schreiben“ auf Seite 148

Dynamische Menüelemente erstellen

Mithilfe der folgenden `menu`-Tags in der Datei `menus.xml` wird das Untermenü **Vorschau in Browser** im Menü **Datei** definiert:

```
<menu name="_Preview in Browser" id="DWMenu_File_PIB">
    <menuitem dynamic name="No Browsers Selected"
        file="Menus/MM/PIB_Dynamic.htm" arguments="'No Browsers'"
        id="DWMenu_File_PIB_Default" />
    <separator />
    <menuitem name="_Edit Browser List..." enabled="true"
        command="dw.editBrowserList()" id="DWMenu_File_PIB_EditBrowserList" />
</menu>
```

Über das erste `menuitem`-Tag wird das Standardmenüelement **Keine Browser ausgewählt** definiert, das im Untermenü angezeigt wird, wenn Sie in den Voreinstellungen des Elements **Vorschau in Browser** keinen Browser angegeben haben. Wenn Sie jedoch als Browser Microsoft Internet Explorer angeben, sieht das Untermenü folgendermaßen aus:



Mit dem Attribut `name` für das erste Menüelement wird die Befehlsdatei `PIB_Dynamic.htm` definiert. Diese Datei enthält die folgende Zeile:

```
<SCRIPT LANGUAGE="javascript" SRC="PIB_Dynamic.js"></SCRIPT>
```

Das `script`-Tag enthält den JavaScript-Code in der Datei `PIB_Dynamic.js`, der mit dem Untermenü **Vorschau in Browser** interagiert. Dieser Code kann direkt in der Datei `PIB_Dynamic.js` gespeichert werden. Wenn Sie ihn jedoch in einer separaten Datei speichern, kann der gleiche Code in mehrere Befehle aufgenommen werden.

JavaScript-Code schreiben

Da das erste `menuitem`-Tag das Attribut `dynamic` enthält, ruft Dreamweaver die Funktion `getDynamicContent()` in der Datei `PIB_Dynamic.js` wie im folgenden Beispiel auf:

```
function getDynamicContent(itemID)
{
    var browsers = null;
    var PIB = null;
    var i;
    var j=0;
    browsers = new Array();
    PIB = dw.getBrowserList();

    for (i=0; i<PIB.length; i=i+2)
    {
        browsers[j] = new String(PIB[i]);

        if (dw.getPrimaryBrowser() == PIB[i+1])
            browsers[j] += "\tF12";
        else if (dw.getSecondaryBrowser() == PIB[i+1])
            browsers[j] += "\tCmd+F12";

        browsers[j] += ";id='"+escQuotes(PIB[i])+"'";

        if (itemID == "DWPopup_PIB_Default")
            browsers[j] = MENU_strPreviewIn + browsers[j];

        j = j+1;
    }
    return browsers;
}
```

Die Funktion `getDynamicContent()` ruft die Funktion `dw.getBrowserList()` auf, um ein Array der im Abschnitt **Vorschau in Browser** der Dreamweaver-Voreinstellungen angegebenen Browsernamen zu erhalten. Das Array enthält den Namen der einzelnen Browser und den Pfad zur ausführbaren Datei. Als Nächstes verschiebt die Funktion `getDynamicContents()` für jedes Element im Array (`i=0; i<PIB.length; i=i+2`) den Namen des Browsers (`PIB[i]`) in ein zweites Array mit der Bezeichnung `browsers` (`browsers[j] = new String(PIB[i]);`). Wurde der Browser als Primär- oder Sekundärbrowser angegeben, hängt die Funktion den Namen des Tastaturbefehls ab, über den sie aufgerufen wird. Dann hängt sie den String `";id="` gefolgt vom Browsernamen in einfachen Anführungszeichen an (z. B. `;id='iexplore'`). Lautet das `itemID`-Argument `"DWPopup_PIB_Default"`, setzt die Funktion den String `Vorschau in` vor das Array. Nachdem sie einen Eintrag für jeden in den Voreinstellungen aufgeführten Browser erstellt hat, übergibt die Funktion `getDynamicContent()` das Array `browsers` an Dreamweaver. Wurden keine Browser ausgewählt, gibt die Funktion den Wert `null` zurück, und Dreamweaver zeigt im Menü die Option **Keine Browser ausgewählt** an.

canAcceptCommand()

Dreamweaver ruft anschließend die Funktion `canAcceptCommand()` für jedes `menuitem`-Tag auf, das auf eine Befehlsdatei mit dem `file`-Attribut verweist. Wenn die Funktion `canAcceptCommand()` den Wert `false` zurückgibt, ist das Menüelement abgeblendet. Gibt die Funktion `canAcceptCommand()` den Wert `true` zurück, aktiviert Dreamweaver das Element im Menü. Falls die Funktion den Wert `true` zurückgibt oder nicht definiert ist, wird die Funktion `isCommandChecked()` aufgerufen, um festzustellen, ob neben dem Menüelement ein Häkchen angezeigt werden soll. Wenn die Funktion `isCommandChecked()` nicht definiert ist, wird kein Häkchen angezeigt.

```
function canAcceptCommand()
{
    var PIB = dw.getBrowserList();

    if (arguments[0] == 'primary' || arguments[0] == 'secondary')
        return havePreviewTarget();

    return havePreviewTarget() && (PIB.length > 0);
}
```

Die Funktion `canAcceptCommand()` in der Datei **PIB_Dynamic.js** ruft erneut die im Dialogfeld **Voreinstellungen** erstellte Browserliste ab. Anschließend überprüft sie, ob das erste Argument (`arguments[0]`) primär bzw. sekundär ist. Ist dies der Fall, wird der durch die Funktion `havePreviewTarget()` zurückgegebene Wert angezeigt. Andernfalls werden der Aufruf der Funktion `havePreviewTarget()` *und* die Angabe anderer Browser geprüft (`PIB.length > 0`). Sind die Bedingungen für *beide* Tests erfüllt, gibt die Funktion den Wert `true` zurück. Sind die Bedingungen für mindestens einen Test nicht erfüllt, wird der Wert `false` zurückgegeben.

havePreviewTarget()

Bei `havePreviewTarget()` handelt es sich um eine benutzerdefinierte Funktion, die den Wert `true` zurückgibt, wenn Dreamweaver über ein gültiges Ziel zur Anzeige im Browser verfügt. Bei einem gültigen Ziel handelt es sich um ein Dokument oder um mehrere im Bedienfeld **Dateien** ausgewählte Dateien. Die Funktion `havePreviewTarget()` sieht folgendermaßen aus:

```
function havePreviewTarget()
{
    var bHavePreviewTarget = false;

    if (dw.getFocus(true) == 'site')
    {
        if (site.getFocus() == 'remote')
        {
            bHavePreviewTarget = site.getRemoteSelection().length > 0 &&
                                site.canBrowseDocument();
        }
        else if (site.getFocus() != 'none')
        {
            var selFiles = site.getSelection();

            if (selFiles.length > 0)
            {
                var i;

                bHavePreviewTarget = true;

                for (i = 0; i < selFiles.length; i++)
                {
                    var selFile = selFiles[i];

                    // Bei Serververbindungen handelt es sich bei den Dateien
                    // bereits um Remote-URLs.

                    if (selFile.indexOf(":/") == (-1))
                    {
                        var urlPrefix = "file:///";
                        var strTemp = selFile.substr(urlPrefix.length);

                        if (selFile.indexOf(urlPrefix) == -1)
                            bHavePreviewTarget = false;
                        else if (strTemp.indexOf("/") == -1)
                            bHavePreviewTarget = false;
                        else if (!DWfile.exists(selFile))
                            bHavePreviewTarget = false;
                        else if (DWfile.getAttributes(selFile).indexOf("D") != -1)
                            bHavePreviewTarget = false;
                    }
                }
            }
        }
    }
}
```

```

        }
        else
        {
            bHavePreviewTarget = true;
        }
    }
}
}
}
else if (dw.getFocus() == 'document' ||
dw.getFocus() == 'textView' || dw.getFocus("true") == 'html' )
{
    var dom = dw.getDocumentDOM('document');
    if (dom != null)
    {
        var parseMode = dom.getParseMode();
        if (parseMode == 'html' || parseMode == 'xml')
            bHavePreviewTarget = true;
    }
}

return bHavePreviewTarget;
}

```

Die Funktion `havePreviewTarget()` legt `false` als Standardrückgabewert für den Wert `bHavePreviewTarget` fest. Die Funktion führt zwei grundlegende Tests durch Aufrufen der Funktion `dw.getFocus()` durch, um zu bestimmen, auf welchem Teil der Anwendung der Eingabefokus liegt. Beim ersten Test wird geprüft, ob das Bedienfeld **Dateien** aktiv ist (`if (dw.getFocus(true) == 'site')`). Ist das Bedienfeld **Dateien** nicht aktiv, wird mit dem zweiten Test geprüft, ob ein Dokument (`dw.getFocus() == 'document'`), eine Textansicht (`dw.getFocus() == 'textView'`) oder der Codeinspektor (`dw.getFocus("true") == 'html'`) aktiv ist. Wenn beide Testkriterien falsch sind, wird der Wert `false` zurückgegeben.

Ist das Bedienfeld **Dateien** aktiv, prüft die Funktion, ob die Einstellung für die Ansicht **Remote-Ansicht** lautet. Ist dies der Fall, setzt die Funktion den Wert `bHavePreviewTarget` auf `true`, falls Remote-Dateien (`site.getRemoteSelection().length > 0`) vorhanden sind und sie in einem Browser geöffnet werden können (`site.canBrowseDocument()`). Wenn die Einstellung weder **Remote-Ansicht** noch **Keine** lautet, ruft die Funktion eine Liste der ausgewählten Dateien (`var selFiles = site.getSelection();`) in Form von `file:///`-URLs ab.

Für jedes Element in der ausgewählten Liste prüft die Funktion, ob der String `://"` vorhanden ist. Wenn er nicht gefunden wird, werden mithilfe des Codes eine Reihe von Tests für das Listenelement durchgeführt. Gilt für das Element nicht die Form `file:///`-URL (`if (selFile.indexOf(urlPrefix) == -1)`), wird der Rückgabewert auf `false` festgelegt. Wenn der restliche Teil des Strings nach dem Präfix `file:///` keinen Schrägstrich (`/`) enthält (`if (strTemp.indexOf("/") == -1)`), wird der Rückgabewert auf `false` gesetzt. Ist die Datei nicht vorhanden (`else if (!DWfile.exists(selFile))`), wird der Rückgabewert auf `false` festgelegt. Abschließend wird überprüft, ob es sich bei der angegebenen Datei um einen Ordner handelt (`else if (DWfile.getAttributes(selFile).indexOf("D") != -1)`). Wenn `selFile` ein Ordner ist, wird der Wert `false` zurückgegeben. Handelt es sich beim Ziel um eine Datei, setzt die Funktion den Wert `bHavePreviewTarget` auf `true`.

Wenn auf einem Dokument, der Textansicht oder dem Codeinspektor der Eingabefokus liegt (`else if (dw.getFocus() == 'document' || dw.getFocus() == 'textView' || dw.getFocus("true") == 'html' )`), ruft die Funktion das DOM ab und überprüft, ob es sich beim Dokument um ein HTML- oder XML-Dokument handelt. Ist dies der Fall, setzt die Funktion den Wert `bHavePreviewTarget` auf `true`. Anschließend gibt die Funktion den in `HavePreviewTarget` gespeicherten Wert zurück.

receiveArguments()

Dreamweaver ruft die Funktion `receiveArguments()` auf, damit der Befehl alle vom Menüelement übergebenen Argumente verarbeiten kann. Für das Menü **Preview in Browser** (Vorschau in Browser) bedeutet dies beispielsweise, dass mithilfe der Funktion `receiveArguments()` der vom Benutzer ausgewählte Browser gestartet wird. Die `receiveArguments()` sieht folgendermaßen aus:

```
function receiveArguments()
{
    var whichBrowser = arguments[0];
    var theBrowser = null;
    var i=0;
    var browserList = null;
    var result = false;

    if (havePreviewTarget())
    {
        // Überprüfungscode, ob Aufruf über einen Kurzbefehl stattfindet
        if (whichBrowser == 'primary' || whichBrowser == 'secondary')
        {
            // Pfad des ausgewählten Browsers ermitteln
            if (whichBrowser == 'primary')
            {
                theBrowser = dw.getPrimaryBrowser();
            }
            else if (whichBrowser == 'secondary')
            {
                theBrowser = dw.getSecondaryBrowser();
            }

            // Pfad mit dem Namen des Browsers abstimmen,
            // der im Menü angezeigt wird.
            browserList = dw.getBrowserList();
            while(i < browserList.length)
            {
                if (browserList[i+1] == theBrowser)
                {
                    theBrowser = browserList[i];
                    i+=2;
                }
            }
        }
        else
        {
            theBrowser = whichBrowser;
            // Browser nur bei Auswahl eines gültigen Browsers starten.
            if (theBrowser != "file:/// " && typeof(theBrowser) != "undefined" &&
                theBrowser.length > 0)
            {
                if (dw.getFocus(true) == 'site')
                {
                    // Nur erstes Element der Auswahl abrufen, da
                    // browseDocument() kein Array aufnehmen kann.
                    //dw.browseDocument(site.getSelection()[0],theBrowser);
                    site.browseDocument(theBrowser);
                }
                else
                {
                    dw.browseDocument(dw.getDocumentPath('document'),theBrowser);
                }
            }
        }
    }
}
```

```

else
{
    // Anderenfalls, wenn <F12> oder <Strg>+<F12> gedrückt wurden, Benutzer fragen,
    // ob jetzt ein primärer oder sekundärer Browser angegeben werden soll.
    if (whichBrowser == 'primary')
    {
        result = window.confirm(MSG_NoPrimaryBrowserDefined);
    }
    else if (whichBrowser == 'secondary')
    {
        result = window.confirm(MSG_NoSecondaryBrowserDefined);
    }

    // Wenn auf "OK" geklickt wurde, Dialogfeld für Voreinstellungen im Browser-Bedienfeld
    anzeigen.
    if (result)
        dw.showPreferencesDialog('browsers');
    }
}
}

```

Zunächst setzt die Funktion die Variable `whichBrowser` auf den von Dreamweaver übergebenen Wert `arguments[0]`. Neben dem Festlegen anderer Standardwerte wird `result` mithilfe der Funktion auf den Standardwert `false` gesetzt.

Nach dem Initialisieren der Variablen ruft die Funktion `receiveArguments()` die benutzerdefinierte Funktion `havePreviewTarget()` auf und überprüft das Ergebnis. Ist das Ergebnis richtig, überprüft die Funktion, ob der Benutzer den Primär- bzw. Sekundärbrowser ausgewählt hat. Ist dies der Fall, wird für die Variable `theBrowser` der Pfad der ausführbaren Datei zum Starten des Browsers (`dw.getPrimaryBrowser()` bzw. `dw.getSecondaryBrowser()`) eingestellt. Die Funktion führt anschließend eine Schleife aus, mit der die von `dw.getBrowsersList()` wiedergegebene Browserliste überprüft wird. Stimmt ein Pfad zum Browser in der Liste mit dem Primär- oder Sekundärbrowser überein, setzt die Funktion die Variable `theBrowser` auf den entsprechenden Wert in `browserList`. Der Wert enthält den Browsernamen sowie den Pfad zur ausführbaren Datei, mit der der Browser gestartet wird. Wenn `havePreviewTarget()` den Wert `false` zurückgibt, setzt die Funktion die Variable `theBrowser` auf den Wert der Variable `whichBrowser`.

Anschließend überprüft die Funktion `receiveArguments()` die Variable `theBrowser`, um sicherzustellen, dass sie nicht mit einem Pfad beginnt, dass sie nicht "undefined" ist und dass sie eine Länge größer als Null aufweist. Wenn all diese Bedingungen erfüllt sind und der Fokus auf dem Bedienfeld liegt, ruft die Funktion `receiveArguments()` die Funktion `site.browseDocument()` auf, um den ausgewählten Browser mit den im Bedienfeld ausgewählten Dateien aufzurufen. Befindet sich der Fokus nicht auf dem Bedienfeld **Dateien**, ruft die Funktion `receiveArguments()` die Funktion `dw.browseDocument()` auf und übergibt ihr den Pfad des aktuellen Dokuments und den Wert der Variable `theBrowser`, die den Namen des Browsers angibt, mit dem das Dokument geöffnet werden soll.

Wenn der Benutzer die Tastaturbefehle (<F12> oder <Strg>+<F12>) verwendet und kein Primär- oder Sekundärbrowser angegeben wurde, wird der Benutzer in einem Dialogfeld darüber informiert. Wenn der Benutzer auf **OK** klickt, ruft die Funktion die Funktion `dw.showPreferencesDialog()` mit dem Argument `browsers` auf, damit der Benutzer zu diesem Zeitpunkt einen Browser angeben kann.

Die Funktionen der Menübefehls-API

Die benutzerdefinierten Funktionen der API für Menübefehle sind nicht erforderlich.

canAcceptCommand()

Beschreibung

Bestimmt, ob das Menüelement aktiv ist oder abgeblendet ist.

Argumente

`{arg1}, {arg2}, ... {argN}`

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige Kennung als einziges Argument übergeben. Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert an die Funktion `canAcceptCommand()` (sowie an die Funktionen „`isCommandChecked()`“ auf Seite 154, „`receiveArguments()`“ auf Seite 155 und „`setMenuText()`“ auf Seite 156) in Form ein oder mehrerer Argumente übergeben. Das Attribut `arguments` eignet sich zur Unterscheidung von zwei Menüelementen, die denselben Menübefehl aufrufen.

Hinweis: Das Attribut `arguments` wird bei dynamischen Menüelementen ignoriert.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn das Element aktiviert werden soll, und andernfalls `false`.

commandButtons()

Beschreibung

Definiert die Schaltflächen für die rechte Seite des Dialogfelds **Optionen** und deren Klickverhalten. Wenn diese Funktion nicht definiert ist, werden keine Schaltflächen angezeigt, und der `body`-Bereich der Menübefehlsdatei füllt das gesamte Dialogfeld aus.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array, in dem eine gerade Anzahl von Elementen gespeichert ist. Das erste Element ist ein String mit der Bezeichnung für die oberste Schaltfläche. Das zweite Element ist ein String mit JavaScript-Code, mit dem das Klickverhalten der obersten Schaltfläche bestimmt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel

Im folgenden Beispiel wird gezeigt, wie mithilfe der Funktion `commandButtons()` die Schaltflächen **OK**, **Abbrechen** und **Hilfe** definiert werden.

```
function commandButtons() {  
    return new Array("OK", "doCommand()", "Cancel", "  
    "window.close()", "Help", "showHelp()");  
}
```

getDynamicContent()

Beschreibung

Ruft den Inhalt des dynamischen Teils des Menüs ab.

Argumente

menuID

Das Argument *menuID* ist der Wert des Attributs *id* im Tag *menuitem*, das mit dem Element verknüpft ist.

Rückgabewerte

Dreamweaver erwartet ein String-Array, bei dem jeder String den Namen eines Menüelements und dessen eindeutige ID enthält. Die einzelnen Elemente sind jeweils durch einen Strichpunkt voneinander getrennt. Wenn die Funktion einen *null*-Wert zurückgibt, ändert sich das Menü nicht.

Beispiel

Im folgenden Beispiel gibt die Funktion `getDynamicContent()` ein Array mit vier Menüelementen zurück (My Menu Item 1, My Menu Item 2, My Menu Item 3 und My Menu Item 4):

```
function getDynamicContent() {
    var stringArray= new Array();
    var i=0;
    var numItems = 4;

    for (i=0; i<numItems;i++)
        stringArray[i] = new String("My Menu Item " + i + " ");
        id='My-MenuItem' + i + "";

    return stringArray;
}
```

isCommandChecked()

Beschreibung

Bestimmt, ob neben dem Menüelement ein Häkchen angezeigt werden soll.

Argumente

{arg1}, {arg2}, ... {argN}

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige Kennung als einziges Argument übergeben. Wenn das Attribut *arguments* für ein Tag des Typs *menuitem* definiert ist, wird der Attributwert an die Funktion `isCommandChecked()` (sowie an die Funktionen „`canAcceptCommand()`“ auf Seite 153, „`receiveArguments()`“ auf Seite 155 und „`setMenuText()`“ auf Seite 156) in Form ein oder mehrerer Argumente übergeben. Das Attribut *arguments* eignet sich zur Unterscheidung von zwei Menüelementen, die denselben Menübefehl aufrufen.

Hinweis: Das Attribut *arguments* wird bei dynamischen Menüelementen ignoriert.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. *true*, wenn neben dem Menüelement ein Häkchen angezeigt werden soll, und andernfalls *false*.

Beispiel

```
function isCommandChecked()
{
    var bChecked = false;
    var cssStyle = arguments[0];

    if (dw.getDocumentDOM() == null)
        return false;

    if (cssStyle == "(None)")
    {
        return dw.cssStylePalette.getSelectedStyle() == '';
    }
    else
    {
        return dw.cssStylePalette.getSelectedStyle() == cssStyle;
    }

    return bChecked;
}
```

receiveArguments()

Beschreibung

Verarbeitet Argumente, die von einem Menüelement oder von der Funktion `dw.runCommand()` werden. Bei dynamischen Menüeinträgen wird die Kennung des dynamischen Menüeintrags verarbeitet.

Argumente

`{arg1}, {arg2}, ... {argN}`

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige Kennung als einziges Argument übergeben. Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert an die Funktion `receiveArguments()` (sowie an die Funktionen „`canAcceptCommand()`“ auf Seite 153, „`isCommandChecked()`“ auf Seite 154 und „`setMenuText()`“ auf Seite 156) in Form ein oder mehrerer Argumente übergeben. Das Attribut `arguments` eignet sich zur Unterscheidung von zwei Menüelementen, die denselben Menübefehl aufrufen.

Hinweis: Das Attribut `arguments` wird bei dynamischen Menüelementen ignoriert.

Rückgabewerte

Dreamweaver erwartet keinen Wert.

Beispiel

```
function receiveArguments()
{
    var styleName = arguments[0];
    if (styleName == "(None)")
        dw.getDocumentDOM('document').applyCSSStyle('', '');
    else
        dw.getDocumentDOM('document').applyCSSStyle('', styleName);
}
```

setMenuText()

Beschreibung

Bestimmt, welcher Text im Menü angezeigt wird.

Hinweis: Setzen Sie diese Funktion nicht ein, wenn Sie „`getDynamicContent()`“ auf Seite 154 verwenden.

Argumente

`{arg1}, {arg2}, ... {argN}`

Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert an die Funktion `setMenuText()` (sowie an die Funktionen „`canAcceptCommand()`“ auf Seite 153, „`isCommandChecked()`“ auf Seite 154 und „`receiveArguments()`“ auf Seite 155) in Form ein oder mehrerer Argumente übergeben. Das Attribut `arguments` eignet sich zur Unterscheidung von zwei Menüelementen, die denselben Menübefehl aufrufen.

Rückgabewerte

Dreamweaver erwartet den String, der im Menü angezeigt werden soll.

Beispiel

```
function setMenuText()
{
    if (arguments.length != 1) return "";

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
        return dw.getUndoText();
    else if (whatToDo == "redo")
        return dw.getRedoText();
    else return "";
}
```

windowDimensions()

Beschreibung

Legt die Abmessungen für das Dialogfeld **Parameter** fest. Wenn diese Funktion nicht definiert ist, werden die Abmessungen automatisch berechnet.

Hinweis: Verwenden Sie diese Funktion nur dann, wenn ein Dialogfeld größer als 640 x 480 Pixel sein soll.

Argumente

`platform`

Der Wert des Arguments `platform` ist entweder „`macintosh`“ oder „`windows`“, abhängig von der verwendeten Plattform.

Rückgabewerte

Dreamweaver erwartet einen String der Form „`widthInPixels,heightInPixels`“.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da die Schaltflächen für **OK** und **Abbrechen** nicht mit einbezogen sind. Wenn mit den zurückgegebenen Abmessungen nicht für alle Optionen Platz vorhanden ist, werden Bildlaufleisten angezeigt.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Parameter-Dialogfelds auf 648 x 520 Pixel eingestellt:

```
function windowDimensions(){
    return "648,520";
}
```

Kapitel 11: Symbolleisten

Eine Symbolleiste für Adobe Dreamweaver CS3 können Sie erstellen, indem Sie einfach eine Datei erstellen, in der die Symbolleiste definiert wird, und diese Datei im Ordner **Configuration\Toolbars** ablegen. In eine Symbolleistendatei können Sie mithilfe einiger benutzerdefinierter XML-Tags Kontrollkästchen, Optionsschalter, Textfelder und Popupmenüs definieren. Um ihr Aussehen und Verhalten festzulegen, lassen sich Symbolleistenelementen Attribute und Befehle zuweisen, z. B. andere Symbolleistendateien und Referenz-Symbolleistenelemente, die in anderen Symbolleisten definiert sind.

In der folgenden Tabelle sind die Dateien zum Erstellen von Symbolleisten aufgeführt:

Pfad	Datei	Beschreibung
Configuration\Toolbars\	toolbars.xml	Bearbeiten Sie diese Datei, um den Inhalt der Symbolleiste zu ändern.
Configuration\Toolbars\	neue_Symbolleiste.xml	Diese Datei erstellen Sie zum Erstellen einer neuen Symbolleiste.
Configuration\Toolbars\	Bilddatei.gif	Bild für das Symbolleistensteuerelement.
Configuration\Commands\	MyCommand.htm	Mit dem Symbolleistenelement verknüpfte Befehlsdatei.

Funktionsweise von Symbolleisten

Symbolleisten werden durch XML- und Bilddateien definiert, die im Unterordner **Toolbars** des Hauptordners **Configuration** von Dreamweaver gespeichert sind. Die Standard-Symbolleisten von Dreamweaver sind im Ordner **Configuration\Toolbars\toolbars.xml** gespeichert. Wenn Dreamweaver gestartet wird, werden die Symbolleistendateien geladen, die im Ordner **Toolbars** abgelegt sind. Um neue Symbolleisten hinzuzufügen, brauchen Sie nicht die Hauptsymbolleistendatei zu ändern, sondern Sie können einfach eine Datei in den Ordner **Toolbars** kopieren.

Symbolleisten-XML-Dateien definieren eine oder mehrere Symbolleisten und deren Elemente. Eine Symbolleiste ist eine Liste von Elementen, z. B. Schaltflächen, Textfelder, Popupmenüs usw. Ein Symbolleistenelement stellt ein einzelnes Steuerelement dar, auf das die Benutzer in einer Symbolleiste zugreifen können.

Mit einigen Typen von Symbolleistenelementen, z. B. Druckschaltflächen und Popupmenüs, sind Symbolbilder verknüpft. Symbolbilder werden unter dem Ordner **Toolbars** im Ordner **images** gespeichert. Die Bilder können jedes Format besitzen, das Dreamweaver darstellen kann, sind normalerweise aber GIF- oder JPEG-Dateien. Bilder für Symbolleisten, die von Adobe erstellt wurden, sind im Ordner **Toolbars\images\MM** gespeichert.

Wie bei Menüs können Sie die Funktionalität einzelner Symbolleistenelemente entweder über die Elementattribute oder eine Befehlsdatei festlegen. Von Adobe bereitgestellte Befehlsdateien sind im Ordner **Toolbars\MM** gespeichert.



Da die API für die Symbolleiste kompatibel ist mit der API für Menübefehle, können Symbolleisten-Steuerelemente die Menübefehlsdateien wiederverwenden.

Im Gegensatz zu Menüs lassen sich Symbolleistenelemente unabhängig von den Symbolleisten definieren, die sie verwenden. Durch diese Flexibilität können Sie Symbolleistenelemente mithilfe des `itemref`-Tags in mehreren Symbolleisten verwenden.

Wenn Dreamweaver eine Symbolleiste zum ersten Mal lädt, werden ihre Sichtbarkeit und Position durch die Symbolleistendefinition festgelegt. Danach werden Sichtbarkeit und Position in der Registrierung (Windows) bzw. in der Voreinstellungsdatei von Dreamweaver (Macintosh) gespeichert und von dort aus wiederhergestellt.

Verhaltensweisen von Symbolleisten

In Windows verhalten sich Dreamweaver-Symbolleisten im Allgemeinen genau wie Windows-Symbolleisten. Dreamweaver-Symbolleisten besitzen folgende Kennzeichen:

- Sie können Symbolleisten durch Ziehen und Ablegen andocken, abdocken und ihre Position relativ zu anderen Symbolleisten ändern.
- Sie können Symbolleisten horizontal an den oberen oder unteren Rand des Frame-Fensters andocken.

Im Dreamweaver-Arbeitsbereich, in dem alle Dreamweaver-Dokumentfenster in einem einzigen übergeordneten Fenster integriert sind, können Sie festlegen, ob Symbolleisten am Arbeitsbereichsfenster oder am Dokumentfenster andocken.

Von Symbolleisten, die am Dreamweaver-Arbeitsbereichsfenster andocken, existiert jeweils nur eine Instanz. In diesem Fall gelten die Symbolleisten immer für das Dokument im Vordergrund. Im Dreamweaver-Arbeitsbereich können Sie Symbolleisten über, unter, links oder rechts der Symbolleiste **Einfügen** andocken. Symbolleisten, die mit dem Dreamweaver-Arbeitsbereichsfenster verknüpft sind, werden nicht automatisch deaktiviert, wenn kein Dokumentfenster vorhanden ist. Die Symbolleistenelemente bestimmen, ob sie aktiviert sind, wenn kein Dokument geöffnet ist.

Wenn Symbolleisten immer an das Dokumentfenster andockt, existiert für jedes Fenster eine Symbolleiste. Symbolleisten, die mit einem Dokumentfenster verknüpft sind, deaktivieren sich selbständig, wenn ihr Fenster nicht mehr im Vordergrund liegt, und führen ihre Aktualisierungsprozeduren erneut aus, wenn ihr Fenster wieder in den Vordergrund tritt.

Symbolleisten können nicht durch Ziehen und Ablegen zwischen dem Dokumentfenster und dem Dreamweaver-Arbeitsbereichsfenster verschoben werden.

- Symbolleisten besitzen immer eine feste Größe. Eine Symbolleiste wird nicht verkleinert, wenn ihr Container verkleinert wird oder andere Symbolleisten neben ihr platziert werden.
- Sie können mithilfe der Befehle **Ansicht > Symbolleisten** Symbolleisten ein- oder ausblenden.
- Symbolleisten können sich nicht überlappen.
- Wenn Sie eine Symbolleiste ziehen, wird nur ihr Rahmen angezeigt.

Auf Macintosh-Rechnern sind Symbolleisten immer mit dem Dokumentfenster verknüpft. Sie lassen sich im Menü ein- und ausblenden, nicht jedoch ziehen und ablegen, neu anordnen oder abdocken.

Funktionsweise von Befehlen

Wenn Dreamweaver eine Symbolleiste darstellt, geschieht Folgendes:

- 1 Für jedes Symbolleistenelement bestimmt Dreamweaver, ob das Attribut `file` existiert.
- 2 Falls das Attribut `file` existiert, ruft Dreamweaver `canAcceptCommand()` auf, um zu bestimmen, ob das Steuerelement im aktuellen Kontext des Dokuments aktiviert werden soll.

Für das Textfeld **Dokumenttitel** der Dreamweaver-Symbolleiste prüft `canAcceptCommand()` beispielsweise, ob ein aktuelles DOM (Document Object Model) existiert und das aktuelle Dokument eine HTML-Datei ist. Falls beide Bedingungen erfüllt sind, gibt die Funktion `true` zurück, und Dreamweaver aktiviert das Textfeld in der Symbolleiste.

- 3 Wenn das Attribut `file` vorhanden ist, ignoriert Dreamweaver die folgenden Attribute, sofern diese definiert sind: `checked`, `command`, `DOMRequired`, `enabled`, `script`, `showif`, `update` und `value`.

- 4 Wenn das Attribut `file` nicht existiert, verarbeitet Dreamweaver die Attribute, die für das Symbolleisten-Steuerelement festgelegt sind: `checked`, `command`, `DomRequired` usw.

Weitere Informationen zu bestimmten Element-Tag-Attributen finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

- 5 Dreamweaver ruft die Funktion `getCurrentValue()` bei jedem Aktualisierungszyklus auf, wie durch das Attribut `update` festgelegt, um den anzuzeigenden Wert für das Steuerelement zu bestimmen.
- 6 Der Benutzer wählt ein Element in der Symbolleiste aus.
- 7 Dreamweaver ruft die Funktion `receiveArguments()` auf, um alle Argumente zu verarbeiten, die durch das Attribut `arguments` des Symbolleistenelements festgelegt sind.

Weitere Informationen über den Zweck bestimmter Funktionen in der API für Symbolleistenbefehle finden Sie unter „Die API-Funktionen für Symbolleistenbefehle“ auf Seite 174.

Eine einfache Symbolleisten-Befehlsdatei

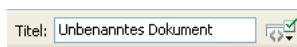
Bei diesem einfachen Beispiel wird ein Element des Felds „Titel“ wie bei der Dokument-Symbolleiste von Dreamweaver implementiert. Im Textfeldelement kann der Benutzer einen Namen für das aktuelle Dreamweaver-Dokument eingeben. Sie können dieses Symbolleistenbeispiel anhand der folgenden Schritte nachvollziehen:

- „Textfelder erstellen“ auf Seite 159
- „JavaScript-Code schreiben“ auf Seite 159

Textfelder erstellen

Sie können Dreamweaver eine Symbolleiste hinzufügen, indem Sie eine XML-Datei mit der Symbolleistendefinition im Ordner **Toolbars** ablegen, der sich im Dreamweaver-Ordner **Configuration** befindet.

Diese Abbildung stellt das Textbearbeitungsfeld **Titel** dar:



Das folgende `editcontrol`-Element der Symbolleiste definiert ein Textbearbeitungsfeld mit der Bezeichnung „Titel“:

```
<EDITCONTROL ID="DW_SetTitle"
  label="Title: "
  tooltip="Document Title"
  width="150"
  file="Toolbars/MM/EditTitle.htm"/>
```

Das Attribut `tooltip` bewirkt, dass Dreamweaver den Dokumenttitel als QuickInfo anzeigt, wenn der Benutzer den Mauszeiger auf dem Textfeld platziert. Das Attribut `width` definiert die Größe des Feldes in Pixel. Das Attribut `file` legt fest, dass die Datei `EditTitle.htm` die JavaScript-Funktionen enthält, die auf das Textfeld angewendet werden. Eine vollständige Definition der Symbolleiste des Dreamweaver-Dokuments ist in der Datei **toolbars.xml** in der Hauptsymbolleiste (`id="DW_Toolbar_Main"`) enthalten.

JavaScript-Code schreiben

Wenn der Benutzer mit dem Textfeld interagiert, ruft Dreamweaver die Befehlsdatei `EditTitle.htm` auf, die im Ordner **Toolbars\MM** abgelegt ist. Diese Datei beinhaltet drei JavaScript-Funktionen, die auf das Textfeld **Titel** angewendet werden können. Dies sind `canAcceptCommand()`, `receiveArguments()` und `getCurrentValue()`.

canAcceptCommand(): aktiviert das Symbolleistenelement

Die Funktion `canAcceptCommand()` besteht aus einer Codezeile, mit der überprüft wird, ob ein aktuelles Dokumentobjektmodell (DOM) vorhanden ist und ob das Dokument als HTML analysiert wird. Die Funktion gibt die Ergebnisse dieser Tests zurück. Wenn die Bedingungen den Wert `true` ergeben, aktiviert Dreamweaver das Textfeldelement auf der Symbolleiste. Wenn die Funktion den Wert `false` zurückgibt, deaktiviert Dreamweaver das Element.

Die Funktion lautet:

```
function canAcceptCommand()
{
  return (dw.getDocumentDOM() != null && dw.getDocumentDOM().getParseMode() == 'html');
}
```

receiveArguments(): legt den Titel fest

Dreamweaver ruft die im vorangegangenen Beispiel dargestellte Funktion `receiveArguments()` auf, wenn der Benutzer einen Wert in das Textfeld **Titel** eingibt und die Eingabetaste drückt oder den Fokus vom Steuerelement weg verlagert.

Die Funktion lautet:

```
function receiveArguments(newTitle)
{
    var dom = dw.getDocumentDOM();
    if (dom)
        dom.setTitle(newTitle);
}
```

Dreamweaver übergibt `newTitle`, d. h. den Wert, den der Benutzer eingibt, an die Funktion `receiveArguments()`. Die Funktion `receiveArguments()` überprüft zunächst, ob ein aktuelles DOM existiert. Ist dies der Fall, legt die Funktion `receiveArguments()` den neuen Dokumenttitel fest, indem sie `newTitle` an die Funktion `dom.setTitle()` übergibt.

getCurrentValue(): Titel abrufen

Bei jedem Aktualisierungszyklus – wie durch die standardmäßige Aktualisierungshäufigkeit der Ereignisprozedur `onEdit` festgelegt – ruft Dreamweaver die Funktion `getCurrentValue()` auf, um festzulegen, welcher Wert für das Steuerelement angezeigt werden soll. Die standardmäßige Aktualisierungshäufigkeit der Prozedur `onEdit` legt die Aktualisierungshäufigkeit fest, da das Textbearbeitungsfeld **Titel** über kein `update`-Attribut verfügt.

Die folgende Funktion `getCurrentValue()` ruft für das Textfeld **Titel** die JavaScript-API-Funktion `dom.getTitle()` auf, um den aktuellen Titel abzurufen und zurückzugeben.

Die Funktion lautet:

```
function getCurrentValue()
{
    var title = "";
    var dom = dw.getDocumentDOM();
    if (dom)
        title = dom.getTitle();
    return title;
}
```

Solange der Benutzer keinen Titel für das Dokument eingibt, gibt die Funktion `getTitle()` den Wert „Unbenanntes Dokument“ zurück, der im Textfeld angezeigt wird. Nachdem der Benutzer einen Titel eingegeben hat, gibt die Funktion `getTitle()` diesen Wert zurück, und Dreamweaver zeigt diesen Wert als neuen Dokumenttitel an.

Die vollständige HTML-Datei mit den JavaScript-Funktionen für das Textfeld **Titel** ist in der Datei **EditTitle.htm** enthalten, die sich im Ordner **Toolbars\MM** befindet.

Der Ordner **MM** ist für Adobe-Dateien reserviert. Sie müssen daher einen anderen Ordner im Ordner **Toolbars** erstellen und Ihren JavaScript-Code in diesem Ordner ablegen.

Die Symbolleisten-Definitionsdatei

Bei einer Symbolleiste handelt es sich um eine Reihe von Optionsschaltern, Kontrollkästchen, Bearbeitungsfeldern und anderen Symbolleistenelementen, die optional durch `separator`-Tags getrennt sind. Jedes Symbolleistenelement kann entweder eine Referenz auf ein Element sein, das das Tag `itemref` verwendet, eine Trennlinie, die das Tag `separator` verwendet, oder eine vollständige Symbolleistenelement-Definition für ein Kontrollkästchen oder ein Bearbeitungsfeld, wie unter „Symbolleistenelement-Tags“ auf Seite 165 beschrieben.

Jede Symbolleisten-Definitionsdatei beginnt mit den folgenden Deklarationen:

```
<?xml version="1.0" encoding="optional_encoding"?>
<!DOCTYPE toolbarset SYSTEM "-//Macromedia//DWEExtension toolbar 5.0">
```

Wenn die Kodierung weggelassen wird, verwendet Dreamweaver die Standardkodierung des Betriebssystems.

Gemäß den Deklarationen besteht die Datei aus einem einzigen `toolbarset`-Tag. Dieses wiederum umfasst eine beliebige Anzahl der folgenden Tags: `toolbar`, `itemref`, `separator`, `include` und `itemtype`, wobei `itemtype` ein `button`, `checkbox`, `radiobutton`, `menubutton`, `dropdown`, `combobox`, `editcontrol` oder `colorpicker` ist. Im folgenden Beispiel, einem ungekürzten Auszug aus der Datei **toolbars.xml**, wird die Hierarchie von Tags in der Symbolleistendatei veranschaulicht. Im Beispiel werden Auslassungszeichen (...) in denjenigen Attributen von Symbolleistenelementen ersetzt, die in den folgenden Abschnitten beschrieben werden.

```
<?xml version="1.0"?>
<!DOCTYPE toolbarset SYSTEM "-//Macromedia//DWExtension toolbar 5.0">
<toolbarset>

<!-- Hauptsymboleiste -->
  <toolbar id="DW_Toolbar_Main" label="Document">
    <radiobutton id="DW_CodeView" . . ./>
    <radiobutton id="DW_SplitView" . . ./>
    <radiobutton id="DW_DesignView" . . ./>
    <separator/>
    <checkbox id="DW_LiveDebug" . . ./>
    <checkbox id="DW_LiveDataView" . . ./>
    <separator/>
    <editcontrol id="DW_SetTitle" . . ./>
    <menubutton id="DW_FileTransfer" . . ./>
    <menubutton id="DW_Preview" . . ./>
    <separator/>
    <button id="DW_DocRefresh" . . ./>
    <button id="DW_Reference" . . ./>
    <menubutton id="DW_CodeNav" . . ./>
    <menubutton id="DW_ViewOptions" . . ./>
  </toolbar>
</toolbarset>
```

Im folgenden Abschnitt werden die einzelnen Symbolleisten-Tags beschrieben.

<Symbolleiste>

Beschreibung

Definiert eine Symbolleiste. Dreamweaver zeigt die Elemente und Trennlinien von links nach rechts in der angegebenen Reihenfolge an und ordnet die Elemente automatisch an. Die Symbolleistendatei steuert nicht den Abstand zwischen den Elementen, doch können Sie die Breite einiger Elementtypen festlegen.

Attribute

`id`, `label`, `{container}`, `{initiallyVisible}`, `{initialPosition}`, `{relativeTo}`

- `id="unique_id"` Erforderlich. Ein Bezeichner-String muss innerhalb einer Datei und sämtlichen Dateien, die diese Datei beinhalten, eindeutig sein. Die JavaScript-API-Funktionen, die eine Symbolleiste manipulieren, verweisen durch deren ID darauf. Weitere Informationen zu diesen Funktionen finden Sie im *Dreamweaver API-Referenzhandbuch*. Wenn zwei in derselben Datei enthaltene Symbolleisten dieselbe ID besitzen, zeigt Dreamweaver eine Fehler an.
- `label="string"` Erforderlich. Das `label`-Attribut definiert die Bezeichnung, die Dreamweaver dem Benutzer als Zeichenfolge anzeigt. Die Bezeichnung wird im Menü **Ansicht > Symbolleisten** und in der Titelleiste der Symbolleiste angezeigt, wenn diese abgedockt ist.
- `container="mainframe"` oder `"document"` Standard: `"mainframe"`. Gibt an, wo die Symbolleiste im Dreamweaver-Arbeitsbereich unter Windows andockt ist. Falls der Container als `"mainframe"` festgelegt ist, wird die Symbolleiste im äußeren Arbeitsbereichsfenster angezeigt und gilt für das Dokument im Vordergrund. Wenn der Container auf `"document"` gesetzt ist, wird die Symbolleiste in jedem Dokumentfenster angezeigt. Bei einem Macintosh-Computer werden alle Symbolleisten in jedem Dokumentfenster angezeigt.

- `initiallyVisible="true"` oder `"false"`. Dieses Tag gibt an, ob die Symbolleiste eingeblendet wird, wenn Dreamweaver sie zum ersten Mal aus dem Ordner **Toolbars** lädt. Nach dem ersten Laden überprüft der Benutzer die Sichtbarkeit. Dreamweaver speichert den aktuellen Status der Systemregistrierung (Windows) oder der Dreamweaver -Voreinstellungsdatei (Macintosh), wenn der Benutzer Dreamweaver beendet. Dreamweaver stellt die Einstellungen der Registrierung bzw. der Voreinstellungsdatei wieder her, wenn das Programm wieder gestartet wird. Das Ein- und Ausblenden der Symbolleiste können Sie mit den Funktionen `dom.getToolbarVisibility()` und `dom.setToolbarVisibility()` einstellen, wie im *Dreamweaver API-Referenzhandbuch* beschrieben. Falls Sie das Attribut `initiallyVisible` nicht festlegen, wird standardmäßig `true` verwendet.
- `initialPosition="top", "below" oder "floating"`. Gibt an, wo Dreamweaver anfangs die Symbolleiste relativ zu anderen Symbolleisten positioniert, wenn Dreamweaver sie zum ersten Mal lädt. Die möglichen Werte für `initialPosition` sind in der folgenden Liste beschrieben:
 - `top` Dies ist die Standardposition. Die Symbolleiste wird oben im Dokumentfenster angezeigt. Wenn `top` bei mehreren Symbolleisten für einen bestimmten Fenstertyp festgelegt ist, werden die Symbolleisten in der Reihenfolge angezeigt, in der Dreamweaver sie lädt. Falls die Symbolleisten sich in unterschiedlichen Dateien befinden, ist diese Reihenfolge nicht immer vorhersehbar.
 - `below` Die Symbolleiste wird am Anfang einer Zeile unterhalb der Symbolleiste angezeigt, die im Attribut `relativeTo` festgelegt ist. Dreamweaver gibt einen Fehler aus, wenn die Symbolleiste `relativeTo` nicht gefunden werden konnte. Wenn bei mehreren Symbolleisten `below` relativ zu derselben Symbolleiste festgelegt ist, werden die Symbolleisten in der Reihenfolge angezeigt, in der Dreamweaver sie lädt. Falls die Symbolleisten sich in unterschiedlichen Dateien befinden, ist diese Reihenfolge nicht immer vorhersehbar.
 - `floating` Die Symbolleiste ist anfangs nicht am Fenster angedockt, sondern schwebt über dem Dokument. Dreamweaver fügt die Symbolleiste automatisch versetzt zu anderen schwebenden Symbolleisten an. Auf dem Macintosh wird `floating` gleichbedeutend mit `top` behandelt.

Wie das Attribut `initiallyVisible` wird das Attribut `initialPosition` nur angewandt, wenn Dreamweaver die Symbolleiste zum ersten Mal lädt. Danach wird die Position der Symbolleiste in der Registrierung bzw. in der Voreinstellungsdatei von Dreamweaver gespeichert. Mithilfe der Funktion `dom.setToolbarPosition()` lässt sich die Position der Symbolleiste zurücksetzen. Weitere Informationen über die Funktion `dom.setToolbarPosition()` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Wenn Sie das Attribut `initialPosition` nicht angegeben haben, positioniert Dreamweaver die Symbolleiste entsprechend der beim Laden auftretenden Reihenfolge.

- `relativeTo="toolbar_id"` Dieses Attribut ist erforderlich, wenn bei dem Attribut `initialPosition` `below` festgelegt ist. Andernfalls wird der Parameter ignoriert. Gibt die ID der Symbolleiste an, unterhalb der diese Symbolleiste positioniert werden soll.

Inhalt

Das Tag `toolbar` beinhaltet die Tags `include`, `itemref` und `separator` sowie einzelne Elementdefinitionen wie z. B. `button`, `combobox`, `dropdown` usw. Beschreibungen der Elementdefinitionen, die Sie angeben können, finden Sie unter „Symbolleistenelement-Tags“ auf Seite 165.

Container

Das Tag `toolbarset`.

Beispiel

```
<toolbar id="MyDWedit_toolbar" label="Edit">
```


<include/>

Beschreibung

Lädt Symbolleistenelemente aus der angegebenen Datei, bevor mit dem Laden der aktuellen Datei fortgefahren wird. Symbolleistenelemente, die in der unter <include> angegebenen Datei definiert sind, können in der aktuellen Datei referenziert werden. Wenn eine Datei versucht, eine andere Datei rekursiv aufzunehmen, zeigt Dreamweaver eine Fehlermeldung an und ignoriert das rekursive <include>. Alle `toolbar`-Tags in der unter <include> angegebenen Datei werden übersprungen, jedoch können Symbolleistenelemente dieser Symbolleisten in der aktuellen Datei referenziert werden.

Attribute

- Der Pfad `file` der XML-Datei, die aufgenommen werden soll, relativ zum Ordner **Toolbars**.

Inhalt

Keine

Container

`toolbar`-Tag oder `toolbarset`-Tag

Beispiel

```
<include file="mine/editbar.xml"/>
```

<itemtype/>

Beschreibung

Definiert ein einzelnes Symbolleistenelement. Zu Symbolleistenelementen zählen Schaltflächen, Optionsschalter, Kontrollkästchen, Kombinationsfelder, Popupmenüs usw. Eine Liste der Symbolleistenelemente, die Sie definieren können, finden Sie unter „Symbolleistenelement-Tags“ auf Seite 165.

Attribute

Die Attribute variieren abhängig von dem Element, das Sie definieren. Eine vollständige Liste der Attribute, die Sie für Symbolleistenelemente definieren können, finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

`toolbar`-Tag oder `toolbarset`-Tag

Beispiel

```
<button id="strikeout_button" .../>
```

<itemref/>

Beschreibung

Verweist auf ein Symbolleistenelement (und fügt es in die aktuelle Symbolleiste ein), das innerhalb einer vorherigen Symbolleiste oder außerhalb aller Symbolleisten definiert wurde.

Attribute`id, {showIf}`

- `id="id_reference"` Erforderlich. Muss die ID eines Elements sein, das vorher definiert oder in die Datei aufgenommen wurde. Dreamweaver lässt keine vorwärts gerichteten Verweise zu. Falls ein Symbolleistenelement-Tag auf eine nicht definierte ID verweist, gibt Dreamweaver einen Fehler aus und ignoriert den Verweis.
- `showIf="script"` Legt fest, dass dieses Element nur dann in der Symbolleiste angezeigt wird, wenn das angegebene Skript den Wert `true` zurückgibt. Beispielsweise können Sie mit `showIf` bestimmte Schaltflächen nur in einer bestimmten Anwendung einblenden lassen oder nur dann, wenn eine Seite in einer serverbasierten Sprache, z. B. ColdFusion, ASP oder JSP verfasst ist. Falls Sie `showIf` nicht angeben, wird das Element immer angezeigt. Dreamweaver prüft diese Eigenschaft immer, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig von dem Wert des Attributs `update`. Dieses Attribut sollte sparsam verwendet werden. Das Attribut kann entweder in der Elementdefinition oder in einem Verweis auf das Element von einer Symbolleiste aus verwendet werden. Wenn das Attribut `showIf` sowohl in der Definition als auch in dem Verweis angegeben ist, zeigt Dreamweaver das Element nur dann an, wenn beide Bedingungen erfüllt sind. Das Attribut `showIf` entspricht der Funktion `showIf()` in einer Befehlsdatei.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel`<itemref id="strikeout_button">`**<separator/>****Beschreibung**

Fügt an der aktuellen Position eine Trennlinie in die Symbolleiste ein.

Attribute`{showIf}`

- Das Attribut `showIf` legt fest, dass die Trennlinie nur dann in der Symbolleiste angezeigt wird, wenn das angegebene Skript den Wert `true` aufweist. Beispielsweise können Sie mit dem Attribut `showIf` die Trennlinie nur in einer bestimmten Anwendung einblenden lassen oder nur dann, wenn die Seite einen bestimmten Dokumenttyp besitzt. Wenn das Attribut `showIf` nicht definiert ist, wird die Trennlinie immer angezeigt.

Inhalt

Keiner

ContainerDas Tag `toolbar`.**Beispiel**`<separator/>`

Symbolleistenelement-Tags

Jeder Typ von Symbolleistenelementen besitzt ein eigenes Tag und eine eigene Gruppe erforderlicher oder optionaler Attribute. Sie können Symbolleisten elemente außerhalb oder innerhalb von Symbolleisten definieren. Normalerweise ist es besser, sie außerhalb von Symbolleisten zu definieren und innerhalb von Symbolleisten mithilfe des Tags `itemref` auf sie zu verweisen.

Die folgenden Elementtypen lassen sich in Symbolleisten definieren.

<button>

Beschreibung

Wenn Sie auf diese Schaltfläche klicken, wird ein spezieller Befehl ausgeführt. Sie sieht wie die Schaltfläche **Referenz** in der Dreamweaver-Symbolleiste aus und verhält sich auch so.

Attribute

`id`, `image`, `tooltip`, `command`, `{showIf}`, `{disabledImage}`, `{overImage}`, `{label}`, `{file}`, `{domRequired}`, `{enabled}`, `{update}`, `{arguments}`

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<BUTTON ID="DW_DocRefresh"
  image="Toolbars/images/MM/refresh.gif"
  disabledImage="Toolbars/images/MM/refresh_dis.gif"
  tooltip="Refresh Design View (F5)"
  enabled="((dw.getDocumentDOM() != null) && (dw.getDocumentDOM().getView() != 'browse')
    && (!dw.getDocumentDOM().isDesignViewUpdated()))"
  command="dw.getDocumentDOM().synchronizeDocument()"
  update="onViewChange,onCodeViewSyncChange"/>
```

<checkboxbutton>

Beschreibung

Ein Kontrollkästchen kann einen aktivierten oder deaktivierten Zustand aufweisen und einen bestimmten Befehl ausführen, wenn es aktiviert wird. Wenn es aktiviert ist, wird es gedrückt und hervorgehoben dargestellt. Ist es deaktiviert, wird es flach dargestellt. In Dreamweaver existieren die folgenden Statusoptionen für ein Kontrollkästchen: Mauszeiger über dem Kontrollkästchen, gedrücktes Kontrollkästchen, Mauszeiger über dem gedrückten Kontrollkästchen und deaktiviertes gedrücktes Kontrollkästchen. Die Verarbeitungsprozedur, die durch das Attribut `checked` oder die Funktion `isCommandChecked()` festgelegt wird, muss sicherstellen, dass der Status der Schaltfläche durch Klicken auf das Kontrollkästchen umgeschaltet wird.

Attributes

`id`, `{showIf}`, `image`, `{disabledImage}`, `{overImage}`, `tooltip`, `{label}`, `{file}`, `{domRequired}`, `{enabled}`, `checked`, `{update}`, `command`, `{arguments}`

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<CHECKBUTTON ID="DW_LiveDebug"
  image="Toolbars/images/MM/debugview.gif"
  disabledImage="Toolbars/images/MM/globe_dis.gif"
  tooltip="Live Debug"
  enabled="dw.canLiveDebug()"
  checked="dw.getDocumentDOM() != null && dw.getDocumentDOM().getView() == 'browse'"
  command="dw.toggleLiveDebug()"
  showIf="dw.canLiveDebug()"
  update="onViewChange"/>
```

<radiobutton>

Beschreibung

Ein Optionsschalter entspricht einem Kontrollkästchen mit der Ausnahme, dass er erhoben dargestellt wird, wenn er ausgeschaltet ist. In Dreamweaver existieren die folgenden Statusoptionen für einen Optionsschalter: Mauszeiger über dem Kontrollkästchen, gedrücktes Kontrollkästchen, Mauszeiger über dem gedrückten Kontrollkästchen und deaktiviertes gedrücktes Kontrollkästchen. In Dreamweaver müssen Optionsschalter sich nicht gegenseitig ausschließen. Die Verarbeitungsprozedur, die durch das Attribut `checked` oder die Funktion `isCommandChecked()` festgelegt wird, muss sicherstellen, dass die aktivierten und deaktivierten Statusoptionen von Optionsschaltern konsistent sind.

Optionsschalter verhalten sich wie die Schaltflächen für Codeansicht, Entwurfsansicht und geteilte Ansicht in der Dreamweaver-Dokumentsymbolleiste.

Attribute

`id`, `image`, `tooltip`, `checked`, `command`, `{showIf}`, `{disabledImage}`, `{overImage}`, `{label}`, `{file}`, `{domRequired}`, `{enabled}`, `{update}`, `{arguments}`

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<RADIOBUTTON ID="DW_CodeView"
  image="Toolbars/images/MM/codeView.gif"
  disabledImage="Toolbars/images/MM/codeView_dis.gif"
  tooltip="Show Code View"
  domRequired="false"
  enabled="dw.getDocumentDOM() != null"
  checked="dw.getDocumentDOM() != null && dw.getDocumentDOM().getView() == 'code'"
  command="dw.getDocumentDOM().setView('code')"
  update="onViewChange"/>
```

<menubutton>

Beschreibung

Mit einer Menüschaltfläche wird das Kontextmenü eingeblendet, das durch das Attribut `menuid` festgelegt ist. In Dreamweaver existieren für Menüschaltflächen die Statusoptionen „Mauszeiger über dem Kontrollkästchen“ und „gedrücktes Kontrollkästchen“. Dreamweaver zeichnet nicht den Menüpfel. Bei diesem Pfeil handelt es sich um den Abwärts-Pfeil, der darauf hinweist, dass zu der Schaltfläche noch weitere Menüelemente gehören. Dieser Pfeil muss in Ihrem Symbol enthalten sein. Beispiele für Menüschaltflächen auf der Dokumentsymbolleiste von Dreamweaver sind die Schaltflächen **Dateiverwaltung** und **Code-Verwaltung**.

Attribute

id, image, tooltip, menuID, domRequired, enabled, {showIf}, {disabledImage}, {overImage}, {label}, {file}, {update}

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<MENUBUTTON ID="DW_CodeNav"
  image="Toolbars/images/MM/codenav.gif"
  disabledImage="Toolbars/images/MM/codenav_dis.gif"
  tooltip="Code Navigation"
  enabled="dw.getFocus() == 'textView' || dw.getFocus() == 'html'"
  menuID="DWCodeNavPopup"
  update="onViewChange"/>
```

<dropdown>**Beschreibung**

Ein Dropdownmenü (oder Popupmenü) ist ein nicht bearbeitbares Menü, das einen bestimmten Befehl ausführt, wenn Sie einen Eintrag auswählen, und das sich basierend auf einer verknüpften JavaScript-Funktion selbst aktualisiert. Das Dropdownmenü sieht aus und verhält sich wie das Format-Steuerelement im Eigenschafteninspektor für Text, mit der Ausnahme, dass es eine Standardgröße besitzt und damit größer als im Eigenschafteninspektor dargestellt wird.

Attribute

id, tooltip, file, enabled, checked, value, command, {showIf}, {label}, {width}, {domRequired}, {update}, {arguments}

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<dropdown id="Font_Example"
  width="115"
  tooltip="Font"
  domRequired="false"
  file="Toolbars/mine/fontExample.htm"
  update="onSelChange"/>
```

<combobox>**Beschreibung**

Ein Kombinationsfeld ist ein bearbeitbares Popupmenü, das seinen Befehl ausführt, wenn Sie einen Eintrag wählen oder der Benutzer das Textfeld bearbeitet und den Fokus wechselt. Das Menü sieht aus und verhält sich wie das Schriftarten-Steuerelement im Eigenschafteninspektor für Text, mit der Ausnahme, dass es eine Standardgröße besitzt und damit größer als im Eigenschafteninspektor dargestellt wird.

Attribute

id, file, tooltip, enabled, value, command, {showIf}, {label}, {width}, {domRequired}, {update}, {arguments}

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<COMBOBOX ID="Address_URL"
  width="300"
  tooltip="Address"
  label="Address: "
  file="Toolbars/MM/AddressURL.htm"
  update="onBrowserPageBusyChange"/>
```

<editcontrol>**Beschreibung**

Ein Bearbeitungsfeld ist ein Feld zur Bearbeitung von Text, das seinen Befehl ausführt, wenn der Benutzer den Text im Feld ändert und den Fokus wechselt.

Attribute

id, tooltip, file, value, command, {showIf}, {label}, {width}, {domRequired}, {enabled}, {update}, {arguments}

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<EDITCONTROL ID="DW_SetTitle"
  label="Title: "
  tooltip="Document Title"
  width="150"
  file="Toolbars/MM/EditTitle.htm"/>
```

<colorpicker>**Beschreibung**

Ein Farbwähler ist eine Farbpalette ohne verknüpftes Textfeld, die ihren Befehl ausführt, wenn der Benutzer eine neue Farbe auswählt. Diese Palette sieht aus und verhält sich wie der Farbwähler im Dreamweaver-Eigenschafteninspektor. Das Standardsymbol lässt sich durch ein anderes Symbol ersetzen.

Attribute

id, tooltip, value, command, {showIf}, {image}, {disabledImage}, {overImage}, {label}, {colorRect}, {file}, {domRequired}, {enabled}, {update}, {arguments}

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 169.

Inhalt

Keiner

Container

toolbar-Tag oder toolbarset-Tag

Beispiel

```
<colorpicker id="Color_Example"
  image="Toolbars/images/colorpickerIcon.gif"
  disabledImage="Toolbars/images/colorpickerIconD.gif"
  colorRect="0 12 16 16"
  tooltip="Text Color"
  domRequired="false"
  file="Toolbars/mine/colorExample.htm"
  update="onSelChange"/>
```

Attribute für Symbolleistenelement-Tags

Die Attribute für Symbolleistenelement-Tags besitzen folgende Bedeutung:

id="eindeutige_Id"

Erforderlich. Das Attribut `id` ist ein Bezeichner für das Symbolleistenelement. Das Attribut `id` muss innerhalb der aktuellen Datei und aller Dateien, die in die aktuelle Datei aufgenommen sind, eindeutig sein. Das Tag `itemref` verweist mithilfe des Tags `id` auf ein Element einer Symbolleiste und fügt es in die Symbolleiste ein.

Beispiel

```
<button id="DW_DocRerefresh" . . . >
```

showIf="Skript"

Optional. Dieses Attribut legt fest, dass das Element nur dann in der Symbolleiste angezeigt wird, wenn das Skript den Wert `true` zurückgibt. Beispielsweise können Sie mit dem Attribut `showIf` bestimmte Schaltflächen nur dann einblenden lassen, wenn eine Seite in einer serverbasierten Sprache verfasst ist, z. B. in ColdFusion, ASP oder JSP. Falls Sie `showIf` nicht angeben, wird das Element immer eingeblendet.

Das Attribut `showIf` wird immer überprüft, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig von dem Wert des Attributs `update`. Das Attribut `showIf` sollte sparsam verwendet werden.

Sie können das Attribut `showIf` in der Elementdefinition angeben und in einer Referenz auf das Element in einem `itemref`-Tag. Wenn das Attribut `showIf` sowohl in der Definition als auch in dem Verweis angegeben ist, wird das Element nur dann angezeigt, wenn beide Bedingungen erfüllt sind. Das Attribut `showIf` entspricht der Funktion `showIf()` in einer Symbolleisten-Befehlsdatei. Wenn Sie sowohl das Attribut `showIf` als auch die Funktion `showif()` angeben, setzt `showIf()` das Attribut außer Kraft.

Beispiel

```
showIf="dw.canLiveDebug()" "
```

image="Bildpfad"

Dieses Attribut ist erforderlich für Schaltflächen, Kontrollkästchen, Optionsschalter, Menüs Schaltflächen und Kombinationsfelder. Das Attribut `image` ist für Farbwähler optional und wird bei anderen Elementtypen ignoriert. Das Attribut `image` legt den Pfad der auf der Schaltfläche angezeigten Symboldatei relativ zum Ordner **Configuration** fest. Das Symbol kann jedes Format besitzen, das Dreamweaver darstellen kann, ist jedoch normalerweise eine GIF- oder JPEG-Datei.

Wenn ein Symbol für einen Farbwähler angegeben wird, ersetzt es den Farbwähler vollständig. Wenn auch das Attribut `colorRect` angegeben ist, wird die aktuelle Farbe in dem festgelegten Rechteck über dem Symbol angezeigt.

Beispiel

```
image="Toolbars/images/MM/codenav.gif"
```

disabledImage="Bildpfad"

Optional. Dreamweaver ignoriert das Attribut `disabledImage` bei anderen Elementen als Kontrollkästchen, Optionsschaltern, Menüschaltflächen, Farbwählern und Kombinationsfeldern. Dieses Attribut legt den Pfad der Symboldatei relativ zum Ordner **Configuration** fest, die Dreamweaver anzeigt, wenn die Schaltfläche deaktiviert ist. Wenn Sie das Attribut `disabledImage` nicht angeben, zeigt Dreamweaver bei deaktivierter Schaltfläche das Bild an, das im Attribut `image` festgelegt ist.

Beispiel

```
disabledImage="Toolbars/images/MM/codenav_dis.gif"
```

overImage="Bildpfad"

Optional. Dreamweaver ignoriert das Attribut `overImage` bei anderen Elementen als Schaltflächen, Kontrollkästchen, Optionsschaltern, Menüschaltflächen, Farbwählern und Kombinationsfeldern. Dieses Attribut legt den Pfad der Symboldatei relativ zum Ordner **Configuration** fest, die Dreamweaver anzeigt, wenn der Benutzer die Maus über die Schaltfläche bewegt. Falls Sie das Attribut `overImage` nicht angeben, ändert sich die Schaltfläche, wenn der Benutzer die Maus darüber bewegt, nur insofern, als Dreamweaver einen Ring um sie herum anzeigt.

Beispiel

```
overImage="Toolbars/images/MM/codenav_ovr.gif"
```

tooltip="QuickInfo-String"

Erforderlich. Dieses Attribut legt den Identifizierungstext fest (die so genannte QuickInfo), der angezeigt wird, wenn der Mauszeiger auf das Symbolleistenelement bewegt wird.

Beispiel

```
tooltip="Code Navigation"
```

label="Bezeichnungs-String"

Optional. Dieses Attribut legt eine Bezeichnung fest, die neben dem Element angezeigt wird. Dreamweaver fügt den Bezeichnungen nicht automatisch einen Doppelpunkt hinzu. Bezeichnungen für Elemente, bei denen es sich nicht um Schaltflächen handelt, werden immer links neben dem Element positioniert. Dreamweaver platziert Bezeichnungen für Schaltflächen, Kontrollkästchen, Optionsschalter, Menüschaltflächen und Kombinationsschaltflächen innerhalb der Schaltfläche und rechts neben dem Symbol.

Beispiel

```
label="Title: "
```

width="Zahl"

Optional. Dieses Attribut wird auf die Elemente von Textfeldern, Popupmenüs und Kombinationsfeldern angewandt und bestimmt die Breite dieser Elemente in Pixel. Wenn Sie das Attribut `width` nicht angeben, verwendet Dreamweaver eine Standardbreite.

Beispiel

```
width="150"
```


menuID="Menü-Id"

Dieses Attribut ist für Menüschaltflächen und Kombinationsschaltflächen erforderlich, es sei denn, Sie legen die Funktion `getMenuID()` in einer verknüpften Befehlsdatei fest. Dreamweaver ignoriert das Attribut `menuID` für andere Elementtypen. Dieses Attribut legt die ID der Menüleiste mit dem Kontextmenü fest, das geöffnet wird, wenn der Benutzer auf die Schaltfläche, die Menüschaltfläche oder die Kombinationsschaltfläche klickt. Die ID stammt aus dem ID-Attribut eines `menubar`-Tags in der Datei `menus.xml`.

Beispiel

```
menuID="DWCodeNavPopup"
```

colorRect="links oben rechts unten"

Dieses Attribut ist bei Farbwählern optional, die über ein `image`-Attribut verfügen. Bei anderen Elementtypen und bei Farbwählern, die kein Bild angeben, wird das Attribut `colorRect` ignoriert. Wenn Sie das Attribut `colorRect` angeben, zeigt Dreamweaver die Farbe, die im Farbwähler aktuell ausgewählt ist, im Rechteck links oder über dem Symbol an. Wenn Sie das Attribut `colorRect` nicht angeben, zeigt Dreamweaver die aktuelle Farbe im Bild nicht an.

Beispiel

```
colorRect="0 12 16 16"
```

file="Befehlsdateipfad"

Erforderlich für Pop-up-Menüs und Kombinationsfelder. Bei anderen Elementtypen ist das Attribut `file` optional. Das Attribut `file` legt den Pfad einer Befehlsdatei relativ zum Ordner **Configuration** fest, die JavaScript-Funktionen zum Auffüllen, Aktualisieren und Ausführen des Elements enthält. Das Attribut `file` setzt die Attribute `enabled`, `checked`, `value`, `update`, `domRequired`, `menuID`, `showIf` und `command` außer Kraft. Wenn Sie mit dem Attribut `file` eine Befehlsdatei angeben, ignoriert Dreamweaver im Allgemeinen alle äquivalenten Attribute, die im Tag angegeben sind. Weitere Informationen zu Befehlsdateien finden Sie unter „Die API-Funktionen für Symbolleistenbefehle“ auf Seite 174.

Beispiel

```
file="Toolbars/MM/EditTitle.htm"
```

domRequired="true" oder "false"

Optional. Wie bei Menüs gibt das Attribut `domRequired` an, ob die Entwurfsansicht mit der Codeansicht synchronisiert werden soll, bevor Dreamweaver den verknüpften Befehl ausführt. Wenn Sie dieses Attribut nicht angeben, lautet der Standardwert `true`. Dieses Attribut entspricht der Funktion `isDOMRequired()` in einer Symbolleisten-Befehlsdatei.

Beispiel

```
domRequired="false"
```

enabled="Skript"

Optional. Wie bei Menüs gibt das Skript einen Wert zurück, der angibt, ob das Element aktiviert ist. Wenn Sie dieses Attribut nicht angeben, lautet der Standardwert „enabled“. Das Attribut `enabled` entspricht der Funktion `canAcceptCommand()` in einer Symbolleisten-Befehlsdatei.

Beispiel

```
enabled="dw.getFocus() == 'textView' || dw.getFocus() == 'html'"
```

checked="Skript"

Dieses Attribut ist für Kontrollkästchen und Optionsschalter erforderlich. Dreamweaver ignoriert das Attribut `checked` für andere Elementtypen. Wie bei Menüs gibt das Skript einen Wert zurück, der angibt, ob das Element markiert oder nicht markiert ist. Das Attribut `checked` entspricht `isCommandChecked()` in einer Symbolleisten-Befehlsdatei. Wenn Sie dieses Attribut nicht angeben, lautet der Standardwert „unchecked“.

Beispiel

```
checked="dw.getDocumentDOM() != null && dw.getDocumentDOM().getView() == 'code'"
```

value="Skript"

Dieses Attribut ist für Popupmenüs, Kombinationsfelder und Farbwähler erforderlich. Dreamweaver ignoriert das Attribut `value` für andere Elementtypen.

Um zu bestimmen, welcher Wert für Popupmenüs und Kombinationsfelder angezeigt werden soll, ruft Dreamweaver zuerst `isCommandchecked()` für jedes Element im Menü auf. Wenn die Funktion `isCommandchecked()` für Elemente den Wert `true` zurückgibt, zeigt Dreamweaver den Wert des ersten Elements an. Wenn keine Elemente den Wert `true` zurückgeben oder die Funktion `isCommandchecked()` nicht definiert ist, ruft Dreamweaver die Funktion `getCurrentValue()` auf oder führt das Skript aus, das im Attribut `value` angegeben ist. Falls das Steuerelement ein Kombinationsfeld ist, zeigt Dreamweaver den zurückgegebenen Wert an. Falls das Steuerelement ein Popupmenü ist, fügt Dreamweaver den zurückgegebenen Wert vorübergehend der Liste hinzu und zeigt ihn an.

In allen anderen Fällen gibt das Skript den aktuellen Wert für die Anzeige zurück. Bei Popupmenüs oder Kombinationsfeldern sollte es sich bei dem Wert um eines der Elemente in der Menüliste handeln. Bei Kombinationsfeldern und Textfeldern kann der Wert ein beliebiger String sein, den das Skript zurückgibt. Bei Farbwählern sollte der Werte eine gültige Farbe sein, doch erzwingt Dreamweaver dies nicht.

Das `value`-Attribut entspricht der Funktion `getCurrentValue()` in einer Symbolleisten-Befehlsdatei.

update="update_frequency_list"

Optional. Dieses Attribut legt fest, wie oft die Verarbeitungsprozeduren für `enabled`, `checked`, `showif` und `value` ausgeführt werden sollen, um die Anzeige des Elements zu aktualisieren. Das Attribut `update` entspricht der Funktion `getUpdateFrequency()` in einer Symbolleisten-Befehlsdatei.

Die Aktualisierungshäufigkeit für Symbolleistenelemente muss angegeben werden, weil diese Elemente im Gegensatz zu Menüelementen immer dargestellt werden. Aus diesem Grund sollten Sie die kleinstmögliche Häufigkeit wählen und sicherstellen, dass die Verarbeitungsprozeduren für `enabled`, `checked` und `value` so einfach wie möglich sind.

In der folgenden Tabelle werden die möglichen Werte für *Aktualisierungshäufigkeitsliste* von der kleinsten bis zur größten Häufigkeit aufgeführt. Wenn Sie das Attribut `update` nicht festlegen, besitzt die Aktualisierungshäufigkeit standardmäßig den Wert `onEdit`. Sie können mehrere Aktualisierungshäufigkeiten getrennt durch Kommas angeben. Die Verarbeitungsprozeduren werden bei allen folgenden angegebenen Ereignissen ausgeführt:

- `onServerModelChange` wird ausgeführt, wenn sich das Servermodell der aktuellen Seite ändert.
- `onCodeViewSyncChange` wird ausgeführt, wenn die Synchronisierung der Codeansicht mit der Entwurfsansicht hergestellt wird oder verloren geht.
- `onViewChange` wird immer dann ausgeführt, wenn der Benutzer den Fokus zwischen Codeansicht und Entwurfsansicht wechselt oder wenn er zwischen Codeansicht, Entwurfsansicht und Code- und Entwurfsansicht wechselt.
- `onEdit` wird immer ausgeführt, wenn das Dokument in der Entwurfsansicht bearbeitet wird. In der Codeansicht vorgenommene Änderungen lösen dieses Ereignis nicht aus.
- `onSelChange` wird immer dann ausgeführt, wenn sich die Auswahl in der Entwurfsansicht ändert. In der Codeansicht vorgenommene Änderungen lösen dieses Ereignis nicht aus.
- `onEveryIdle` wird regelmäßig ausgeführt, wenn sich die Anwendung im Wartezustand befindet. Dies kann sehr viel Zeit in Anspruch nehmen, weil die Verarbeitungsprozeduren `enabled`, `checked`, `showif` und `value` häufig ausgeführt werden. Es sollte nur bei Schaltflächen verwendet werden, deren Aktivierungsstatus zu besonderen Zeiten geändert werden muss. Außerdem sollten die Verarbeitungsprozeduren schnell sein.

Hinweis: In allen diesen Fällen führt Dreamweaver die Verarbeitungsprozeduren erst aus, nachdem das angegebene Ereignis eingetreten ist und die Anwendung sich im Ruhezustand befindet. Es ist nicht sicher, dass die Verarbeitungsprozeduren nach jeder Bearbeitungs- oder Auswahländerung ausgeführt werden, vielmehr werden sie erst einige Zeit nach mehreren Bearbeitungs- oder Auswahländerungen ausgeführt. Die Verarbeitungsprozeduren werden jedoch garantiert ausgeführt, wenn der Benutzer auf ein Symbolleistenelement klickt.

Beispiel

```
update="onViewChange"
```

command="Skript"

Dieses Attribut ist für alle Elemente mit Ausnahme der Menüs Schaltflächen erforderlich. Dreamweaver ignoriert das Attribut `command` für Menüs Schaltflächen. Legt die JavaScript-Funktion fest, die ausgeführt werden soll, wenn der Benutzer eine der folgenden Aktionen ausführt:

- Auf eine Schaltfläche klicken
- Element aus einem Pop-up-Menü oder einem Kombinationsfeld auswählen
- In einem Text- oder Kombinationsfeld die Tabulatortaste oder die Eingabetaste drücken oder auf eine Stelle außerhalb des Felds klicken
- Er wählt eine Farbe in einem Farbwähler.

Das Attribut `command` entspricht der Funktion `receiveArguments()` in einer Symbolleisten-Befehlsdatei.

Beispiel

```
command="dw.toggleLiveDebug() "
```

arguments="argument_list"

Optional. Dieses Attribut gibt eine Liste an, in der die Attribute durch Kommas getrennt sind. Die Argumente werden an die Funktion `receiveArguments()` in einer Symbolleisten-Befehlsdatei übergeben. Wenn Sie das Attribut `arguments` nicht angeben, übergibt Dreamweaver die ID des Symbolleistenelements. Darüber hinaus übergeben Pop-up-Menüs, Kombinationsfelder, Textfelder und Farbwähler ihren aktuellen Wert als erstes Argument, noch vor allen Argumenten, die das Attribut `arguments` festlegt, und vor der Element-ID, falls keine Argumente angegeben wurden.

Beispiel

In einer Symbolleiste mit den Schaltflächen **Rückgängig** und **Wiederholen** ruft jede der beiden Schaltflächen die Menübefehlsdatei `Edit_Clipboard.htm` auf und übergibt, wie im folgenden Beispiel dargestellt, ein Argument, das die Aktion festlegt.

```
<button id="DW_Undo"
  image="Toolbars/images/MM/undo.gif"
  disabledImage="Toolbars/images/MM/undo_dis.gif"
  tooltip="Undo"
  file="Menus/MM/Edit_Clipboard.htm"
  arguments="'undo' "
  update="onEveryIdle"/>

<button id="DW_Redo"
  image="Toolbars/images/MM/redo.gif"
  disabledImage="Toolbars/images/MM/redo_dis.gif"
  tooltip="Redo"
  file="Menus/MM/Edit_Clipboard.htm"
  arguments="'redo' "
  update="onEveryIdle"/>
```

Die API-Funktionen für Symbolleistenbefehle

In vielen Fällen, in denen Sie ein Skript für ein Attribut angeben, können Sie das Attribut auch über eine JavaScript-Funktion in einer Befehlsdatei implementieren. Diese Aktion ist dann notwendig, wenn die Funktionen Argumente benötigen, wie in der Befehlsprozedur für ein Textfeld. Dieses Verfahren ist für Popupmenüs und Kombinationsfelder erforderlich.

Die Befehlsdatei-API für Symbolleistenelemente ist eine Erweiterung der Menübefehlsdatei-API. Sie können also Menübefehlsdateien, vielleicht mit einigen zusätzlichen Funktionen für Symbolleisten, direkt als Symbolleisten-Befehlsdateien wieder verwenden.

canAcceptCommand()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Bestimmt, ob das Symbolleistenelement aktiviert ist. Standardmäßig befindet sich ein Element in aktiviertem Zustand. Deshalb sollten Sie diese Funktion nur dann definieren, wenn in mindestens einem Fall der Wert `false` zurückgegeben wird.

Argumente

Bei Popupmenüs, Kombinationsfeldern und Farbwählern ist das erste Argument der aktuelle Wert im Steuerelement. Die Funktion `getDynamicContent()` kann optional einzelne IDs mit Elementen innerhalb eines Popupmenüs verknüpfen. Wenn mit dem ausgewählten Element im Popupmenü eine ID verknüpft ist, übergibt Dreamweaver nicht den Wert, sondern diese ID an `canAcceptCommand()`. Bei Kombinationsfeldern übergibt Dreamweaver den Inhalt des Textfelds, wenn der aktuelle Inhalt des Textfelds nicht mit einem Eintrag im Popupmenü übereinstimmt. Dreamweaver vergleicht den Inhalt des Textfelds mit den Einträgen im Popupmenü ohne Berücksichtigung von Groß- und Kleinschreibung, um Übereinstimmungen zwischen dem Inhalt des Textfelds und den Einträgen in der Liste zu ermitteln.

Wenn Sie das Attribut `arguments` für dieses Symbolleistenelement in der Datei `toolbars.xml` angeben, werden diese Argumente als Nächstes übergeben. Wenn Sie das Attribut `arguments` nicht angegeben haben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn das Element aktiviert ist und `false`, wenn das Element nicht aktiviert ist.

Beispiel

```
function canAcceptCommand()  
{  
    return (dw.getDocumentDOM() != null);  
}
```

getCurrentValue()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Gibt den aktuellen Wert zurück, der im Element angezeigt werden soll. Dreamweaver ruft die Funktion `getCurrentValue()` für Popupmenüs, Kombinationsfelder, Textfelder und Farbwähler auf. Bei Popupmenüs sollte der aktuelle Wert eines der Elemente im Menü sein. Falls der Wert kein Popupmenü ist, wählt Dreamweaver das erste Element aus. Bei Kombinationsfeldern und Textfeldern kann dieser Wert ein beliebiger String sein, den die Funktion zurückgibt. Bei Farbwählern sollte der Werte eine gültige Farbe sein, doch erzwingt Dreamweaver dies nicht. Diese Funktion entspricht dem Attribut `value`.

Argumente

Keiner

Rückgabewerte

Dreamweaver erwartet einen String, der den aktuellen Anzeigewert enthält. Beim Farbwähler enthält der String die ausgewählte Farbe in RGB-Format, z. B. „#FFFFFF“ für die Farbe Weiß.

Beispiel

```
function getCurrentValue()  
{  
    var title = "";  
    var dom = dw.getDocumentDOM();  
    if (dom)  
        title = dom.getTitle();  
    return title;  
}
```

getDynamicContent()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion ist für Popupmenüs und Kombinationsfelder erforderlich. Wie bei Menüs gibt diese Funktion ein Array von Strings zurück, die das Popupmenü füllen. Jeder String kann optional mit „;id=id“ enden. Wenn eine ID angegeben ist, übergibt Dreamweaver die ID an die Funktion `receiveArguments()` und nicht an den eigentlichen String, der im Menü angezeigt werden soll.

Der Name `getDynamicContent()` ist eine falsche Bezeichnung, weil diese Funktion auch dann verwendet werden sollte, wenn die Liste der Einträge im Menü fest ist. Beispielsweise handelt es sich bei der Datei **Text_Size.htm** im Ordner **Configuration\Menus\MM** nicht um ein dynamisches Menü. Sie ist dafür vorgesehen, von allen Elementen eines statischen Menüs aufgerufen zu werden. Durch Hinzufügen einer `getDynamicContent()`-Funktion, die einfach die Liste möglicher Schriftgrößen zurückgibt, lässt sich dieselbe Befehlsdatei jedoch auch für ein Popupmenü in der Symbolleiste verwenden. Unterstriche in den Strings eines zurückgegebenen Arrays werden von Symbolleistenelementen ignoriert. Menübefehlsdateien lassen sich somit erneut verwenden. In der Menübefehlsdatei ignoriert Dreamweaver die Funktion `getDynamicContent()`, da das Menüelement nicht als dynamisch markiert ist.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array von Strings, mit denen das Menü aufgefüllt wird.

Beispiel

```
function getDynamicContent()
{
    var items = new Array;
    var filename = dw.getConfigurationPath() + "/Toolbars/MM/AddressList.xml";
    var location = MMNotes.localURLToFilePath(filename);
    if (DWfile.exists(location))
    {
        var addressData = DWfile.read(location);
        var addressDOM = dw.getDocumentDOM(dw.getConfigurationPath() +
            '/Shared/MM/Cache/empty.htm');
        addressDOM.documentElement.outerHTML = addressData;
        var addressNodes = addressDOM.getElementsByTagName("url");
        if (addressNodes.length)
        {
            for (var i=0; i < addressNodes.length ; i++ )
            {
                items[i] = addressNodes[i].address + ";id='" +
                    addressNodes[i].address + "'";
            }
        }
    }
    return items;
}
```

getMenuID()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Gilt nur für Menüschaltflächen. Dreamweaver ruft die Funktion `getMenuID()` auf, um die ID des Menüs abzurufen, das angezeigt werden soll, wenn der Benutzer auf die Schaltfläche klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String, der eine in der Datei **menus.xml** definierte Menü-ID enthält.

Beispiel

```
function getMenuID()
{
    var dom = dw.getDocumentDOM();
    var menuID = '';
    if (dom)
    {
        var view = dom.getView();
        var focus = dw.getFocus();
        if (view == 'design')
        {
            menuID = 'DWDesignOnlyOptionsPopup';
        }
        else if (view == 'split')
        {
            if (focus == 'textView')
            {
                menuID = 'DWSplitCodeOptionsPopup';
            }
            else
            {
                menuID = 'DWSplitDesignOptionsPopup';
            }
        }
    }
}
```

```

    }
}
else if (view == 'code')
{
    menuID = 'DWCodeOnlyOptionsPopup';
}
else
{
    menuID = 'DWBrowseOptionsPopup';
}
}
return menuID;
}

```

getUpdateFrequency()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Legt fest, wie oft die Verarbeitungsprozeduren für die Attribute `enabled`, `checked`, `showIf` und `value` ausgeführt werden, um die Anzeige des Elements zu aktualisieren.

Die Aktualisierungshäufigkeit für Symbolleistenelemente muss angegeben werden, weil diese Elemente im Gegensatz zu Menüs immer dargestellt werden. Aus diesem Grund sollten Sie die kleinstmögliche Häufigkeit wählen und sicherstellen, dass die Verarbeitungsprozeduren für `enabled`, `checked` und `value` so einfach wie möglich sind.

Diese Funktion entspricht dem Attribut `update` in einem Symbolleistenelement.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String, der eine durch Kommas getrennte Liste von Aktualisierungsprozeduren enthält. Eine vollständige Liste der möglichen Aktualisierungsprozeduren finden Sie unter „`update=update_frequency_list`“ auf Seite 172.

Beispiel

```

function getUpdateFrequency()
{
    return onSelChange";
}

```

isCommandChecked()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Gibt einen Wert zurück, der angibt, ob das Element ausgewählt ist. Bei einer Schaltfläche bedeutet „checked“, dass sie aktiviert bzw. gedrückt angezeigt wird. Die Funktion `isCommandChecked()` entspricht dem Attribut `checked` in einem Symbolleistenelement-Tag.

Argumente

Bei Popupmenüs, Kombinationsfeldern und Farbwählern ist das erste Argument der aktuelle Wert im Steuerelement. Die Funktion `getDynamicContent()` kann optional einzelne IDs mit Elementen innerhalb eines Popupmenüs verknüpfen. Wenn mit dem ausgewählten Element im Menü eine ID verknüpft ist, übergibt Dreamweaver nicht den Wert, sondern diese ID an die Funktion `isCommandChecked()`. Bei Kombinationsfeldern übergibt Dreamweaver den Inhalt des Textfelds, wenn der aktuelle Inhalt des Textfelds nicht mit einem Eintrag im Popupmenü übereinstimmt. Um zu bestimmen, ob eine Übereinstimmung mit dem Textfeld besteht, führt Dreamweaver einen Vergleich mit dem Menü ohne Berücksichtigung von Groß-/Kleinschreibung durch.

Wenn Sie das Attribut `arguments` angegeben haben, werden diese Argumente als Nächstes übergeben. Wenn Sie das Attribut `arguments` nicht angeben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das Element aktiviert ist, und `false`, wenn das Element nicht aktiviert ist.

Beispiel

Im folgenden Beispiel wird bestimmt, welches Element, soweit vorhanden, in einem Popupmenü mit Absatzformaten und CSS-Stilen markiert sein soll:

```
function isCommandChecked()
{
    var bChecked = false;
    var style = arguments[0];
    var textFormat = dw.getDocumentDOM().getTextFormat();

    if (dw.getDocumentDOM() == null)
        bChecked = false;

    if (style == "(None)")
        bChecked = (dw.cssStylePalette.getSelectedStyle() == '' || textFormat ==
" || textFormat == "P" || textFormat == "PRE");
    else if (style == "Heading 1")
        bChecked = 0(textFormat == "h1");
    else if (style == "Heading 2")
        bChecked = 0(textFormat == "h2");
    else if (style == "Heading 3")
        bChecked = 0(textFormat == "h3");
    else if (style == "Heading 4")
        bChecked = 0(textFormat == "h4");
    else if (style == "Heading 5")
        bChecked = 0(textFormat == "h5");
    else if (style == "Heading 6")
        bChecked = 0(textFormat == "h6");
    else
        bChecked = (dw.cssStylePalette.getSelectedStyle() == style);

    return bChecked;
}
```

isDOMRequired()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Legt fest, ob für den Symbolleisten-Befehl ein gültiges DOM erforderlich ist. Sollte diese Funktion den Wert `true` zurückgeben oder nicht definiert sein, geht Dreamweaver davon aus, dass für den Befehl ein gültiges DOM erforderlich ist, und synchronisiert die Code- und Entwurfsansicht des Dokuments vor der Ausführung des verknüpften Befehls. Diese Funktion entspricht dem Attribut `domRequired` in einem Symbolleistenelement.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das DOM erforderlich ist, und `false`, wenn dies nicht der Fall ist.

Beispiel

```
function isDOMRequired()
{
    return false;
}
```

receiveArguments()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Verarbeitet alle Argumente, die von einem Symbolleistenelement übergeben werden. Die Funktion `receiveArguments()` entspricht dem Attribut `command` in einem Symbolleistenelement-Tag.

Argumente

Bei Popupmenüs, Kombinationsfeldern und Farbwählern ist das erste Argument der aktuelle Wert im Steuerelement. Die Funktion `getDynamicContent()` kann optional einzelne IDs mit Elementen innerhalb eines Popupmenüs verknüpfen. Wenn mit dem ausgewählten Element im Popupmenü eine ID verknüpft ist, übergibt Dreamweaver nicht den Wert, sondern diese ID an die Funktion `receiveArguments()`. Bei Kombinationsfeldern übergibt Dreamweaver den Inhalt des Textfelds, wenn der aktuelle Inhalt des Textfelds nicht mit einem Eintrag im Popupmenü übereinstimmt. Um zu bestimmen, ob eine Übereinstimmung mit dem Textfeld besteht, führt Dreamweaver einen Vergleich mit dem Popupmenü ohne Berücksichtigung von Groß-/Kleinschreibung durch.

Wenn Sie das Attribut `arguments` angegeben haben, werden diese Argumente als Nächstes übergeben. Wenn Sie das Attribut `arguments` nicht angegeben haben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte

Dreamweaver erwartet keinen Wert.

Beispiel

```
function receiveArguments(newTitle)
{
    var dom = dw.getDocumentDOM();
    if (dom)
        dom.setTitle(newTitle);
}
```

showIf()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Legt fest, dass ein Element nur dann in der Symbolleiste angezeigt wird, wenn die Funktion den Wert `true` zurückgibt. Beispielsweise können Sie mit der Funktion `showIf()` bestimmte Schaltflächen nur dann einblenden lassen, wenn die Seite ein bestimmtes Servermodell besitzt. Falls `showIf()` nicht definiert ist, wird das Element immer eingeblendet. Die Funktion `showIf()` entspricht dem Attribut `showIf` in einem Symbolleistenelement-Tag.

Die Funktion `showIf()` wird immer aufgerufen, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig von dem Wert, den die Funktion `getUpdateFrequency()` zurückgibt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das Element angezeigt wird, und `false`, wenn dies nicht der Fall ist.

Beispiel

```
function showIf()  
{  
    var retval = false;  
    var dom = dw.getDocumentDOM();  
  
    if (dom)  
    {  
        var view = dom.getView();  
        if(view == 'design')  
        {  
            retval = true;  
        }  
    }  
    return retval;  
}
```

Kapitel 12: Berichte

Adobe Dreamweaver CS3 unterstützt zwei Berichtstypen: Site-Berichte und eigenständige Berichte.

Site-Berichte

Die API für Berichte dient zum Erstellen benutzerdefinierter Site-Berichte oder zum Ändern der mit Dreamweaver gelieferten vordefinierten Berichte. Der Zugriff auf Site-Berichte ist nur über das Dialogfeld **Site-Berichte** möglich.

Site-Berichte befinden sich im Ordner **Configuration\Reports** von Dreamweaver. Der Ordner **Reports** enthält Unterordner für die einzelnen Berichtskategorien. Jeder Bericht kann nur einer Kategorie angehören. Kategorienamen können maximal 31 Zeichen umfassen. In jedem Unterordner kann sich eine Datei mit dem Namen **_foldername.txt** befinden. Wenn diese Datei vorhanden ist, verwendet Dreamweaver ihren Inhalt als Kategorienamen. Wenn die Datei **_foldername.txt** nicht vorhanden ist, verwendet Dreamweaver den Ordernamen als Kategorienamen.

Wenn der Benutzer im Dialogfeld **Site-Berichte** mehrere Site-Berichte auswählt, gibt Dreamweaver alle Ergebnisse im selben Ergebnisfenster unter der Registerkarte **Site-Berichte** im Bedienfeld **Ergebnisse** aus. Dreamweaver ersetzt diese Ergebnisse, wenn der Benutzer das nächste Mal einen Site-Bericht ausführt.

In der folgenden Tabelle sind die Dateien zum Erstellen von Site-Berichten aufgeführt:

Pfad	Datei	Beschreibung
Configuration\Reports\{Typ\}	Berichtname.js	Enthält die Funktionen zum Generieren des Berichtsinhalts.
Configuration\Reports\{Typ\}	Berichtname.htm	Ruft die erforderlichen JavaScript-Dateien auf. Definiert die Benutzeroberfläche (UI) für das Dialogfeld Einstellungen des Berichts, sofern erforderlich.
Configuration\Reports\	Reports.js	Stellt allgemeine Funktionen zum Generieren von Berichten zur Verfügung.

So funktionieren Site-Berichte

- 1 Berichte werden über den Befehl **Site > Berichte** aufgerufen. Wenn Sie diesen Menübefehl wählen, wird ein Dialogfeld angezeigt, in dem Sie einen Bericht auswählen können, der an einem von mehreren möglichen Zielen ausgeführt wird.
- 2 Legen Sie im Popupmenü **Bericht über** fest, welche Dateien in den ausgewählten Bericht aufgenommen werden sollen. Dieses Menü enthält die Befehle **Aktuelles Dokument**, **Gesamte aktuelle lokale Site**, **Ausgewählte Dateien der Site** und **Ordner**. Wenn der Benutzer den Befehl **Ordner** wählt, kann er über die Schaltfläche **Durchsuchen** und ein Textfeld den gewünschten Ordner auswählen.
- 3 Sie können Berichte, die Parameter enthalten, benutzerspezifisch anpassen. Klicken Sie hierzu auf die Schaltfläche **Einstellungen**, und geben Sie Werte für die Parameter ein. Damit die Benutzer Berichtparameter festlegen können, muss der jeweilige Bericht das Dialogfeld **Einstellungen** enthalten. Dieses Dialogfeld ist optional. Nicht alle Berichte erfordern die Angabe von Berichtsparemtern durch den Benutzer. Wenn ein Bericht nicht über das Dialogfeld **Einstellungen** verfügt, ist bei der Auswahl des Berichts in der Liste die Schaltfläche **Einstellungen** abgeblendet.
- 4 Wenn der Benutzer die Berichte ausgewählt und die Einstellungen festgelegt hat, klickt er auf die Schaltfläche **Ausführen**.

Hinweis: Verwendet ein Bericht die Prozedur `preventFileActivity`, so verhindert Dreamweaver, dass der Benutzer eine andere Dateiaktivität durchführen kann, während dieser Bericht ausgeführt wird.

Daraufhin löscht Dreamweaver auf der Registerkarte **Site-Berichte** alle Elemente im Bedienfeld **Ergebnisse**. Dreamweaver ruft in jedem Bericht die Funktion `beginReporting()` auf, bevor der Berichtsprozess beginnt. Wenn ein Bericht von dieser Funktion den Wert `false` zurück erhält, wird er aus dem Berichtsprozess entfernt.

5 Dateien werden mit der Funktion `processFile()` an jeden Bericht übergeben, der im Dialogfeld **Berichte** ausgewählt wurde. Wenn der Bericht Informationen über eine Datei in der Ergebnisliste anzeigen soll, muss er die Funktion `dw.resultsPalette.siteReports.addResultItem()` aufrufen. Dies wird so lange fortgesetzt, bis alle ausgewählten Dateien verarbeitet sind oder Sie unten im Fenster auf die Schaltfläche **Stopp** klicken. Dreamweaver zeigt den Namen aller bereits verarbeiteten Dateien und die Anzahl der noch zu verarbeitenden Dateien an.

6 Dreamweaver ruft in jedem Bericht die Funktion `endReporting()` auf, nachdem alle Dateien verarbeitet wurden und der Berichtsprozess abgeschlossen ist.

Einfaches Beispiel für einen Site-Bericht

Die einfache Beispielerweiterung listet alle Bilder auf, auf die in einer bestimmten Datei, auf einer gesamten Site, in ausgewählten Dateien oder in einem Ordner verwiesen wird, und zeigt den Bericht im Ergebnissenfenster unter der Registerkarte **Site-Berichte** an.

Führen Sie zum Erstellen der Erweiterung die folgenden Schritte aus:

- „Berichtdefinition erstellen“ auf Seite 182
- „JavaScript-Code schreiben“ auf Seite 183

In diesem Beispiel werden zwei Dateien im Ordner **HTML Reports** erstellt: die Datei `List Images.htm` mit der Berichtdefinition und die Datei `List Images.js`, die den für diesen Bericht spezifischen JavaScript-Code enthält. Zusätzlich verweisen Sie auf die Datei **Reports.js**, die in Dreamweaver integriert ist.

Berichtdefinition erstellen

Die Berichtdefinition gibt den Berichtsnamen an, wie er im Dialogfeld **Berichte** angezeigt wird, ruft benötigte JavaScript-Dateien auf und definiert die Benutzeroberfläche des Dialogfelds **Einstellungen**, sofern erforderlich.

So erstellen Sie die Berichtdefinition:

1 Erstellen Sie im Ordner **Configuration\Reports\HTML Reports** die Datei **ListImages.htm**.

2 Fügen Sie Folgendes hinzu, um den Berichtsnamen anzugeben, der im Dialogfeld **Berichte** im Titel der HTML-Seite angezeigt werden soll.

```
<html>
<head>
<title>List Images</title>
```

3 Fügen Sie an das Ende der Datei das `script`-Tag ein, und geben Sie mit dem Attribut `src` die Datei **Reports.js** an.

```
<script src="../../Reports.js"></script>
```

4 Fügen Sie am Ende der Datei ein weiteres `script`-Tag ein, und geben Sie mit dem Attribut `src` die Datei **ListImages.js** an, die Sie erstellen.

```
<html>
<head>
<title>List Images</title>
<script src="../../Reports.js"></script>
<script src="List Images.js"></script>
```

5 Schließen Sie das `head`-Tag, fügen Sie ein öffnendes und schließendes `body`-Tag ein, und schließen Sie das `html`-Tag.

```
</head>
<body>
</body>
</html>
```

6 Speichern Sie die Datei unter dem Namen **ListImages.js** im Ordner **Configuration\Reports\HTML Reports**.

JavaScript-Code schreiben

Die Datei **Reports.js** ist bereits in Dreamweaver enthalten. Sie können beliebige Funktionen aus **Reports.js** aufrufen, müssen aber auch die JavaScript-Datei mit etwaigen für Ihren benutzerdefinierten Site-Bericht spezifischen Funktionen erstellen.

So erstellen Sie die JavaScript-Datei:

1 Erstellen Sie im Ordner **Configuration\Reports\HTML Reports** die Datei **ListImages.js** mit folgendem Inhalt:

```
// Funktion: configureSettings
// Beschreibung: Standardbericht-API, zum Initialisieren und Laden
//der Standardwerte. Initialisiert die UI nicht.
//
function configureSettings() {
    return false;
}
// Funktion: processFile
// Beschreibung: Während Dateiverarbeitung aufgerufene Berichtsbefehls-API.
//
function processFile (fileURL) {
    if (!isHTMLType(fileURL)) //Handelt es sich nicht um eine HTML-Datei,
        return; //wird die Datei übersprungen.
    var curDOM = dw.getDocumentDOM(fileURL); // Variable für DOM
    var tagList = curDOM.getElementsByTagName('img'); // Variable für img-Tags
    var imgfilename; // Variable für den im img-Tag angegebenen Dateinamen
    for (var i=0; i < tagList.length; i++) { // Für img-Tagliste
        imgfilename = tagList[i].getAttribute('src'); // Bilddateinamen abrufen
        if (imgfilename != null) { // Wenn ein Dateiname angegeben ist
            // Entsprechendes Symbol, HTML-Dateiname,
            // Bilddateiname und Zeilennummer ausgeben

            reportItem(REP_ITEM_CUSTOM, fileURL, imgfilename,
                curDOM.nodeToSourceViewOffsets(tagList[i])); }
        }
    }
}
```

2 Speichern Sie die Datei unter dem Namen **ListImages.js** im Ordner **Configuration\Reports\HTML Reports**.

Eigenständige Berichte

Mit der API für das Ergebnisfenster lassen sich eigenständige Berichte erstellen. Eigenständige Berichte sind reguläre Befehle, die direkt die API für das Ergebnisfenster und nicht die API für Berichte verwenden. Auf einen eigenständigen Bericht können Sie wie auf alle anderen Befehle über die Menüs oder einen anderen Befehl zugreifen.

Eigenständige Berichte befinden sich im Ordner **Configuration\Commands** von Dreamweaver. Benutzerdefinierte Befehle für eigenständige Berichte werden im Menü **Befehle** angezeigt.

Dreamweaver erstellt jedes Mal ein neues Fenster **Ergebnisse**, wenn der Benutzer einen neuen eigenständigen Bericht ausführt.

Pfad	Datei	Beschreibung
Configuration\Commands\	Befehlsname.htm	Definiert die Benutzeroberfläche (UI) für das Dialogfeld, das eingeblendet wird, wenn der Benutzer den Befehl wählt. Die Datei enthält außerdem den JavaScript-Code bzw. einen Verweis auf eine JavaScript-Datei mit dem Code, der die zum Generieren des Berichts erforderlichen Aktionen durchführt.
Configuration\Commands\	Befehlsname.js	Generiert ein Ergebnisfenster und fügt den Bericht in das Fenster ein.

Funktionsweise von eigenständigen Berichten

- 1 Der benutzerdefinierte Befehl (der Befehl, den Sie zum Generieren des Berichts erstellen) öffnet ein neues Ergebnisfenster, indem er die Funktion `dw.createResultsWindow()` aufruft und die zurückgegebenen Ergebnisobjekte in einer Fenstervariable speichert. Die übrigen Funktionen in diesem Prozess sollten als Methoden dieser Objekte aufgerufen werden.
- 2 Der benutzerdefinierte Prozess initialisiert den Titel und das Format des Fensters **Ergebnisse**, indem er die Funktionen `setTitle()` und `setColumnWidths()` als Methoden des Ergebnisfenster-Objekts aufruft.
- 3 Der Befehl kann entweder dem Fenster **Ergebnisse** durch Aufrufen der Funktion `addItem()` Elemente direkt hinzufügen oder durch Aufrufen von `setFileList()` und `startProcessing()` als Methoden des Ergebnisfenster-Objekts eine Dateiliste durchlaufen.
- 4 Wenn `resWin.startProcessing()` vom Befehl aufgerufen wird, ruft Dreamweaver für jede Datei-URL in der Liste die Funktion `processFile()` auf. Definieren Sie die Funktion `processFile()` in dem eigenständigen Befehl. Als einziges Argument erhält sie die Datei-URL. Verwenden Sie die Funktion `setCallbackCommands()` des Ergebnisfenster-Objekts, wenn Dreamweaver die Funktion `processFile()` in einem anderen Befehl aufrufen soll.
- 5 Um `addItem()` aufzurufen, benötigt die Funktion `processFile()` Zugriff auf das Fenster **Ergebnisse**, das von dem eigenständigen Befehl erstellt wurde. Die Funktion `processFile()` kann auch die Funktion `stopProcessing()` des Ergebnisfenster-Objekts aufrufen, um die Verarbeitung der Dateiliste anzuhalten.

Einfaches Beispiel für einen eigenständigen Bericht

Die einfache Erweiterung für einen eigenständigen Bericht listet alle Bilder auf, auf die in einer bestimmten Datei verwiesen wird, und zeigt den Bericht im Fenster **Ergebnisse** an.

Führen Sie zum Erstellen der Erweiterung die folgenden Schritte aus:

- 1 „Benutzeroberfläche des Dialogfelds erstellen“ auf Seite 184
- 2 „JavaScript-Code schreiben“ auf Seite 185

In diesem Beispiel werden zwei Dateien im Ordner **Configuration\Commands** erstellt: die Datei **ListImages.htm** zur Definition der Benutzeroberfläche des Dialogfelds, das angezeigt wird, wenn der Benutzer den benutzerdefinierten Befehl wählt sowie die Datei **ListImages.js**, die den für diesen Bericht spezifischen JavaScript-Code enthält.

Benutzeroberfläche des Dialogfelds erstellen

Der body-Abschnitt der HTML-Datei gibt den Inhalt des Dialogfelds an, das angezeigt wird, wenn der Benutzer den benutzerdefinierten Befehl wählt, und ruft etwaige benötigte JavaScript-Dateien auf.

So erstellen Sie die HTML-Datei:

- 1 Erstellen Sie eine Datei namens **ListImages.htm** im Ordner **Configuration\Commands**.
- 2 Geben Sie Folgendes in die Datei **ListImages.htm** ein:

```
<html>
<head>
<title>Standalone report example</title>
<script src="Listimages.js">
</script>
</head>
<body>
<div name="test">
<form name="myForm">
<table>
<tr>
<td>Click OK to display the standalone report.</td>
</tr>
</table>
</form>
</div>
</body>
```

- 3 Speichern Sie die Datei unter dem Namen **ListImages.htm** im Ordner **Configuration\Commands**.

JavaScript-Code schreiben

Als Nächstes erstellen Sie die JavaScript-Datei mit etwaigen für den eigenständigen Bericht spezifischen Funktionen.

So erstellen Sie die JavaScript-Datei:

1 Erstellen Sie die Datei **ListImages.js** im Ordner **Configuration\Commands** mit folgendem Code:

```
function stdaloneresultwin()
{
    var curDOM = dw.getDocumentDOM("document");
    var tagList = curDOM.getElementsByTagName('img');
    var imgfilename;
    var iOffset = 0;
    var iLineNumber = 0;

    var resWin = dw.createResultsWindow("Images in File", ["Line", "Image"]);

    for (var i=0; i < tagList.length; i++)
    {
        // Namen der Quelldatei abrufen.
        imgfilename = tagList[i].getAttribute('src');
        // Zeichen-Offset ab Dateianfang
        // bis zum Beginn des img-Tags abrufen.
        iOffset = curDOM.nodeToOffsets(curDOM.images[i]);
        // Auf Grundlage des Offsets ermitteln, in welcher
        // Zeile der Datei sich das img-Tag befindet.
        iLineNumber = curDOM.getLineFromOffset(iOffset[0]);
        // Sofern das src-Attribut einen Dateinamen angibt,
        if (imgfilename != null)
        { // Zeilennummer und Bilddateipfad anzeigen.
            resWin.addItem(resWin, "0", "Images in Current File", null, -
                null, null, [iLineNumber, imgfilename]);
        }
    }
    return;
}

// Schaltflächen in Dialogfeld einfügen
function commandButtons()
{
    return new Array("OK", "stdaloneresultwin()", "Cancel", "window.close()");
}
```

2 Speichern Sie die Datei unter dem Namen **ListImages.js** im Ordner **Configuration\Commands**.

Die API-Funktionen für Berichte

Die einzige erforderliche Funktion der API für Berichte ist die Funktion `processFile()`. Alle anderen Funktionen sind optional.

processFile()

Verfügbarkeit

Dreamweaver 4

Beschreibung

Diese Funktion wird aufgerufen, wenn eine Datei verarbeitet werden soll. Der Report-Befehl verarbeitet die Datei, ohne sie zu modifizieren, und verwendet die Funktion `dw.ResultsPalette.SiteReports()`, `addResultItem()` oder `resWin.addItem()`, um Informationen über die Datei zurückzugeben. Dreamweaver gibt das DOM im Anschluss an die Verarbeitung automatisch wieder frei.

Argumente

strFilePath

Das Argument *strFilePath* enthält den vollständigen Pfad und den Dateinamen der zu verarbeitenden Datei.

Rückgabewerte

Dreamweaver erwartet keinen Wert.

beginReporting()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Diese Funktion wird am Anfang des Berichtsprozesses noch vor der Ausführung des ersten Berichts aufgerufen. Wenn der Report-Befehl den Wert `false` von dieser Funktion zurückgibt, wird der Report-Befehl aus dem Berichtsprozess entfernt.

Argumente

target

Das Argument *target* ist ein String, der das Ziel der Berichtssitzung angibt. Hierbei kann es sich um einen der folgenden Werte handeln: "CurrentDoc", "CurrentSite", "CurrentSiteSelection" (für die ausgewählten Dateien innerhalb einer Site) oder "Folder: + der vom Benutzer gewählte Pfad" (beispielsweise "Folder:C:\temp").

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn der Bericht erfolgreich ausgeführt wird, und `false`, wenn *target* aus der Ausführung des Berichts ausgeschlossen wird.

endReporting()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Diese Funktion wird am Ende des Berichtsprozesses aufgerufen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

commandButtons()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Definiert die Schaltflächen für die rechte Seite des Dialogfelds **Optionen** und deren Klickverhalten. Wenn diese Funktion nicht definiert ist, werden keine Schaltflächen angezeigt, und der body-Bereich der Berichtsdatei füllt das gesamte Dialogfeld aus.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array, in dem eine gerade Anzahl von Elementen gespeichert ist. Das erste Element ist ein String mit der Bezeichnung für die oberste Schaltfläche. Das zweite Element ist ein String mit JavaScript-Code, mit dem das Klickverhalten der obersten Schaltfläche bestimmt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel

Im folgenden Beispiel wird gezeigt, wie mithilfe der Funktion `commandButtons()` die Schaltflächen **OK**, **Abbrechen** und **Hilfe** definiert werden.

```
function commandButtons() {  
    return new Array("OK" , "doCommand()" , "Cancel" , "  
    "window.close()" , "Help" , "showHelp()");  
}
```

configureSettings()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Legt fest, ob die Schaltfläche **Berichtseinstellungen** im Dialogfeld **Berichte** aktiviert wird, wenn dieser Bericht ausgewählt wird.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn die Schaltfläche **Berichtseinstellungen** aktiviert werden soll, andernfalls `false`.

windowDimensions()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Legt die Abmessungen für das Dialogfeld **Parameter** fest. Wenn diese Funktion nicht definiert ist, werden die Abmessungen automatisch berechnet.

Hinweis: Verwenden Sie diese Funktion nur dann, wenn das Dialogfeld **Optionen** größer als 640 x 480 Pixel sein soll.

Argumente

platform

Der Wert des Arguments *platform* ist entweder "macintosh" oder "windows", abhängig von der verwendeten Plattform.

Rückgabewerte

Dreamweaver erwartet einen String der Form "widthInPixels,heightInPixels".

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da die Schaltflächen für **OK** und **Abbrechen** nicht mit einbezogen sind. Wenn mit den zurückgegebenen Abmessungen nicht für alle Optionen Platz vorhanden ist, werden Bildlaufleisten angezeigt.

Beispiel

Im folgenden Beispiel für die Funktion `windowDimensions()` werden die Abmessungen des Dialogfelds **Parameter** auf 648 x 520 Pixel eingestellt:

```
function windowDimensions() {  
    return "648,520";  
}
```

Kapitel 13: Tag-Bibliotheken und Tag-Editoren

Benutzer von Adobe Dreamweaver CS3 können mithilfe von Tag-Editoren neue Tags einfügen, vorhandene Tags bearbeiten und auf Referenzinformationen zu Tags zuzugreifen. Dreamweaver umfasst Editoren für die folgenden Sprachen: HTML, ASP.NET, CFML, JRun und JSP. Sie können die mit Dreamweaver gelieferten Tag-Editoren anpassen oder neue Tag-Editoren erstellen. Weiterhin können Sie den Tag-Bibliotheken neue Tags hinzufügen.

Über die Tag-Auswahl können Dreamweaver-Benutzer verfügbare Tags anzeigen und für die Verwendung im aktiven Dokument auswählen. Zu diesem Zweck greift die Tag-Auswahl auf Informationen zurück, die in den Tag-Bibliotheken gespeichert sind.

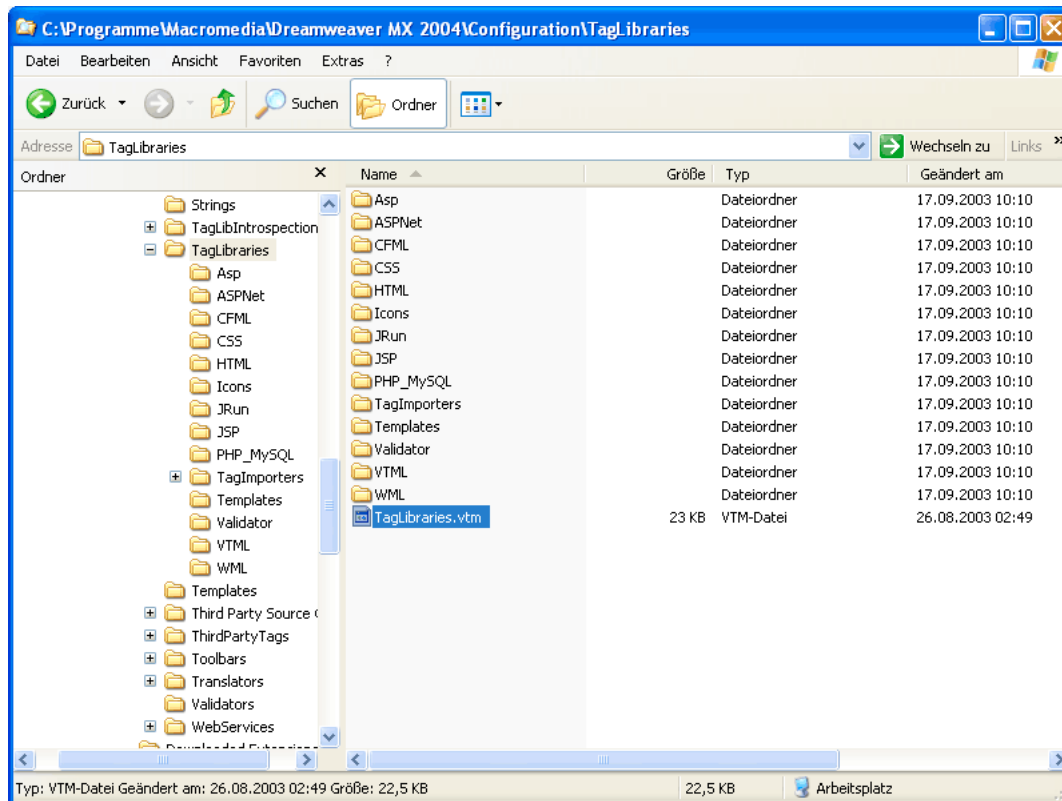
Dreamweaver speichert Informationen über die einzelnen Tags einschließlich aller Tag-Attribute in mehreren Unterverordnern des Ordners **Configuration\TagLibraries**. Bei der Bearbeitung von Tags verwenden der Tag-Editor und die Tag-Auswahl die in diesem Ordner gespeicherten Informationen. Bevor Sie benutzerdefinierte Tag-Editoren erstellen, sollten Sie mit der Struktur einer Tag-Bibliothek vertraut sein.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Tag-Bibliotheken:

Pfad	Datei	Beschreibung
Configuration\TagLibraries\	TagLibraries.vtm	Listet sämtliche installierten Tags auf.
Configuration\TagLibraries\Sprache\	Tag.vtm	Enthält Informationen über Tags (z. B. Tag-Attribute, ob das Tag über ein schließendes Tag verfügt, und Formatierungsregeln).
Configuration\TagLibraries\Sprache\	Tag-Bilddatei.gif	Optionale Datei für die Anzeige im Eigenschafteninspektor.

Das Dateiformat einer Tag-Bibliothek

Eine Tag-Bibliothek besteht aus einer einzelnen Stammdatei, genannt **TagLibraries.vtm**, in der alle installierten Tags aufgelistet werden, und je einer .vtm-Datei (VTML) für jedes Tag in der Tag-Bibliothek. Die Datei **TagLibraries.vtm** hat die Rolle eines Inhaltsverzeichnis. Sie enthält Verweise auf die VTML-Dateien der einzelnen Tags. In der folgenden Abbildung wird veranschaulicht, wie die VTML-Dateien in Dreamweaver nach Markup-Sprache organisiert werden:



Benutzer von Macromedia HomeSite von Adobe sind vermutlich bereits mit der VTML-Dateistruktur vertraut, sie sollten jedoch beachten, dass Dreamweaver VTML-Dateien nicht auf dieselbe Weise wie HomeSite verwendet. Der wichtigste Unterschied besteht darin, dass Dreamweaver über einen eigenen HTML-Renderer zur Anzeige von Erweiterungsflächen verfügt. Die VTML-Dateien werden daher beim Rendering von grafischen Benutzeroberflächen nicht verwendet.

Im folgenden Beispiel wird die Struktur der Datei **TagLibraries.vtm** dargestellt:

```
<taglibraries>
<taglibrary name="Name of tag library" doctypes="HTML,ASP-JS,ASP-VB" tagchooser="relative
  path to TagChooser.xml file" id="DWTagLibrary_html">
  <tagref name="tag name" file="relative path to tag .vtm file"/>
</taglibrary>

<taglibrary name="CFML Tags" doctypes="ColdFusion" servermodel="Cold Fusion"
  tagchooser="cfml/TagChooser.xml" id="DWTagLibrary_cfml">
  <tagref name="cfabort" file="cfml/cfabort.vtm"/>
</taglibrary>

<taglibrary name="ASP.NET Tags" doctypes="ASP.NET_CSharp,ASP.NET_VB" servermodel="ASPNet"
  prefix="<asp:" tagchooser="ASPNet/TagChooser.xml" id="DWTagLibrary_aspnet">
  <tagref name="dataset" file="aspnet/dataset.vtm" prefix="<mm:dataset"/>
</taglibrary>
</taglibraries>
```

Mit dem Tag `taglibrary` werden Tags in einer Tag-Bibliothek gruppiert. Wenn Sie Tags importieren oder neue Tags erstellen, können Sie sie in Tag-Bibliotheken gruppieren. In der Regel entspricht eine `taglibrary`-Gruppe einer Tag-Gruppe, die in einer TLD-Datei von JavaServer Pages (JSP), einer DTD-Datei (Document Type Definition) von XML, einem ASP.NET-Namespace oder in einer anderen logischen Gruppierung definiert ist.

In der folgenden Tabelle sind die Attribute für `taglibrary` aufgelistet:

Attribut	Beschreibung	Obligatorisch/optional
<code>taglibrary.name</code>	Mit diesem Namen wird die Tag-Bibliothek in der Benutzeroberfläche identifiziert.	Obligatorisch
<code>taglibrary.doctypes</code>	Gibt die Dokumenttypen an, für die diese Bibliothek aktiv ist. Wenn die Bibliothek aktiv ist, werden im Menü Codehinweise Bibliotheks-Tags angezeigt. Wegen möglicher Namenskonflikte können nicht alle Tag-Bibliotheken gleichzeitig aktiv sein (HTML- und WML-Dateien sind beispielsweise nicht kompatibel).	Obligatorisch
<code>taglibrary.prefix</code>	Wenn angegeben, weisen Tags in der Tag-Bibliothek das Format <code>taglibrary.prefix + tagref.name</code> auf. Wenn beispielsweise <code>taglibrary.prefix</code> den Wert <code>"<jrun: "</code> hat und <code>tagref.name</code> den Wert <code>"if"</code> , so hat das Tag das Format <code>"<jrun:if"</code> . Dies kann für ein bestimmtes Tag außer Kraft gesetzt werden.	Optional
<code>taglibrary.servermodel</code>	Wenn die Tags in der Tag-Bibliothek auf einem Anwendungsserver ausgeführt werden, gibt das Attribut <code>servermodel</code> das Servermodell des Tags an. Bei Client-seitigen Tags (also nicht serverbasierten Tags) wird das Attribut <code>servermodel</code> nicht angegeben. Das Attribut <code>servermodel</code> wird auch für die Funktion Zielbrowser überprüfen verwendet.	Optional
<code>taglibrary.id</code>	Dies kann ein beliebiger String sein, der sich von den Attributen <code>taglibrary.ID</code> anderer Tag-Bibliotheken in der Datei unterscheidet. Der Extension Manager verwendet das ID-Attribut, damit die MXP-Dateien neue <code>taglibrary</code> - und <code>tags</code> -Dateien in die Datei TagLibraries.vtm einfügen können.	Optional
<code>taglibrary.tagchooser</code>	Ein relativer Pfad zur Datei <code>TagChooser.xml</code> , die zu dieser Tag-Bibliothek gehört.	Optional

In der folgenden Tabelle werden die Attribute für `tagref` aufgelistet:

Attribut	Beschreibung	Obligatorisch/optional
<code>tagref.name</code>	Mit diesem Namen wird das Tag in der Benutzeroberfläche identifiziert.	Obligatorisch
<code>tagref.prefix</code>	Gibt an, wie das Tag in der Codeansicht angezeigt wird. Wenn das Attribut <code>tagref.prefix</code> verwendet wird, bestimmt es das Präfix des aktuellen Tags. Wurde das Attribut definiert, setzt es den für <code>taglibrary.prefix</code> angegebenen Wert außer Kraft.	Optional
<code>tagref.file</code>	Verweist auf die VTML-Datei des Tags.	Optional

Da das Attribut `tagref.prefix` das Attribut `taglibrary.prefix` außer Kraft setzen kann, ist die Beziehung zwischen diesen beiden Attributen möglicherweise etwas unklar. In der folgenden Tabelle wird die Beziehung zwischen den Attributen `taglibrary.prefix` und `tagref.prefix` verdeutlicht:

Wurde das Attribut <code>taglibrary.prefix</code> definiert?	Wurde das Attribut <code>tagref.prefix</code> definiert?	Ergebnis für das Tag-Präfix
Nein	Nein	<code>'<' + tagref.name</code>
Ja	Nein	<code>taglibrary.prefix + tagref.name</code>
Nein	Ja	<code>tagref.prefix</code>
Ja	Ja	<code>tagref.prefix</code>

Zur Definition von Tags verwendet Dreamweaver eine abgeänderte Version des VTML-Dateiformats. Das folgende Beispiel zeigt alle Elemente, die Dreamweaver zur Definition eines individuellen Tags verwenden muss:

```
<tag name="input" bind="value" casesensitive="no" endtag="no">
  <tagformat indentcontents="yes" formatcontents="yes" nlbeforetag nlbeforecontents=0
  nlaftercontents=0 nlaftertag=1 />
  <tagdialog file = "input.HTM"/>
  <Attribute>
    <attrib name="name"/>
    <attrib name="wrap" type="Enumerated">
      <attriboption value="off"/>
      <attriboption value="soft"/>
      <attriboption value="hard"/>
    /attrib>
    <attrib name="onFocus" casesensitive="yes"/>
    <event name="onFocus"/>
  </attributes>
</tag>
```

In der folgenden Tabelle werden die Attribute aufgelistet, mit denen Tags definiert werden:

Attribut	Beschreibung	Obligatorisch/optional
tag.bind	Wird vom Bedienfeld Bindungen verwendet. Wenn Sie ein Tag dieses Typs auswählen, gibt das Attribut bind das Standardattribut für die Datenbindung an.	Optional
tag.casesensitive	Gibt an, ob beim Tagnamen zwischen Groß- und Kleinschreibung unterschieden wird. Wenn ja, wird beim Einfügen des Tags in das Dokument des Benutzers genau die Groß- und Kleinschreibung verwendet, die in der Tag-Bibliothek angegeben ist. Wird nicht zwischen Groß- und Kleinschreibung unterschieden, wird beim Einfügen des Tags die Standardeinstellung für die Groß- oder Kleinschreibung verwendet, die im Dialogfeld Voreinstellungen auf der Registerkarte Codeformat angegeben ist. Wenn das Attribut casesensitive nicht angegeben wurde, wird davon ausgegangen, dass beim Tag nicht zwischen Groß- und Kleinschreibung unterschieden wird.	Optional
tag.endtag	Gibt an, ob das Tag über ein öffnendes und schließendes Tag verfügt. Das Tag input hat beispielsweise kein schließendes Tag, da es kein zugehöriges Tag /input gibt. Wenn das schließende Tag optional ist, sollte das Attribut ENDTAG auf Yes gesetzt werden. Geben Sie xml an, um für ein leeres Tag XML-Syntax festzulegen. Zum Beispiel wird mit <tag name="foo" endtag="xml" tagtype="empty"> Folgendes eingefügt <foo/>.	Optional
tagformat	Gibt die Formatierungsregeln für das Tag an. In älteren Dreamweaver-Versionen (vor Dreamweaver MX) wurden diese Regeln in der Datei SourceFormat.txt gespeichert.	Optional
tagformat.indentcontents	Gibt an, ob der Inhalt dieses Tags eingerückt werden soll.	Optional
tagformat.formatcontents	Gibt an, ob der Inhalt dieses Tags analysiert werden soll. Bei Tags wie SCRIPT und STYLE, die keinen HTML-Inhalt haben, ist dieses Attribut auf No eingestellt.	Optional
tagformat.nlbeforetag	Legt fest, ob vor diesem Tag ein Zeilenumbruchzeichen eingefügt wird. Der Wert 0 bedeutet nein, der Wert 1 ja.	Optional
tagformat.nlbeforecontents	Gibt an, wie viele Zeichen für Zeilenumbrüche vor dem Inhalt dieses Tags eingefügt werden sollen.	Optional
tagformat.nlaftercontents	Gibt an, wie viele Zeichen für Zeilenumbrüche nach dem Inhalt dieses Tags eingefügt werden sollen.	Optional
tagformat.nlaftertag	Legt fest, ob nach diesem Tag ein Zeilenumbruchzeichen eingefügt wird. Der Wert 0 bedeutet nein, der Wert 1 ja.	Optional

Attribut	Beschreibung	Obligatorisch/optional
attrib.name	Der Name des Attributs, wie er im Quellcode dargestellt wird.	Obligatorisch
attrib.type	Falls das Attribut <code>attrib.type</code> nicht angegeben wird, hat es den Wert "text". Folgende Werte sind möglich: TEXT – freier Textinhalt ENUMERATED – eine Liste aufgezählter Werte COLOR – ein Farbwert (Name oder Hexadezimalwert) FONT – Schriftname oder Schriftfamilie STYLE – CSS-Stilattribut CSSSTYLE – CSS-Klassenname CSSID – CSS-Klassen-ID FILEPATH – ein vollständiger Dateipfad DIRECTORY – ein Verzeichnispfad FILENAME – nur ein Dateiname RELATIVEPATH – eine relative Pfadangabe FLAG – ein ON/OFF-Attribut ohne Wert	Optional
attrib.casesensitive	Gibt an, ob beim Attributnamen zwischen Groß- und Kleinschreibung unterschieden wird. Wenn das Attribut <code>CASESENSITIVE</code> fehlt, wird beim Attributnamen nicht zwischen Groß- und Kleinschreibung unterschieden.	Optional

Hinweis: In älteren Dreamweaver-Versionen (vor Dreamweaver MX) werden die Tag-Informationen in der Datei **Configuration\TagAttributeList.txt** gespeichert.

Die Tag-Auswahl

Über die Tag-Auswahl können Sie Tags in funktionellen Gruppen anzeigen und so einfach auf häufig verwendete Tags zugreifen. Wenn Sie der Tag-Auswahl einzelne Tags oder Tag-Gruppen hinzufügen möchten, müssen Sie sie in die Tag-Bibliothek einfügen. Dazu können Sie das Dialogfeld **Tag-Bibliothek-Editor** verwenden oder eine Dreamweaver-Erweiterung installieren, die in eine MXP-Datei gepackt ist.

TagChooser.xml-Dateien

Die **TagChooser.xml**-Dateien enthalten die Metadaten für die Organisation der Tag-Gruppen, die in der Tag-Auswahl angezeigt werden. Jedes zu Dreamweaver gehörige Tag wird in einer funktionellen Gruppe gespeichert und steht in der Tag-Auswahl zur Verfügung. Sie können die **TagChooser.xml**-Dateien bearbeiten, um vorhandene Tags neu in Gruppen anzuordnen oder um neue Tags zu gruppieren. Außerdem können Sie Tags umorganisieren, indem Sie Unterkategorien erstellen, über die Benutzer ganz einfach auf die wichtigsten Tags zugreifen können.

Die **TagLibraries.vtm**-Dateien unterstützen das Attribut `taglibrary.tagchooser`, das auf die **TagChooser.xml**-Dateien verweist. Wenn Sie **TagChooser.xml**-Dateien ändern oder neu erstellen, muss das Attribut `taglibrary.tagchooser` auf das korrekte Verzeichnis verweisen, damit die Tag-Auswahl mit vollem Funktionsumfang zur Verfügung steht.

Ist das Attribut `taglibrary.tagchooser` nicht vorhanden, wird in der Tag-Auswahl die Struktur aus der Datei **TagLibraries.vtm** angezeigt.

TagChooser.xml-Dateien befinden sich im Ordner **Configuration\TagLibraries\Tag-Bibliotheksnamen**. Im folgenden Beispiel wird die Struktur der **TagChooser.xml**-Dateien veranschaulicht:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>

<taglibrary name="Friendly name for library node" desc="Description for incorporated
  reference" reference="Language[,Topic[,Subtopic]]">

  <category name="Friendly name for category node" desc="Description for incorporated
    reference" reference="Language[,Topic[,Subtopic]]" id="Unique id">

    <category name="Friendly name for subcategory node" ICON="Relative path"
      desc="Description for incorporated reference" reference="Language,Topic[,Subtopic]"
      id="Unique id">
```

```

<element name="Friendly name for list item" value='Value to pass to visual dialog
editors' desc='Description for incorporated reference'
reference="Language[,Topic[,Subtopic]]" id="Unique id"/>

... weitere in der Listenansicht anzuzeigende Elemente ...

</category>

... weitere Unterkategorien ...

</category>

... weitere Kategorien ...

</tclibrary>

```

In der folgenden Tabelle werden die Tags aufgelistet, die in den **TagChooser.xml**-Dateien verwendet werden können:

Tag	Beschreibung	Obligatorisch/optional
tclibrary	Dies ist das äußerste Tag, das die Struktur der Tag-Auswahl dieser Tag-Bibliothek umschließt.	Obligatorisch
tclibrary.name	Der Wert wird in der Strukturansichtsknoten angezeigt.	Obligatorisch
tclibrary.desc	Der Wert ist ein HTML-String, der im Bereich Tag-Info des Dialogfelds Tag-Auswahl angezeigt wird. Wenn das Attribut <code>desc</code> nicht vorhanden ist, stammen die Informationen für Tag-Info aus dem Bedienfeld Referenz . Austauschbar mit <code>tclibrary.reference</code> .	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)
tclibrary.reference	Der Wert beschreibt Sprache, Thema und Unterthema für die Anzeige im Bereich Tag-Info des Dialogfelds Tag-Auswahl . Austauschbar mit <code>tclibrary.desc</code> .	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)

Das Tag `category` steht für alle anderen Knoten in der Strukturansicht unterhalb des Knotens `tclibrary`, wie in der folgenden Tabelle gezeigt:

Tag	Beschreibung	Obligatorisch/optional
category.name	Der Wert wird im der Strukturansichtsknoten angezeigt.	Obligatorisch
category.desc	Der Wert ist ein HTML-String, der im Bereich Tag-Info des Dialogfelds Tag-Auswahl angezeigt wird. Wenn weder <code>desc</code> noch <code>reference attr</code> angegeben wird, ist der Bereich Tag-Info leer.	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)
category.reference	Der Wert beschreibt Sprache, Thema und Unterthema für die Anzeige im Bereich Tag-Info .	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)
category.icon	Der Wert ist ein relativer Pfad zu einem GIF-Symbol.	Optional
category.id	Ein beliebiger String, der sich von den Attributen <code>category.id</code> anderer Kategorien in dieser Datei unterscheidet.	Obligatorisch

In der folgenden Tabelle sind die Attribute des Tags `element` aufgeführt, das für das einzufügende Tag steht:

Attribut	Beschreibung	Obligatorisch/optional
element.name	Der Wert wird als Element in der Listenansicht angezeigt.	Obligatorisch
element.value	Ein Wert, der direkt in den Code platziert wird, oder ein Parameter, der an grafische Dialogfelder übergeben wird.	Obligatorisch

Attribut	Beschreibung	Obligatorisch/optional
<code>element.desc</code>	Der Wert ist ein HTML-String, der im integrierten Bedienfeld Referenz angezeigt wird. Wenn dieses Attribut nicht angegeben wird, zeigt das Attribut <code>reference</code> den Referenzinhalt im integrierten Bedienfeld Referenz an.	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)
<code>element.reference</code>	Bis zu drei durch Kommas voneinander getrennte Strings, die jeweils die Sprache, das Thema und das Unterthema beschreiben. Diese Informationen werden im Bedienfeld Referenz angezeigt. Der erste String ist obligatorisch. Der zweite String ist nur für das Tag <code>element</code> obligatorisch. Für die Tags <code>category</code> und <code>tblibrary</code> ist er dagegen optional. Der dritte String ist optional.	Optional (<code>desc</code> und <code>reference</code> schließen sich gegenseitig aus)
<code>element.id</code>	Ein beliebiger String, der sich von den Attributen <code>element.id</code> anderer Elemente in dieser Datei unterscheidet.	Optional

Einfaches Beispiel für die Erstellung neuer Tag-Editoren

In den Beispielen dieses Abschnitts wird veranschaulicht, mit welchen Arbeitsschritten ein neuer Tag-Editor erstellt wird. Dazu wird das hypothetische ColdFusion-Tag `cfweather` verwendet, das geschrieben wurde, um die aktuelle Temperatur aus einer Wetter-Datenbank zu extrahieren.

Die Attribute für `cfweather` werden in der folgenden Tabelle beschrieben:

Attribut	Beschreibung
<code>zip</code>	Eine fünfstellige Postleitzahl
<code>tempaturescale</code>	Eine Temperaturskala (Celsius oder Fahrenheit)

Führen Sie zum Erstellen dieses neuen Tag-Editors die folgenden Schritte aus:

- „Das Tag in der Tag-Bibliothek registrieren“ auf Seite 195
- „Eine Tag-Definitionsdatei (VTML) erstellen“ auf Seite 196
- „Eine Benutzeroberfläche für den Tag-Editor erstellen“ auf Seite 196
- „Der Tag-Auswahl ein Tag hinzufügen“ auf Seite 198

Das Tag in der Tag-Bibliothek registrieren

Damit Dreamweaver das neue Tag erkennen kann, muss es in der Datei **TagLibraries.vtm** im Ordner **Configuration\TagLibraries** angegeben werden. Wenn der Benutzer auf einem System arbeitet, das mehrere Benutzer unterstützt (wie z. B. Windows XP, Windows 2000, Windows NT oder Mac OS X), befindet sich eine weitere Datei **TagLibraries.vtm** im Konfigurationsordner des Benutzers. Diese Datei muss aktualisiert werden, da sie die Instanz ist, die Dreamweaver sucht und analysiert.

Der Speicherort des Ordners **Configuration** auf dem Computer des Benutzers richtet sich nach dem Betriebssystem.

Unter Windows 2000 und Windows XP:

```
<Laufwerk>:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Adobe\~
Dreamweaver 9\Configuration
```

Hinweis: In Windows XP befindet sich dieser Ordner möglicherweise in einem versteckten Ordner.

Unter Mac OS X:

```
<Laufwerk>:Benutzer:<Benutzername>:Library:Application Support:Adobe:~
Dreamweaver 9:Configuration
```

Wenn Dreamweaver die Datei **TagLibraries.vtm** nicht im Konfigurationsordner des Benutzers finden kann, wird sie im Konfigurationsordner von Dreamweaver gesucht.

Hinweis: Wenn Sie in einer Umgebung mit mehreren Benutzern die Kopie der Datei **TagLibraries.vtm** bearbeiten, die sich im Dreamweaver-Ordner **Configuration** befindet (also nicht die Kopie im Ordner **Configuration** des Benutzers), erkennt Dreamweaver diese Änderung nicht, da Dreamweaver die Datei **TagLibraries.vtm** im Ordner **Configuration** des Benutzers und nicht im Dreamweaver-Ordner **Configuration** analysiert.

Da es sich bei `cfweather` um ein ColdFusion-Tag handelt, ist bereits eine Gruppe in der Tag-Bibliothek vorhanden, die Sie zum Registrieren des Tags `cfweather` verwenden können.

So registrieren Sie das Tag:

- 1 Öffnen Sie die Datei **TagLibraries.vtm** in einem Texteditor.
- 2 Navigieren Sie durch die vorhandenen Tag-Bibliotheken, bis Sie die CFML-Tag-Gruppe `taglibrary` gefunden haben.
- 3 Fügen Sie ein neues Tag-Referenzelement hinzu, wie im folgenden Beispiel gezeigt:

```
<tagref name="cfweather" file="cfml/cfweather.vtm"/>
```

- 4 Speichern Sie die Datei.

Das Tag ist nun in der Tag-Bibliothek registriert und verfügt über einen Dateizeiger auf die Tag-Definitionsdatei **cfweather.vtm**.

Eine Tag-Definitionsdatei (VTML) erstellen

Wenn ein Benutzer in der Tag-Auswahl oder in einem Tag-Editor ein registriertes Tag auswählt, sucht Dreamweaver eine entsprechende VTML-Datei mit der Tag-Definition.

So erstellen Sie eine Tag-Definitionsdatei:

- 1 Erstellen Sie in einem Texteditor eine Datei mit folgendem Inhalt:

```
<TAG NAME="cfweather" endtag="no">
    <TAGFORMAT NLBEFORETAG="1" NLAFTERTAG="1"/>
    <TAGDIALOG FILE="cfweather.htm"/>

    <ATTRIBUTES>
        <ATTRIB NAME="zip" TYPE="TEXT"/>
        <ATTRIB NAME="tempaturescale" TYPE="ENUMERATED">
            <ATTRIBOPTION VALUE="Celsius"/>
            <ATTRIBOPTION VALUE="Fahrenheit"/>
        </ATTRIB>
    </ATTRIBUTES>
</TAG>
```

- 2 Speichern Sie die Datei **cfweather.vtm** im Ordner **Configuration\Taglibraries\CFML**.

Mithilfe der Tag-Definitionsdatei kann Dreamweaver folgende Funktionen für das Tag `cfweather` ausführen: Codehinweise anzeigen, Code vervollständigen und das Tag formatieren.

Eine Benutzeroberfläche für den Tag-Editor erstellen

So erstellen Sie die Benutzeroberfläche für den Tag-Editor `cfweather`:

- 1 Speichern Sie die Datei **cfweather.vtm** im Ordner **Configuration\Taglibraries\CFML**:

```
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//dialog">
<html>
<head>
<title>CFWEATHER</title>
<script src="../../Shared/Common/Scripts/dwscripts.js"></script>
<script src="../../Shared/Common/Scripts/ListControlClass.js"></script>
<script src="../../Shared/Common/Scripts/tagDialogsCmn.js"></script>
<script>
```

```

/***** GLOBAL VARS *****/
var TEMPATURESCALELIST; // Temperaturlistkontrolle (initializeUI() initialisiert())
var theUIObjects; // Array an UI-Objekten, die von allgemeinen API-Funktionen verwendet werden

/*****

// definierte inspectTag()-API-Funktion (wird von allen Tag-Editoren benötigt)
function inspectTag(tagNodeObj)
{
    // Aufruf einer allgemeinen Bibliotheksversion von
    // in tagDialogCmns.js definiertem applyTagCommon (beachten Sie, dass sie eingebunden wurde)
    // Weitere Information zu dieser Funktion sind in den Kommentaren
    // zu inspectTagCommon in tagDialogCmn.js erhältlich.
    tagDialog.inspectTagCommon(tagNodeObj, theUIObjects);
}
function applyTag(tagNodeObj)
{
    // Aufruf einer allgemeinen Bibliotheksversion von
    // in tagDialogCmns.js definiertem applyTagCommon (beachten Sie, dass sie eingebunden wurde)
    // Weitere Information zu dieser Funktion sind in den Kommentaren
    // zu applyTagCommon in tagDialogCmn.js erhältlich.
    tagDialog.applyTagCommon(tagNodeObj, theUIObjects);
}
function initializeUI()
{
    // Definition von zwei Arrays für die Werte und Anzeigeüberschriften für die Liste
    control
    var theTemperatureScaleCap = new Array("celsius","fahrenheit");
    var theTemperatureScaleVal = new Array("celsius","fahrenheit");

    // Instantiieren einer neuen Liste
    TEMPATURESCALELIST = new ListControl("thetempaturescale");

    // Hinzufügen der Dropdownliste tempaturescalelist in die uiobjects
    theUIObjects0= new Array(TEMPATURESCALELIST);

    // Aufruf allgemeiner in tagDialogCmn.js definierter populateDropDownList-Funktion zum
    // Auffüllen der Liste tempaturescale
    tagDialog.populateDropDownList(TEMPATURESCALELIST, theTemperatureScaleCap,
    theTemperatureScaleVal, 1);
}
</script>

</head>
<body onLoad="initializeUI()">
<div name="General">
    <table border="0" cellspacing="4">
        <tr>
            <td valign="baseline" align="right" nowrap="nowrap">Zip Code: </td>
            <td nowrap="nowrap">
                <input type="text" id="attr:cfargument:zip" name="thezip" attname="zip"
                style="width:100px"0/>&nbsp;
            </td>
        </tr>
        <tr>
            <td valign="baseline" align="right" nowrap="nowrap">Type: </td>
            <td nowrap="nowrap">
                <select name="thetempaturescale" id="attr:cfargument:tempaturescale"
                attname="tempaturescale" editable="false" style="width:200px">
                </select>
            </td>
        </tr>
    </table>

```

```
</div>
</body>
</html>
```

2 Überprüfen Sie anhand der folgenden Schritte, ob der Tag-Editor korrekt funktioniert:

- a Starten Sie Dreamweaver.
- b Geben Sie in der Codeansicht `cfweather` ein.
- c Klicken Sie mit der rechten Maustaste auf das Tag.
- d Wählen Sie im Kontextmenü **Tag bearbeiten** `cfweather`.

Wenn der Tag-Editor nun gestartet wird, wurde das Tag erfolgreich erstellt.

Der Tag-Auswahl ein Tag hinzufügen

So fügen Sie der Tag-Auswahl das Tag `cfweather` hinzu:

1 Bearbeiten Sie die Datei **TagChooser.xml**, indem Sie ihr eine neue Kategorie namens „Third Party Tags“ hinzufügen. Diese Kategorie enthält das Tag `cfweather`, wie im folgenden Beispiel gezeigt:

```
<category name="Third Party Tags" icon="icons/Elements.gif" reference='CFML'>
  <element name="cfweather" value='cfweather zip="" temperaturescale="fahrenheit">' />
</category>
```

***Hinweis:** In einer Umgebung mit mehreren Benutzern befindet sich eine Kopie der Datei **TagChooser.xml** im Ordner **Configuration** auf dem Computer des Benutzers. Weitere Informationen über Umgebungen mit mehreren Benutzern finden Sie unter „Das Tag in der Tag-Bibliothek registrieren“ auf Seite 195.*

2 Führen Sie die folgenden Schritte aus, um zu überprüfen, ob das Tag `cfweather` nun in der Tag-Auswahl enthalten ist:

- a Wählen Sie **Einfügen > Tag**.
- b Erweitern Sie die Gruppe CFML-Tags.
- c Wählen Sie unten in der Tag-Auswahl die Gruppe „Third Party Tags“ aus. Das Tag `cfweather` wird im Listenfeld rechts angezeigt.
- d Wählen Sie `cfweather` aus, und klicken Sie auf die Schaltfläche **Einfügen**.

Nun wird der Tag-Editor eingeblendet.

Die API-Funktionen für den Tag-Editor

Zur Erstellung eines neuen Tag-Editors müssen Sie eine Implementierung für die Funktionen `inspectTag()`, `validateTag()` und `applyTag()` bereitstellen. Ein Beispiel für eine Implementierung finden Sie unter „Eine Benutzeroberfläche für den Tag-Editor erstellen“ auf Seite 196.

inspectTag()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Die Funktion wird beim ersten Aufrufen des Tag-Editors aufgerufen. Als Argument erhält die Funktion das Tag, das der Benutzer bearbeitet und das als `dom`-Objekt ausgedrückt wird. Die Funktion extrahiert Attributwerte aus dem Tag, das bearbeitet wird, und verwendet diese Werte, um Formularelemente im Tag-Editor zu initialisieren.

Argumente

Tag

- Das Argument `tag` ist der DOM-Knoten des bearbeiteten Tags.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Angenommen, der Benutzer bearbeitet das folgende Tag:

```
<crfweather zip = "94065"/>
```

Wenn der Editor ein Textfeld für die Bearbeitung des Attributs `zip` enthält, muss die Funktion das Formularelement initialisieren, damit kein leeres Feld, sondern ein Textfeld mit der Postleitzahl angezeigt wird.

Der folgende Code bewirkt die Initialisierung:

```
function inspectTag(tag)
{
    document.forms[0].zip.value = tag.zip
}
```

validateTag()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Wenn der Benutzer auf einen Knoten in der Strukturansicht oder auf **OK** klickt, überprüft die Funktion die Gültigkeit der Daten, die in die derzeit angezeigten HTML-Formularelemente eingegeben werden.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn die Eingabe für die HTML-Formularelemente gültig sind, `false`, wenn die Eingabewerte nicht gültig sind.

Beispiel

Der Benutzer erstellt eine Tabelle und gibt dabei eine negative Ganzzahl für die Anzahl der Tabellenzeilen ein. Die Funktion `validateTag()` erkennt, dass die Eingabe gültig ist, zeigt eine Warnmeldung an und gibt den Wert `false` zurück.

applyTag()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Wenn der Benutzer auf **OK** klickt, ruft Dreamweaver die Funktion `validateTag()` auf. Wenn die Funktion `validateTag()` den Wert `true` zurückgibt, ruft Dreamweaver diese Funktion auf und übergibt das `dom`-Objekt, das für das aktuelle Tag steht (das heißt das Tag, das bearbeitet wird). Die Funktion liest die Werte aus den Formularelementen und schreibt sie in das `dom`-Objekt.

Argumente

`tag`

- Das Argument `tag` ist der DOM-Knoten des bearbeiteten Tags.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Um beim Beispiel `cfweather` zu bleiben: Wenn der Benutzer im folgenden Beispiel die Postleitzahl von 94065 zu 53402 ändert, muss das `dom`-Objekt aktualisiert werden, damit die neue Postleitzahl im Dokument des Benutzers angezeigt wird:

```
function applyTag(tag)
{
    tag.zip = document.forms[0].zip.value
}
```

Kapitel 14: Eigenschafteninspektoren

Der Eigenschafteninspektor ist wahrscheinlich das bekannteste schwebende Bedienfeld der Benutzeroberfläche von Adobe Dreamweaver CS3. Er ist unverzichtbar für das Definieren, Prüfen und Ändern des Namens, der Größe, der Darstellung und anderer Attribute der Auswahl sowie für das Starten interner und externer Editoren für das jeweils ausgewählte Element.

Dreamweaver verfügt über mehrere Oberflächen für den Eigenschafteninspektor, mit denen Sie die Eigenschaften für viele HTML-Standardtags definieren können. Da diese integrierten Inspektoren Teil des Kernprogramms von Dreamweaver sind, gibt es für sie keine separaten Eigenschafteninspektor-Dateien im Ordner **Configuration**. Durch benutzerspezifische Dateien für die Eigenschafteninspektoren können Sie diese Oberflächen jedoch austauschen oder neue Oberflächen zum Prüfen benutzerspezifischer Tags erstellen. Benutzerspezifische Eigenschafteninspektor-Dateien befinden sich im Ordner **Configuration\Inspectors** des Dreamweaver-Anwendungsordners.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Eigenschafteninspektoren:

Pfad	Datei	Beschreibung
Configuration\Inspectors\	Name_des_Eigenschafteninspektors.htm	Definiert die Benutzeroberfläche (UI) des Eigenschafteninspektors.
Configuration\Inspectors\	Name_des_Eigenschafteninspektors.js	Enthält die für den Eigenschafteninspektor erforderlichen Funktionen.
Configuration\Inspectors\	Tag-Bilddatei.gif	Optionale Datei für die Anzeige im Eigenschafteninspektor.

Eigenschafteninspektor-Dateien

Die HTML-Dateien für den Eigenschafteninspektor muss, zusätzlich zum doctype-Kommentar, direkt vor dem öffnenden HTML-Tag einen Kommentar in folgendem Format aufweisen:

```
<!-- tag:Servermodell:Tag-Name_oder_Schlüsselwort,priority:1_bis_10,selection:-
exact_oder_within,hline,vline, Servermodell-->
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWExtension layout-engine5.0//pi">
```

Dieser Kommentar enthält folgende Elemente:

- Das Element *Servermodell* gibt an, dass Dreamweaver diesen Eigenschafteninspektor nur dann laden soll, wenn das angegebene Servermodell aktiv ist.
- Das Element *Tag-Name_oder_Schlüsselwort* enthält das zu prüfende Tag oder eines der folgenden Schlüsselwörter: **COMMENT** (für Kommentare), **LOCKED** (für gesperrte Bereiche) oder **ASP** (für ASP-Tags).
- Das Element *1_bis_10* stellt die Priorität der Eigenschafteninspektor-Datei dar: 1 bedeutet, dass dieser Inspektor nur verwendet werden soll, wenn keine anderen Inspektoren für den ausgewählten Bereich verwendet werden können. 10 bedeutet, dass dieser Inspektor Vorrang vor allen anderen Inspektoren hat, die die Auswahl untersuchen können.
- Das Element *exact_oder_within* gibt an, ob die aktuelle Auswahl innerhalb eines Tags liegen darf (*within*) oder ein Tag (*exact*) enthalten muss.
- Das Element *hline* legt fest, dass eine waagerechte graue Linie zwischen der oberen und der unteren Hälfte in der erweiterten Darstellung des Inspektors angezeigt werden soll.
- Das Element *vline* (optional) gibt an, dass eine senkrechte graue Linie zwischen dem Tag-Namensfeld und dem Rest der Eigenschaften im Inspektor angezeigt werden soll.

- Das Element *Servermodell* (optional) legt das Servermodell des Eigenschafteninspektors fest. Wenn das Servermodell des Eigenschafteninspektors nicht mit dem Servermodell des Dokuments identisch ist, verwendet Dreamweaver den Eigenschafteninspektor nicht, um die Eigenschaften der aktuellen Auswahl anzuzeigen. Wenn beispielsweise das Servermodell des Dokuments ColdFusion ist, dasjenige des Eigenschafteninspektors aber ASP, dann verwendet Dreamweaver diesen Eigenschafteninspektor für eine Auswahl im Dokument nicht.

Der folgende Kommentar würde sich beispielsweise für einen Inspektor eignen, der das Tag `happy` prüfen soll:

```
<!-- tag:happy, priority:8,selection:exact,hline,vline,serverModel:ASP -->
```

In einigen Fällen möchten Sie möglicherweise angeben, dass Ihre Erweiterung nur Dreamweaver-Erweiterungsrendering verwenden soll (und nicht die frühere Rendering-Engine). In diesem Fall fügen Sie die folgende Zeile direkt vor dem Tag-Kommentar hinzu (siehe folgendes Beispiel):

```
<!--DOCTYPE HTML SYSTEM "-//Macromedia//DWExtension layout-engine 5.0//pi"-->
```

Der body-Bereich einer Eigenschafteninspektor-Datei enthält ein HTML-Formular. In Dreamweaver wird der Formularinhalt nicht in einem Dialogfeld angezeigt. Stattdessen wird das Formular verwendet, um Eingabebereiche und Eigenschafteninspektor-Layouts zu definieren.

Der head-Abschnitt einer Eigenschafteninspektor-Datei enthält JavaScript-Funktionen oder einen Verweis auf die JavaScript-Datei bzw. -Dateien.

Funktionsweise von Eigenschafteninspektor-Dateien

Beim Start liest Dreamweaver die erste Zeile aller Dateien des Typs HTM und HTML aus dem Ordner **Configuration\Inspectors**. Dabei sucht Dreamweaver nach dem Kommentar-String für Typ, Priorität und Auswahltyp des Eigenschafteninspektors. Dateien ohne diesen Kommentar in der ersten Zeile werden ignoriert.

Wenn der Benutzer in Dreamweaver eine Auswahl vornimmt oder die Einfügemarke an eine andere Stelle verschiebt, werden die folgenden Ereignisse ausgelöst:

- 1 Dreamweaver sucht nach Inspektoren mit dem Auswahltyp `within`.
- 2 Falls es Inspektoren des Typs `within` gibt, durchsucht Dreamweaver die Dokumentstruktur für das ausgewählte Tag, um festzustellen, ob es Inspektoren für Tags gibt, die den ausgewählten Bereich einschließen. Wenn es keine Inspektoren des Typs `within` gibt, sucht Dreamweaver nach Inspektoren mit dem Auswahltyp `exact`.
- 3 Für das erste gefundene Tag, für das es mindestens einen Inspektor gibt, ruft Dreamweaver die Inspektorfunktion `canInspectSelection()` auf. Falls diese Funktion den Wert `false` zurückgibt, wird der Inspektor von Dreamweaver nicht zur Prüfung des ausgewählten Bereichs herangezogen.
- 4 Wenn nach dem Aufruf der Funktion `canInspectSelection()` mehr als ein potenzieller Inspektor verbleibt, sortiert Dreamweaver die in Frage kommenden Inspektoren nach Priorität.
- 5 Wenn mehrere potenzielle Inspektoren die gleiche Priorität haben, wählt Dreamweaver einen Inspektor alphabetisch nach Namen aus.
- 6 Der ausgewählte Inspektor wird im schwebenden Bedienfeld des Eigenschafteninspektors angezeigt. Wenn in der Eigenschafteninspektor-Datei die Funktion `displayHelp()` definiert ist, wird ein kleines Fragezeichen (?) in der rechten oberen Ecke des Inspektors angezeigt.
- 7 Dreamweaver ruft die Funktion `inspectSelection()` auf, um Informationen zur aktuellen Auswahl zu sammeln und in den Inspektorfeldern entsprechende Einträge vorzunehmen.
- 8 Die an die Felder des Eigenschafteninspektor angehängten Event Handler werden ausgeführt, sobald der Benutzer auf sie stößt. (So kann es beispielsweise ein Ereignis des Typs `onBlur` geben, das die Funktion `setAttribute()` aufruft, um ein Attribut für den gerade vom Benutzer eingegebenen Wert zu setzen.)

Einfaches Beispiel für einen Eigenschafteninspektor

Der folgende Eigenschafteninspektor prüft das `marquee`-Tag, das nur in Microsoft Internet Explorer vorkommt. In diesem Beispiel kann der Wert für das Attribut `direction` im Eigenschafteninspektor festgelegt werden. Verwenden Sie dieses Beispiel als Modell, wenn Sie die Werte der anderen Attribute des `marquee`-Tags festlegen möchten.

Führen Sie zum Erstellen der Erweiterung die folgenden Schritte aus:

- „Benutzeroberfläche erstellen“ auf Seite 203
- „JavaScript-Code schreiben“ auf Seite 204
- „Das Bild erstellen“ auf Seite 206
- „Eigenschafteninspektor testen“ auf Seite 206

Benutzeroberfläche erstellen

Sie erstellen eine HTML-Datei mit einem Formular, das im Eigenschafteninspektor angezeigt wird.

So erstellen Sie die Benutzeroberfläche:

- 1 Erstellen Sie eine neue, leere Datei.
- 2 Fügen Sie wie folgt als erste Zeile den Kommentar in die Datei ein, der den Eigenschafteninspektor definiert:

```
<!-- tag:MARQUEE,priority:9,selection:exact,vline,hline -->
```

- 3 Zur Angabe des Dokumenttitels und der JavaScript-Datei, die Sie noch erstellen, fügen Sie nach dem Kommentar Folgendes ein:

```
<HTML>
<HEAD>
<TITLE>Marquee Inspector</TITLE>
<SCRIPT src="marquee.js"></SCRIPT>
</HEAD>
<BODY>

</BODY>
</HTML>
```

4 Fügen Sie Folgendes zwischen das öffnende und das schließende `body`-Tag ein, um festzulegen, was im Eigenschafteninspektor angezeigt wird:

```
<!-- Geben Sie das im Eigenschafteninspektor anzuzeigende Bild an -->
<SPAN ID="image" STYLE="position:absolute; width:23px; height:17px; 
    z-index:16; left: 3px; top: 2px">
    <IMG SRC="marquee.png" WIDTH="36" HEIGHT="36" NAME="marqueeImage">
</SPAN>
<SPAN ID="label" STYLE="position:absolute; width:23px; height:17px; 
    z-index:16; left: 44px; top: 5px">Marquee</SPAN>

<!-- Wenn sich die Formularfelder in unterschiedlichen AP-Elementen befinden, müssen Sie 
    in jedem AP-Element ein separates Formular erstellen und wie gezeigt in 
    den Funktionen inspectSelection() und setInterjectionTag(). -->

<SPAN ID="topLayer" STYLE="position:absolute; z-index:1; left: 125px; 
    top: 3px; width: 431px; height: 32px">
<FORM NAME="topLayerForm">
    <TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
        <TR>
            <TD VALIGN="baseline" ALIGN="right">Direction:</TD>
            <TD VALIGN="baseline" ALIGN="right">
                <SELECT NAME="marqDirection" STYLE="width:86"
                    onChange="setMarqueeTag()" >
                    <OPTION VALUE="left">Left</OPTION>
                    <OPTION VALUE="right">Right</OPTION>
                </SELECT>
            </TD>
        </TR>
    </TABLE>
</FORM>
</SPAN>
```

5 Speichern Sie die Datei unter dem Namen **marquee.htm** im Ordner **Configuration\Inspectors**.

JavaScript-Code schreiben

Sie müssen JavaScript-Funktionen hinzufügen, um die Prüfmöglichkeit der Auswahl zu gewährleisten, die Auswahl zu prüfen und die passenden Werte in den Eigenschafteninspektor einzugeben.

So schreiben Sie den JavaScript-Code:

1 Erstellen Sie eine neue leere Datei.

2 Um festzulegen, dass der Eigenschafteninspektor angezeigt wird, sobald eine Auswahl das `marquee`-Tag enthält, fügen Sie die folgende Funktion hinzu:

```
function canInspectSelection(){
    return true;
}
```

3 Zum Aktualisieren des Werts des `direction`-Attributs, der im Textfeld angezeigt wird, fügen Sie am Dateiende die folgende Funktion hinzu:

```

function inspectSelection() {
    // Das DOM des aktuellen Dokuments abrufen.
    var theDOM = dw.getDocumentDOM();
    // den ausgewählten Knoten abrufen-
    var theObj = theDOM.getSelectedNode();

    // Den Wert des Attributs DIRECTION für das Tag MARQUEE abrufen.
    var theDirection = theObj.getAttribute('direction');

    // Initialisieren einer Variable für das Attribut DIRECTION auf -1.
    // Wird verwendet, um den Menüindex zu speichern, der
    // dem Wert des Attributs entspricht.
    // var typeIndex = -1;
    var directionIndex = -1;

    // Wenn es ein DIRECTION-Attribut gäbe...
    if (theDirection){
        // Hat DIRECTION den Wert "left", wird typeIndex auf 0 gesetzt.
        if (theDirection.toLowerCase() == "left"){
            directionIndex = 0;
        }
        // Hat DIRECTION den Wert "right", wird typeIndex auf 1 gesetzt.
        }else if (theDirection.toLowerCase() == "right"){
            directionIndex = 1;
        }
    }

    // Wäre der Wert des Attributs DIRECTION "left"
    // oder "right", wird in der Benutzeroberfläche
    // die entsprechende Option aus dem Popupmenü gewählt.
    if (directionIndex != -1){
        document.topLayer.document.topLayerForm.marqDirection.selectedIndex =
        directionIndex;
    }
}

```

4 Zum Abrufen der aktuellen Auswahl, und damit der Wert des Attributs `direction` im Textfeld des Eigenschafteninspektors angezeigt wird, fügen Sie am Dateiende die folgende Funktion hinzu:

```

function setMarqueeTag(){
    // Das DOM des aktuellen Dokuments abrufen.
    var theDOM = dw.getDocumentDOM();
    // den ausgewählten Knoten abrufen-
    var theObj = theDOM.getSelectedNode();

    // Den Index der im Popupmenü ausgewählten Option abrufen
    // in der Benutzeroberfläche.
    var directionIndex = ~
        document.topLayer.document.topLayerForm.marqDirection.selectedIndex;
    // Den Wert der im Popupmenü ausgewählten Option abrufen
    // in der Benutzeroberfläche.
    var theDirection = ~
        document.topLayer.document.topLayerForm.marqDirection.~
        options[directionIndex].value;

    // Der Wert des Attributs direction wird auf theDirection gesetzt.
    theObj.setAttribute('direction',theDirection);
}

```

5 Speichern Sie die Datei unter dem Namen **marquee.js** im Ordner **Configuration\Inspectors**.

Das Bild erstellen

Sie haben die Möglichkeit, ein Bild zu erstellen, das im Eigenschafteninspektor angezeigt werden soll.

So erstellen Sie die Grafik:

- 1 Erstellen Sie eine Grafik mit 36 Pixel Breite und 36 Pixel Höhe.
- 2 Speichern Sie das Bild als **marquee.gif** im Ordner **Configuration\Inspectors**.

Im Allgemeinen können Sie Bilder für Eigenschafteninspektoren in jedem beliebigen von Dreamweaver unterstützten Format speichern.

Eigenschafteninspektor testen

Abschließend können Sie den Eigenschafteninspektor testen.

So testen Sie den Eigenschafteninspektor:

- 1 Starten Sie Dreamweaver neu.
- 2 Erstellen Sie eine neue HTML-Seite oder öffnen Sie eine vorhandene.
- 3 Fügen Sie Folgendes in den body-Abschnitt der Seite ein:

```
<MARQUEE></MARQUEE>
```

- 4 Heben Sie den soeben eingefügten Text hervor.

Der Eigenschafteninspektor, den Sie für das `marquee`-Tag erstellt haben, wird angezeigt.

- 5 Geben Sie einen Wert für das Richtungsattribut in den Eigenschafteninspektor ein.

Das Tag auf der Seite enthält nun das Richtungsattribut und den in den Eigenschafteninspektor eingegebenen Wert.

Die API-Funktionen für den Eigenschafteninspektor

Zwei der API-Funktionen des Eigenschafteninspektors (`canInspectSelection()` und `inspectSelection()`) sind erforderlich.

canInspectSelection()

Beschreibung

Legt fest, ob der Eigenschafteninspektor für die aktuelle Auswahl zulässig ist.

Argumente

Keine

Hinweis: Verwenden Sie `dom.getSelectedNode()`, um die aktuelle Auswahl als JavaScript-Objekt zu erhalten. (Weitere Informationen zu `dom.getSelectedNode()` finden Sie im Dreamweaver API-Referenzhandbuch).

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true` wenn der Inspektor die aktuelle Auswahl untersuchen kann, andernfalls `false`.

Beispiel

Die folgende Instanz der Funktion `canInspectSelection()` gibt `true` zurück, sofern die Auswahl das Attribut `CLASSID` enthält und der Attributwert `"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"` lautet (Class-ID für Adobe Flash Player):

```
function canInspectSelection(){3
    var theDOM = dw.getDocumentDOM();
    var theObj = theDOM.getSelectedNode();
    return (theObj.nodeType == Node.ELEMENT_NODE && ~
        theObj.hasAttribute("classid") && ~
        theObj.getAttribute("classid").toLowerCase() == ~
        "clsid:D27CDB6E-AE6D-11cf-96B8-444553540000");
}
```

displayHelp()

Beschreibung

Wenn die Funktion definiert ist, wird in der rechten oberen Ecke des Eigenschafteninspektors ein kleines Fragezeichen (?) angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf das Fragezeichen klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Mit dem folgenden Beispiel von `displayHelp()` wird eine Datei in einem Browserfenster geöffnet. Die Datei beschreibt die Felder im Eigenschafteninspektor.

```
function displayHelp() {
    dw.browseDocument('http://www.hooha.com/dw/inspectors/inspHelp.html');
}
```

inspectSelection()

Beschreibung

Aktualisiert den Inhalt von Textfeldern, basierend auf den Attributen der aktuellen Auswahl.

Argumente

maxOderMin

- Das Argument *maxOrMin* ist entweder `max` oder `min`, je nachdem, ob der Inspektor in der erweiterten oder minimalen Darstellung angezeigt wird.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Die folgende Instanz der Funktion `inspectSelection()` ruft den Wert des Attributs `content` ab und schreibt ihn in das Formularfeld `keywords`:

```
function inspectSelection(){
    var dom = dreamweaver.getDocumentDOM();
    var theObj = dom.getSelectedNode();
    document.forms[0].keywords.value = theObj.getAttribute("content");
}
```


Kapitel 15: Schwebende Bedienfelder

Sie können beliebige schwebende Bedienfelder oder Inspektoren erstellen, ohne an die Größen- und Layoutbeschränkungen von Eigenschafteninspektoren gebunden zu sein.

Obwohl in der Regel die Eigenschaften der aktuellen Auswahl mit dem benutzerdefinierten Eigenschafteninspektor festgelegt werden, können Sie mit schwebenden Bedienfeldern Informationen zum gesamten Dokument oder zu mehreren Auswahlabschnitten flexibler und umfassender anzeigen lassen.

Die benutzerspezifischen Dateien für schwebende Bedienfelder sind HTML-Dateien. Sie befinden sich im Ordner **Configuration\Floaters** des Anwendungsordners von Adobe Dreamweaver CS3. Der `body`-Teil einer Datei für ein schwebendes Bedienfeld enthält ein HTML-Formular. An Formularelemente angehängte Ereignisprozeduren können JavaScript-Code aufrufen, mit dem das aktuelle Dokument beliebig bearbeitet werden kann.

Im Lieferumfang von Dreamweaver sind mehrere schwebende Bedienfelder enthalten, die Sie über das Menü **Fenster** aufrufen können. (Diese integrierten Bedienfelder gehören zum Kernprogramm von Dreamweaver. Es wurden dafür keine entsprechenden Dateien im Ordner **Configuration\Floaters** angelegt.)

Sie können eigene Bedienfelder erstellen und sie dem Menü **Fenster** hinzufügen. Weitere Informationen zum Hinzufügen von Elementen zum Menüsystem finden Sie in „Menüs und Menübefehle“ auf Seite 133.

In der folgenden Tabelle sind die Dateien zum Erstellen von schwebenden Bedienfeldern aufgeführt:

Pfad	Datei	Beschreibung
Configuration\Floaters\	Bedienfeld-name.htm	Gibt den Text an, der in der Titelleiste des schwebenden Bedienfelds angezeigt wird, definiert das schwebende Bedienfeld und enthält die erforderlichen JavaScript-Funktionen.
Configuration\Menus\	menus.xml	Fügt einem Menü einen Befehl hinzu.

Funktionsweise von schwebenden Bedienfeldern

Benutzerdefinierte schwebende Bedienfelder lassen sich wie die in Dreamweaver integrierten schwebenden Bedienfelder verschieben, in der Größe verändern und in Gruppen zusammenfassen. Benutzerdefinierte schwebende Bedienfelder weichen jedoch in folgender Hinsicht von den in Dreamweaver integrierten schwebenden Bedienfeldern ab:

- Benutzerdefinierte schwebende Bedienfelder werden immer in Grau dargestellt. Das Setzen des Attributs `bgcolor` im `body`-Tag hat keinen Einfluss auf die Farbe.
- Alle benutzerdefinierten schwebenden Bedienfelder können entweder im Vordergrund des aktiven Dokumentfensters oder im Hintergrund dargestellt werden (falls sie inaktiv sind). Dieses Verhalten wird durch die Einstellung der Option **Alle anderen Fenster** in den Voreinstellungen für schwebende Bedienfelder eingestellt.

Dateien für schwebende Bedienfelder unterscheiden sich in einigen Aspekten von anderen Erweiterungen. Anders als andere Erweiterungsdateien werden Dateien für schwebende Bedienfelder beim Start nicht in den Speicher geladen, es sei denn, die schwebenden Bedienfelder waren beim letzten Beenden von Dreamweaver sichtbar. Wenn schwebende Bedienfelder beim Beenden von Dreamweaver nicht sichtbar waren, werden die entsprechenden Dateien nur dann geladen, falls eine der folgenden Funktionen sie anfordert: `dreamweaver.getFloaterVisibility()`, `dreamweaver.setFloaterVisibility()` oder `dreamweaver.toggleFloater()`. Weitere Informationen über diese Funktionen finden Sie im *Dreamweaver API-Referenzhandbuch*.

Wenn eine der Dateien im Ordner **Configuration** die Funktionen `dw.getFloaterVisibility(floaterName)`, `dw.setFloaterVisibility(floaterName)` oder `dw.toggleFloater(floaterName)` aufruft, werden folgende Ereignisse ausgelöst:

- 1 Wenn `floaterName` nicht einer der reservierten Namen für schwebende Bedienfelder ist, wird im Ordner **Configuration\Floaters** nach einer Datei mit dem Namen `floaterName.htm` gesucht. (Eine vollständige Liste der reservierten Namen finden Sie im Abschnitt zur Funktion `dreamweaver.getFloaterVisibility()` des *Dreamweaver API-Referenzhandbuchs*. Wenn die Datei `floaterName.htm` nicht gefunden werden konnte, sucht Dreamweaver die Datei `floaterName.html`. Wenn auch diese Suche ergebnislos bleibt, wird nichts ausgeführt.
- 2 Beim ersten Laden des schwebenden Bedienfelds wird die Funktion `initialPosition()` aufgerufen (falls definiert), um die Standardposition des schwebenden Bedienfelds auf dem Bildschirm zu bestimmen. Außerdem wird mit der Funktion `initialTabs()` (falls definiert) die Standard-Registergruppe bestimmt.
- 3 Die Funktionen `selectionChanged()` und `documentEdited()` werden aufgerufen, weil sich aller Wahrscheinlichkeit nach Änderungen ergeben haben, während das schwebende Bedienfeld ausgeblendet war.
- 4 Wenn das schwebende Bedienfeld sichtbar ist, werden folgende Ereignisse ausgelöst:
 - Bei Änderung der Auswahl wird die Funktion `selectionChanged()` aufgerufen (falls definiert).
 - Wenn der Benutzer Änderungen am Dokument vornimmt, wird die Funktion `documentEdited()` aufgerufen (falls definiert).
 - Die an die Felder des schwebenden Bedienfelds angehängten Ereignisprozeduren werden ausgeführt, sobald der Benutzer sie auslöst. (Beispielsweise würde eine Schaltfläche mit einer Ereignisprozedur des Typs `onClick`, die die Funktion `dw.getDocumentDOM().body.innerHTML = ''` ausführt, beim Klicken den gesamten Bereich zwischen den `body`-Tags im Dokument löschen.)

Schwebende Bedienfelder unterstützen zwei spezielle Ereignisse im `body`-Tag: `onShow()` und `onHide()`.

- 5 Beim Beenden von Dreamweaver werden die aktuelle Sichtbarkeit, Position und Registergruppe des schwebenden Bedienfelds gespeichert. Beim nächsten Start von Dreamweaver werden für alle schwebenden Bedienfelder, die beim letzten Beenden sichtbar waren, die entsprechenden Dateien geladen. Die schwebenden Bedienfelder werden in der zuletzt gültigen Position und Registergruppe angezeigt.

Einfaches Beispiel für schwebende Bedienfelder

In diesem Beispiel erstellen Sie die Erweiterung **Script Editor** zum Erstellen von schwebenden Bedienfeldern, mit denen der einem ausgewählten Skriptsymbol zugrunde liegende JavaScript-Code in der Entwurfsansicht angezeigt wird. **Script Editor** zeigt den JavaScript-Code im Element `textarea` eines HTML-Formulars an, das in einem schwebenden Bedienfeld namens `scriptlayer` definiert ist. Wenn Sie an dem, im schwebenden Bedienfeld ausgewählten Code Änderungen vornehmen, ruft die Erweiterung die Funktion `updateScript()` auf, um Ihre Änderungen zu speichern. Wenn Sie **Script Editor** aufgerufen haben, ohne ein Skriptsymbol auszuwählen, zeigt die Erweiterung den Text `(no script selected)` in einem schwebenden Bedienfeld namens `blanklayer` an.

Führen Sie zum Erstellen der Erweiterung die folgenden Schritte aus:

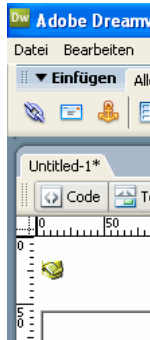
- „Schwebende Bedienfelder erstellen“ auf Seite 211
- „JavaScript-Code schreiben“ auf Seite 212
- „Menüelemente erstellen“ auf Seite 214

JavaScript-Code schreiben

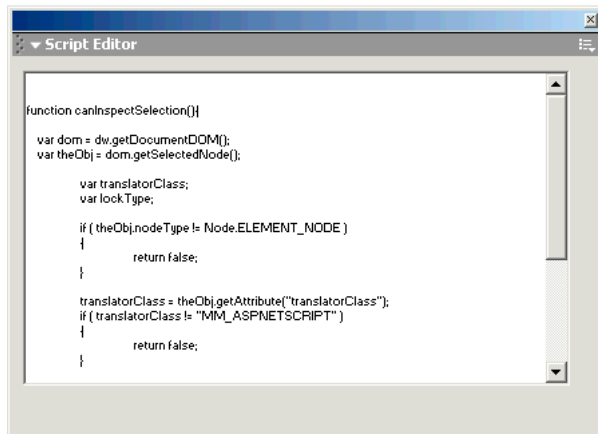
Der JavaScript-Code für den **Script Editor** besteht aus einer Funktion für schwebende Bedienfelder `selectionchanged()`, die Dreamweaver aufruft, und einer benutzerdefinierten Funktion `updateScript()`.

`selectionChanged()`: ist ein Skriptsymbol ausgewählt?

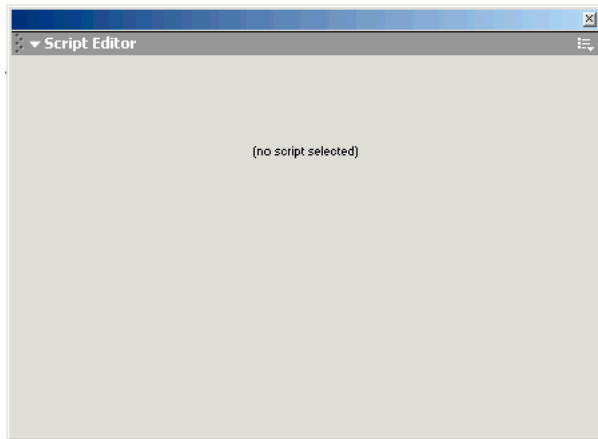
Die Funktion `selectionChanged()` bestimmt, ob ein Skriptsymbol in der Entwurfsansicht ausgewählt ist. Ein Skriptsymbol wird in der Entwurfsansicht angezeigt, wenn eine JavaScript-Routine im body-Teil eines Dokuments vorhanden ist und wenn im Abschnitt **Unsichtbare Elemente** des Dialogfelds **Voreinstellungen** die Option **Skripts** ausgewählt ist. In der folgenden Abbildung ist ein Skriptsymbol dargestellt:



Die Funktion `selectionChanged()` ruft zuerst die Funktion `dw.getDocumentDOM()` auf, um das Dokumentobjektmodell (DOM) für das Dokument des Benutzers abzurufen. Anschließend ruft sie die Funktion `getSelectedNode()` auf, um zu prüfen, ob es sich bei dem ausgewählten Knoten für das Dokument erstens um ein Element und zweitens um ein `SCRIPT`-Tag handelt. Wenn für beide Bedingungen der Wert „true“ gilt, bewirkt die Funktion `selectionChanged()`, dass das schwebende Bedienfeld `scripteditor` sichtbar ist, und lädt dieses schwebende Bedienfeld mit dem zugrunde liegenden JavaScript-Code. Darüber hinaus setzt sie die Eigenschaft `visibility` des schwebenden Bedienfelds `blanklayer` auf den Wert `hidden`. In der folgenden Abbildung ist das schwebende Bedienfeld `scriptlayer` mit dem ausgewählten JavaScript-Code dargestellt:



Wenn es sich bei dem ausgewählten Knoten weder um ein Element noch um ein `script`-Tag handelt, bewirkt die Funktion `selectionChanged()`, dass das schwebende Bedienfeld `blanklayer` sichtbar ist und das Bedienfeld `scriptlayer` ausgeblendet wird. Das schwebende Bedienfeld `blanklayer` zeigt den Text `(no script selected)` wie in der folgenden Abbildung dargestellt an:



So fügen Sie die `selectionChanged()`-Funktion hinzu:

- 1 Öffnen Sie die Datei `scriptEditor.htm` im Ordner `Configuration\Floater`.
- 2 Geben Sie in den Datei-Header den folgenden Code ein:

```
function selectionChanged(){
    /* den ausgewählten Knoten abrufen */
    var theDOM = dw.getDocumentDOM();
    var theNode = theDOM.getSelectedNode();

    /* überprüfen, ob sich bei dem Knoten um ein Skriptsymbol handelt */
    if (theNode.nodeType == Node.ELEMENT_NODE && ~
        theNode.tagName == "SCRIPT"){
        document.layers['scriptlayer'].visibility = 'visible';
        document.layers['scriptlayer'].document.theForm.~
            scriptCode.value = theNode.innerHTML;
        document.layers['blanklayer'].visibility = 'hidden';
    }else{
        document.layers['scriptlayer'].visibility = 'hidden';
        document.layers['blanklayer'].visibility = 'visible';
    }
}
```

- 3 Speichern Sie die Datei.

updateScript(): Änderungen wieder in das Dokument schreiben

Die Funktion `updateScript()` schreibt das ausgewählte Skript wieder in das Dokument zurück, wenn ein `onBlur`-Ereignis in `textarea` des Bedienfelds `scriptlayer` eintritt. Das Formularelement `textarea` beinhaltet den JavaScript-Code, und das `onBlur`-Ereignis tritt ein, wenn `textarea` den Eingabe-Fokus verliert.

So fügen Sie die updateScript()-Funktion hinzu:

1 Öffnen Sie die Datei **scriptEditor.htm** im Ordner **Configuration\Floaters**.

2 Geben Sie in den Datei-Header den folgenden Code ein:

```
/* das Dokument durch Änderungen aktualisieren, die
   der Benutzer im Textbereich vorgenommen hat */

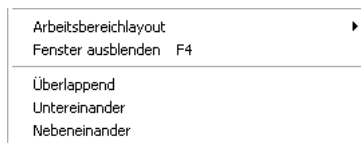
function updateScript(){
    var theDOM = dw.getDocumentDOM();
    var theNode = theDOM.getSelectedNode();
    theNode.innerHTML = document.layers['scriptlayer'].document.
        theForm.scriptCode.value;
}

</script>
</head>
```

3 Speichern Sie die Datei.

Menüelemente erstellen

Es reicht nicht aus, den Script Editor-Code im Ordner **Configuration\Floaters** zu speichern. Sie müssen auch die Funktion `dw.setFloaterVisibility('scriptEditor',true)` oder die Funktion `dw.toggleFloater('scriptEditor')` aufrufen, um das schwebende Bedienfeld zu laden und sichtbar zu machen. Der Script Editor lässt sich am besten aus dem Menü **Fenster** abrufen, das in der Datei **menus.xml** definiert ist. Sie müssen das `menuitem`-Tag erstellen, das einen Eintrag für die Erweiterung **Script Editor** im Menü **Fenster** erstellt, wie in der folgenden Abbildung dargestellt:



Wenn Sie für das aktuelle Dokument zuerst ein Skriptsymbol in der Entwurfsansicht und dann das Menüelement **Script Editor** auswählen, ruft Dreamweaver das schwebende Bedienfeld **Script Editor** auf und zeigt den JavaScript-Code an, der dem Skriptsymbol zugrunde liegt. Wenn Sie das Menüelement auswählen, ohne vorher ein Skriptsymbol ausgewählt zu haben, zeigt Dreamweaver das Bedienfeld `blanklayer` an, das den Text `(no script selected)` anzeigt.

So fügen Sie das Menüelement hinzu:

1 Öffnen Sie die Datei **menus.xml** im Ordner **Configuration\Menus**.

2 Suchen Sie das Tag, das mit `<menuitem name="Tile _Vertically"` beginnt, und setzen Sie die Einfügemarke hinter das schließende `>` des Tags.

3 Fügen Sie Folgendes in eine neue Zeile ein:

```
<menuitem name="Script Editor" enabled="true" ~
command="dw.toggleFloater('scriptEditor')"-
checked="dw.setFloaterVisibility('scriptEditor') " />
```

4 Speichern Sie die Datei.

API-Funktionen für schwebende Bedienfelder

Alle benutzerspezifischen Funktionen der API für schwebende Bedienfelder sind optional.

Einige der in diesem Abschnitt beschriebenen Funktionen können nur unter Windows ausgeführt werden. Ob dies der Fall ist, können Sie jeweils der Beschreibung einer Funktion entnehmen.

displayHelp()

Beschreibung

Wenn diese Funktion definiert ist, wird die Schaltfläche **Hilfe** unter den Schaltflächen **OK** und **Abbrechen** in Ihrem Dialogfeld angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche **Hilfe** klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keinen Wert.

Beispiel

```
// die folgende Instanz von displayHelp() öffnet  
// in einem Browser eine Datei, die die Funktionsweise  
// der Erweiterung erläutert.  
function displayHelp() {  
    var myHelpFile = dw.getConfigurationPath() +  
        '/ExtensionsHelp/superDuperHelp.htm';  
    dw.browseDocument(myHelpFile);  
}
```

documentEdited()

Beschreibung

Diese Funktion wird aufgerufen, sobald das schwebende Bedienfeld sichtbar wird und die aktuellen Bearbeitungsvorgänge abgeschlossen sind (d. h., vor dem Aufrufen dieser Funktion werden möglicherweise mehrere Bearbeitungsvorgänge ausgeführt). Diese Funktion sollte nur dann definiert werden, wenn das schwebende Bedienfeld Änderungen am Dokument verfolgen muss.

Hinweis: Da `documentEdited()` die Systemleistung beeinträchtigt, sollte diese Funktion nur dann definiert werden, wenn sie erforderlich ist.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Im folgenden Beispiel für die Funktion `documentEdited()` wird das Dokument nach AP-Elementen durchsucht. Die Anzahl der AP-Elemente im Dokument wird in ein Textfeld eingetragen:

```
function documentEdited(){
    /* eine Liste mit sämtlichen AP-Elementen im Dokument wird erstellt */
    var theDOM = dw.getDocumentDOM();
    var layersInDoc = theDOM.getElementsByTagName("layer");
    var layerCount = layersInDoc.length;

    /* das numOfLayers-Feld wird mit der neuen Anzahl an AP-Elementen aktualisiert */
    document.theForm.numOfLayers.value = layerCount;
}
```

getDockingSide()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Legt die Positionen fest, an denen ein schwebendes Bedienfeld andocken kann. Die Funktion gibt einen String zurück, der eine Kombination der Wörter "left", "right", "top" und "bottom" enthält. Ist diese Bezeichnung im String enthalten, können Sie ein schwebendes Bedienfeld auf dieser Seite andocken. Fehlt die Funktion, ist das Andocken auf keiner Seite möglich.

Mithilfe dieser Funktion können Sie verhindern, dass Bedienfelder auf einer bestimmten Seite des Dreamweaver-Arbeitsbereichs oder untereinander andocken.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String mit den Wörtern "left", "right", "top" und "bottom", oder eine Kombination davon, mit dem festgelegt wird, an welcher Position Dreamweaver das schwebende Bedienfeld andocken kann.

Beispiel

```
getDockingSide()
{
    return dock_side = "left top";
}
```

initialPosition()

Beschreibung

Bestimmt die Anfangsposition des schwebenden Bedienfelds, wenn es zum ersten Mal aufgerufen wird. Falls diese Funktion nicht definiert ist, wird als Standardposition die Mitte des Bildschirms angenommen.

Argumente

platform

- Das Argument *platform* weist je nach Benutzerplattform entweder den Wert "Mac" oder "Win" auf.

Rückgabewerte

Dreamweaver erwartet einen String der Form "leftPosInPixels,topPosInPixels".

Beispiel

Im folgenden Beispiel für die Funktion `initialPosition()` wird festgelegt, dass das schwebende Bedienfeld beim ersten Aufruf in Windows 420 Pixel vom linken und 20 Pixel vom oberen Bildschirmrand entfernt angezeigt werden soll. Für den Macintosh soll das Bedienfeld 390 Pixel vom linken und 20 Pixel vom oberen Bildschirmrand entfernt sein:

```
function initialPosition(platform) {  
    var initPos = "420,20";  
    if (platform == "macintosh") {  
        initPos = "390,20";  
    }  
    return initPos;  
}
```

initialTabs()

Beschreibung

Legt fest, welche anderen schwebenden Bedienfelder zusammengefasst werden, wenn dieses schwebende Bedienfeld erstmalig angezeigt wird. Wenn eines der aufgeführten Bedienfelder zuvor angezeigt wurde, wird es nicht in die Registergruppe aufgenommen. Um sicherzustellen, dass zwei benutzerdefinierte schwebende Bedienfelder zusammen gruppiert werden, sollten sie mit der Funktion `initialTabs()` aufeinander Bezug nehmen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String der Form "floaterName1,floaterName2,...floaterNameN".

Beispiel

Im folgenden Beispiel für die Funktion `initialTabs()` wird festgelegt, dass das schwebende Bedienfeld beim ersten Aufruf zusammen mit dem schwebenden Bedienfeld **scriptEditor** gruppiert wird:

```
function initialTabs() {  
    return "scriptEditor";  
}
```

isATarget()

Verfügbarkeit

Dreamweaver MX (nur Windows)

Beschreibung

Legt fest, ob andere schwebende Bedienfelder an diesem Bedienfeld andocken können. Wenn die Funktion `isATarget()` nicht definiert ist, verhindert Dreamweaver, dass andere Bedienfelder sich an dieses Bedienfeld andocken. Dreamweaver ruft diese Funktion auf, wenn der Benutzer versucht, dieses Bedienfeld mit anderen Bedienfeldern zu kombinieren.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn andere schwebende Bedienfelder an dieses Bedienfeld andocken können, und `false`, falls dies nicht möglich ist.

Beispiel

```
IsATarget()  
{  
    return true;  
}
```

isAvailableInCodeView()

Beschreibung

Legt fest, ob schwebende Bedienfelder in der Codeansicht aktiviert sein sollen. Wenn diese Funktion nicht definiert wird, sind schwebende Bedienfelder in der Codeansicht deaktiviert.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das schwebende Bedienfeld in der Codeansicht aktiviert sein soll, und `false`, wenn dies nicht gewünscht ist.

isResizable()

Verfügbarkeit

Dreamweaver 4.

Beschreibung

Legt fest, ob der Benutzer die Größe eines schwebenden Bedienfelds ändern kann. Wurde die Funktion nicht definiert oder gibt sie den Wert `true` zurück, kann der Benutzer die Größe des Bedienfelds anpassen. Gibt die Funktion den Wert `false` zurück, kann der Benutzer die Größe des Bedienfelds nicht anpassen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: `true`, wenn der Benutzer die Größe des Bedienfelds anpassen kann, `false`, falls dies nicht möglich ist.

Beispiel

Nachfolgend ein Beispiel für einen Fall, in dem der Benutzer die Größe des Bedienfelds nicht anpassen kann:

```
function isResizable()  
{  
    return false;  
}
```

selectionChanged()

Beschreibung

Wird aufgerufen, wenn das schwebende Bedienfeld sichtbar wird und wenn die Auswahl geändert wird (wenn der Fokus zu einem neuen Dokument wechselt oder die Einfügemarke an eine andere Stelle im aktuellen Dokument verschoben wird). Diese Funktion sollte nur dann definiert werden, wenn das schwebende Bedienfeld die Auswahl verfolgen muss.

Hinweis: Da `selectionChanged()` die Systemleistung beeinträchtigt, sollte diese Funktion nur dann definiert werden, wenn sie unbedingt erforderlich ist.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Im folgenden Beispiel für `selectionChanged()` wird im schwebenden Bedienfeld abhängig davon, ob es sich bei der Auswahl um ein Skriptsymbol oder um ein anderes AP-Element handelt, das entsprechende AP-Element angezeigt: Wenn die Auswahl ein Skriptsymbol ist, wird das AP-Element `scriptlayer` in Dreamweaver angezeigt. Andernfalls zeigt Dreamweaver das AP-Element `blanklayer` an.

```
function selectionChanged(){
    /* den ausgewählten Knoten abrufen */
    var theDOM = dw.getDocumentDOM();
    var theNode = theDOM.getSelectedNode();
    /* überprüfen, ob sich bei dem Knoten um ein Skriptsymbol handelt */
    if (theNode.nodeType == Node.ELEMENT_NODE && ~
        theNode.tagName == "SCRIPT"){
        document.layers['blanklayer'].visibility = 'hidden';
        document.layers['scriptlayer'].visibility = 'visible';
    }
    else{
        document.layers['scriptlayer'].visibility = 'hidden';
        document.layers['blanklayer'].visibility = 'visible';
    }
}
```

Einfluss auf die Programmleistung

Wenn die Funktionen `selectionChanged()` bzw. `documentEdited()` in benutzerdefinierten schwebenden Bedienfeldern deklariert sind, hat dies eventuell negative Auswirkungen auf die Leistung von Dreamweaver. Beachten Sie, dass die Funktionen `documentEdited()` und `selectionChanged()` nach jedem Tastaturanschlag und nach jedem Mausklick aufgerufen werden, wenn in Dreamweaver länger als eine Zehntelsekunde keine Bearbeitung ausgeführt wird. Beim Testen eines schwebenden Bedienfelds sollten Sie auf jeden Fall verschiedene Szenarien verwenden und auch mit großen HTML-Dokumenten von mindestens 100 KB arbeiten, um die Auswirkung auf die Systemleistung zu ermitteln.

Um Leistungseinbußen zu vermeiden, wurde in Dreamweaver 3 die Funktion `setTimeout()` als globale Methode implementiert. Wie bei Browsern nimmt die Funktion `setTimeout()` zwei Argumente an: den aufzurufenden JavaScript-Code und die Wartezeit vor dem Aufruf (in Millisekunden).

Mithilfe der Methode `setTimeout()` können Sie Pausen in die Verarbeitung integrieren. Aufgrund dieser Pausen kann der Benutzer dann weiter interaktiv in der Anwendung arbeiten. Sie müssen diese Pausen integrieren, da während der Skriptbearbeitung die Bildschirmanzeige nicht aktualisiert wird, sodass der Benutzer keine Bearbeitungen durchführen kann. Aufgrund dieser Pausen können Sie auch die Benutzeroberfläche oder das schwebende Bedienfeld nicht aktualisieren.

Der folgende Code wurde für ein schwebendes Bedienfeld geschrieben, in dem Informationen über die einzelnen AP-Elemente des Dokuments angezeigt werden. Er verwendet die `setTimeout()`-Methode, um nach jedem AP-Element eine Pause von einer halben Sekunde einzufügen.

```
/* ein Flag erstellen, das angibt, ob eine Änderung verarbeitet wird, und dieses Flag auf "false"
setzen.*/
document.running = false;

/* diese Funktion wird aufgerufen, wenn das Dokument bearbeitet wird */
function documentEdited(){
    /* eine Liste aller zu verarbeitenden AP-Elemente erstellen */
    var dom = dw.getDocumentDOM();
    document.layers = dom.getElementsByTagName("layer");
    document.numLayers = document.layers.length;
    document.numProcessed = 0;

    /* einen Timer setzen, der processLayer() aufruft; wenn die
    * Verarbeitung der vorigen Änderung noch nicht abgeschlossen ist, ist der Timer
    * bereits gesetzt. */
    if (document.running = false){
        setTimeout("processLayer()", 500);
    }

    /* das Verarbeitungsflag auf "true" setzen */
    document.running = true;
}
```

```
/* ein AP-Element verarbeiten*/
function processLayer(){
    /* Informationen über das nächste nicht verarbeitete AP-Element anzeigen.
       displayLayer() ist eine Funktion, die Sie schreiben würden, um
       das "Zauberwerk" zu vollbringen.0*/
    displayLayer(document.layers[document.numProcessed]);

    /* Wenn es nichts mehr zu tun gibt, wird ein Timeout gesetzt, damit
       * die nächste Ebene verarbeitet wird. Wenn fertig, wird das Flag document.running
       * auf "false" gesetzt. */
    document.numProcessed = document.numProcessed + 1;
    if (document.numProcessed < document.numLayers){
        setTimeout("processLayer()", 500);
    }else{
        document.running = false;
    }
}
```

Kapitel 16: Verhalten

Mithilfe von Verhalten können auch Nicht-Programmierer ihre HTML-Seiten interaktiv gestalten. Verhalten geben Webdesignern die Möglichkeit, Seitenelementen bestimmte Aktionen zuzuweisen. Dazu muss einfach nur ein HTML-Formular ausgefüllt werden.

Der Begriff *Verhalten* bezieht sich auf Kombinationen von Ereignissen (beispielsweise `onClick`, `onLoad` oder `onSubmit`) und Aktionen (Check Plugin, Go to URL, Swap Image). Der Browser legt fest, welche Ereignisse von welchen HTML-Elementen akzeptiert werden. Die Dateien, in denen die Ereignisse aufgeführt sind, die von den einzelnen Browsern unterstützt werden, befinden sich im Unterordner **Configuration\Behaviors\Events** im Anwendungsordner von Dreamweaver CS3.

Aktionen sind die Teile eines Verhaltens, über die Sie Kontrolle haben. Beim Schreiben eines Verhaltens erstellen Sie daher in Wirklichkeit eine Aktionsdatei. Aktionen sind HTML-Dateien. Der body-Bereich einer Aktionsdatei enthält in der Regel ein HTML-Formular für die Annahme von Aktionsparametern (zum Beispiel von Parametern, mit denen festgelegt wird, welche absolut positionierten Elemente angezeigt oder ausgeblendet werden sollen). Der head-Bereich einer Aktionsdatei enthält JavaScript-Funktionen, die Formulareingaben aus dem body-Inhalt verarbeiten und die Funktionen, Argumente und Ereignisprozeduren steuern, die in das Dokument eines Benutzers eingefügt werden.

Schreiben Sie Verhaltensaktionen, wenn Sie Funktionen auch Nicht-Programmierern zur Verfügung stellen möchten oder wenn Sie dieselbe JavaScript-Funktion mehrmals einfügen, aber jedes Mal die Parameter ändern möchten.

Hinweis: Es ist nicht möglich, VBScript-Funktionen mit Verhalten direkt einzufügen. Sie können eine VBScript-Funktion jedoch indirekt einfügen. Bearbeiten Sie dazu das Dokumentobjektmodell (DOM) in der Funktion `applyBehavior()`.

In der folgenden Tabelle sind die Dateien zum Erstellen von Verhaltensaktionen aufgeführt:

Pfad	Datei	Beschreibung
Configuration\Behaviors\Actions\	behavior action.htm	Der body-Teil der Datei enthält ein HTML-Formular für die Parameter der Aktion. Im head-Bereich der Datei sind die JavaScript-Funktionen enthalten.

Hinweis: Weitere Informationen über Serververhalten mit Funktionen für Webanwendungen finden Sie in „Serververhalten“ auf Seite 233.

Funktionsweise von Verhalten

Wenn ein Benutzer ein HTML-Element in einem Dreamweaver-Dokument auswählt und auf die Schaltfläche mit dem Pluszeichen (+) im Bedienfeld **Verhalten** klickt, läuft die folgende Ereigniskette ab:

1 Dreamweaver ruft die Funktion `canAcceptBehavior()` in den einzelnen Aktionsdateien auf, um zu überprüfen, ob die jeweilige Aktion für das Dokument oder das ausgewählte Element zulässig ist.

Ist der Rückgabewert dieser Funktion `false`, zeigt Dreamweaver die Aktion im Popupmenü **Aktionen** ausgeblendet an. (Die Aktion **Shockwave steuern** ist beispielsweise ausgeblendet, wenn das Dokument des Benutzers über keine SWF-Dateien verfügt.) Wenn eine Liste mit Ereignissen zurückgegeben wird, vergleicht Dreamweaver jedes Ereignis mit den gültigen Ereignissen des aktuell ausgewählten HTML-Elements und des Zielbrowsers, bis eine Übereinstimmung gefunden wurde. Dreamweaver fügt ganz oben im Popupmenü „Ereignisse“ das übereinstimmende Ereignis aus der Funktion `canAcceptBehavior()` ein. Gibt es keine Übereinstimmung, wird das Standardereignis für das HTML-Element (das in der Ereignisdatei mit einem Sternchen [*] gekennzeichnet ist) zum obersten Element. Die restlichen Ereignisse im Menü werden der Ereignisdatei entnommen.

2 Der Benutzer wählt eine Aktion im Popupmenü der Aktionen aus.

3 Dreamweaver ruft die Funktion `windowDimensions()` auf, um die Größe des Dialogfelds **Parameter** zu bestimmen. Ist die Funktion `windowDimensions()` nicht definiert, wird die Größe automatisch festgelegt.

Unabhängig vom Inhalt des `body`-Elements wird stets ein Dialogfeld angezeigt, das auf der rechten Seite die beiden Schaltflächen **OK** und **Abbrechen** enthält.

4 Dreamweaver zeigt ein Dialogfeld mit den `BODY`-Elementen der Aktionsdatei an. Wenn das `body`-Tag der Aktionsdatei eine `onLoad`-Ereignisprozedur enthält, führt Dreamweaver diese aus.

5 Der Benutzer gibt die Parameter für die Aktion ein. Dreamweaver führt die an die einzelnen Formularfelder angehängten Ereignisprozeduren aus, sobald sie vom Benutzer aktiviert werden.

6 Der Benutzer klickt auf **OK**.

7 Dreamweaver ruft die Funktionen `behaviorFunction()` und `applyBehavior()` in der ausgewählten Aktionsdatei auf. Diese Funktionen geben Strings zurück, die in das Benutzerdokument eingefügt werden.

8 Wenn der Benutzer später auf die Aktion in der Spalte **Aktionen** doppelklickt, öffnet Dreamweaver das Dialogfeld **Parameter** erneut und führt die Prozedur `onLoad` aus. Danach ruft Dreamweaver die Funktion `inspectBehavior()` in der ausgewählten Aktionsdatei auf, die die Felder mit den vom Benutzer zuvor eingegebenen Daten füllt.

Mehrere Funktionen in die Benutzerdatei einfügen

Aktionen können mehrere Funktionen, d. h. die Hauptverhaltensfunktion sowie eine beliebige Anzahl von Helferfunktionen, in den `head`-Bereich einfügen. Mehrere Verhalten können sogar Helferfunktionen gemeinsam nutzen, sofern die Funktionsdefinition in allen Aktionsdateien genau identisch ist. Eine Möglichkeit zum Prüfen der Übereinstimmung gemeinsam genutzter Funktionen besteht darin, die einzelnen Helferfunktionen in einer externen JavaScript-Datei zu speichern und sie mithilfe von `<SCRIPT SRC="externalFile.js">` in die betreffenden Aktionsdateien einzufügen.

Wenn der Benutzer ein Verhalten löscht, versucht Dreamweaver, alle nicht benutzten Helferfunktionen zu entfernen, die zu diesem Verhalten gehören. Falls auch andere Verhalten eine Helferfunktion verwenden, wird diese nicht gelöscht. Der Algorithmus zum Löschen von Helferfunktionen ist sehr konservativ ausgelegt. Daher lässt Dreamweaver gelegentlich eine nicht mehr benötigte Helferfunktion im Benutzerdokument unangetastet.

Was ist zu tun, wenn eine Aktion einen Rückgabewert erfordert?

In einigen Fällen muss eine Ereignisprozedur einen Rückgabewert haben (zum Beispiel `onMouseOver="window.status='This is a link'; return true"`). Wenn Dreamweaver jedoch die Aktion `"return behaviorName(args)"` in die Ereignisprozedur einfügt, werden spätere Verhalten in der Liste übersprungen.

Um diese Einschränkung zu umgehen, setzen Sie in dem String, der von der Funktion `behaviorFunction()` zurückgegeben wird, die Variable `document.MM_returnValue` auf den gewünschten Rückgabewert. Dadurch wird Dreamweaver veranlasst, `return document.MM_returnValue` am Ende der Liste der Aktionen in die Ereignisprozedur einzufügen. In der Datei **Validate Form.js** finden Sie ein Beispiel, in dem die Variable `MM_returnValue` vorkommt. Die Datei befindet sich im Unterordner **Configuration\Behaviors\Actions** des Dreamweaver-Anwendungsordners.

Einfaches Beispiel für ein Verhalten

Am folgenden Beispiel werden die Funktionsweise und das Erstellen von Verhalten demonstriert. Im Unterordner **Configuration/Behaviors/Actions** des Dreamweaver-Anwendungsordners befinden sich Beispiele, die aber teilweise sehr komplex sind. Dieses Beispiel ist einfacher und besser geeignet, das Erstellen von Verhalten zu erlernen. Die einfachste Aktionsdatei, mit der Sie beginnen können, ist **Call JavaScript.htm** (zusammen mit ihrem Gegenstück **Call JavaScript.js**, das alle JavaScript-Funktionen enthält).

Zum Erstellen des Verhaltens gehen Sie wie folgt vor:

- „Verhaltenserweiterung erstellen“ auf Seite 223
- „Aufzurufende HTML-Dateien erstellen“ auf Seite 225
- „Verhalten testen“ auf Seite 226

Verhaltenserweiterung erstellen

Der folgende Code ist ein relativ einfaches Beispiel. Er prüft die Browsermarke und wechselt dann je nach verwendetem Browser (Netscape oder Microsoft Internet Explorer) zur entsprechenden Seite. Dieser Code kann problemlos erweitert werden, damit auch andere Browserversionen wie Opera und WebTV erkannt werden, und kann auch so bearbeitet werden, dass eine andere Aktion als das Wechseln zu URLs ausgeführt wird.

So erstellen Sie eine Verhaltenserweiterung:

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie Folgendes in die Datei ein:

```
<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//dialog">
<html>
<head>
<title>behavior "Check Browser Brand"</title>
<meta http-equiv="Content-Type" content="text/html; ->
<script language="JavaScript">

// Die Funktion, die im HEAD-Bereich des
// Benutzerdokuments eingefügt wird.
function checkBrowserBrand(netscapeURL,explorerURL) {
    if (navigator.appName == "Netscape") {
        if (netscapeURL) location.href = netscapeURL;
    }else if (navigator.appName == "Microsoft Internet Explorer") {
        if (explorerURL) location.href = explorerURL;
    }
}

//***** API *****

function canAcceptBehavior(){
    return true;
}

// Gibt den Namen der Funktion zurück, die in den
// HEAD-Bereich des Benutzerdokuments eingefügt wird.
function behaviorFunction(){
    return "checkBrowserBrand";
}

// Erstellen des Funktionsaufrufs, der mit der
// Ereignisprozedur eingefügt wird.
function applyBehavior() {
    var nsURL = escape(document.theForm.nsURL.value);
    var ieURL = escape(document.theForm.ieURL.value);
    if (nsURL && ieURL) {
        return "checkBrowserBrand(\"'\" + nsURL + "\"'\",\"'\" + ieURL + "\"'\");"
    }else{
        return "Please enter URLs in both fields."
    }
}

// Extrahieren der Argumente aus dem Funktionsaufruf
// in der Ereignisprozedur und erneutes Ausfüllen
// des Parameterformulars.
function inspectBehavior(fnCall){
    var argArray = getTokens(fnCall, "()",",");
    var nsURL = unescape(argArray[1]);
    var ieURL = unescape(argArray[2]);
    document.theForm.nsURL.value = nsURL;
    document.theForm.ieURL.value = ieURL;
}

//***** LOCALE FUNKTIONEN *****
```

```

// Setzt den Zeiger in das erste Textfeld
// und wählt den Inhalt aus, wenn vorhanden
function initializeUI(){
    document.theForm.nsURL.focus();
    document.theForm.nsURL.select();
}

// den Benutzer die Navigator- und
// IE-URLs anzeigen lassen
function browseForURLs(whichButton){
    var theURL = dreamweaver.browseForFileURL();
    if (whichButton == "nsURL"){
        document.theForm.nsURL.value = theURL;
    }else{
        document.theForm.ieURL.value = theURL;
    }
}

//***** ENDE des JAVASCRIPTS *****
</script>
</head>
<body>
<form method="post" action="" name="theForm">
<table border="0" cellpadding="8">
<tr>
<td nowrap="nowrap">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&Go to this URL if the browser is ~
    Netscape Navigator:<br>
<input type="text" name="nsURL" size="50" value=""> &nbsp;&nbsp;&
<input type="button" name="nsBrowse" value="Browse..." ~
    onClick="browseForURLs('nsURL') "></td>
</tr>
<tr>
<td nowrap="nowrap">&nbsp;&nbsp;&~Go to this URL is the browser is ~
    Microsoft Internet Explorer:<br>
<input type="text" name="ieURL" size="50" value=""> &nbsp;&nbsp;&
<input type="button" name="ieBrowse" value="Browse..." ~
    onClick="browseForURLs('ieURL') "></td>
</tr>
</table>
</form>
</body>
</html>

```

3 Speichern Sie die Datei unter dem Namen **BrowserDependendURL.htm** im Ordner **Configuration\Behaviors\Actions\BrowserDependentURL.htm**.

Aufzurufende HTML-Dateien erstellen

Als Nächstes erstellen Sie die aufzurufenden HTML-Dateien für die Verwendung von Internet Explorer und die Verwendung von Netscape Navigator.

So erstellen Sie die aufzurufenden HTML-Dateien:

1 Erstellen Sie eine neue Datei mit folgendem Inhalt:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Internet Explorer Only</title>
</head>

<body>
This is the page to go to if you are using Internet Explorer.
</body>
</html>
```

2 Speichern Sie die Datei unter dem Namen **iecontent.htm** in einer Site auf Ihrem Computer.

3 Erstellen Sie eine weitere neue Datei mit folgendem Inhalt:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Netscape Navigator content</title>
</head>

<body>
This is the page to go to if you are using Netscape Navigator.
</body>
</html>
```

4 Speichern Sie die Datei unter dem Namen **netscapecontent.htm** in dem Ordner, in dem Sie auch **iecontent.htm** gespeichert haben.

5 Starten Sie Dreamweaver neu.

6 Erstellen Sie eine neue HTML-Datei mit folgendem Inhalt:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Which browser</title>
</head>

<body>
</body>
</html>
```

7 Speichern Sie die Datei unter dem Namen **whichbrowser.htm** in dem Ordner, in dem Sie auch **iecontent.htm** gespeichert haben.

8 Klicken Sie auf die Plus-Schaltfläche (+) im Bedienfeld **Verhalten**, und wählen Sie das Verhalten **Check Browser Brand** (Browser-Marke prüfen) aus.

9 Klicken Sie auf die Schaltfläche **Durchsuchen** neben dem Textfeld **Go to the URL if the browser is Netscape Navigator** (Bei Verwendung von Netscape Navigator diese URL aufrufen), und wählen Sie die Datei **netscapecontent.htm** aus.

10 Klicken Sie auf die Schaltfläche **Durchsuchen** neben dem Textfeld **Go to the URL if the browser is Internet Explorer** (Bei Verwendung von Internet Explorer diese URL aufrufen), und wählen Sie die Datei **iecontent.htm** aus.

11 Klicken Sie auf **OK**.

Dreamweaver fügt der Datei **whichbrowser.htm** den angegebenen JavaScript-Code hinzu. Die Datei sieht dann wie folgt aus:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Which browser</title>
<script language="JavaScript" type="text/JavaScript">
<!--
function checkBrowserBrand(netscapeURL,explorerURL) {
    if (navigator.appName == "Netscape") {
        if (netscapeURL) location.href = netscapeURL;
    }else if (navigator.appName == "Microsoft Internet Explorer") {
        if (explorerURL) location.href = explorerURL;
    }
}
//-->
</script>
</head>

<body onLoad="checkBrowserBrand('netscapecontent.htm','iecontent.htm')">
</body>
</html>
```

Verhalten testen

Zum Schluss können Sie das Verhalten testen.

So testen Sie das Verhalten:

❖ Öffnen Sie die Datei **whichbrowser.htm** in Ihrem Browser.

Je nach dem verwendeten Browser wird entweder **iecontent.htm** oder **netscapecontent.htm** angezeigt.

Die Funktionen der Verhaltens-API

Zwei API-Funktionen für Verhalten sind obligatorisch (`applyBehavior()` und `behaviorFunction()`); die übrigen sind optional.

applyBehavior()**Beschreibung**

Diese Funktion fügt in das Benutzerdokument eine Ereignisprozedur ein, die die von `behaviorFunction()` eingefügte Funktion aufruft. Die Funktion `applyBehavior()` kann am Benutzerdokument auch weitere Bearbeitungsschritte ausführen, darf aber nicht das Objekt löschen, auf das das Verhalten angewendet wird, bzw. das Objekt, das die Aktion empfängt.

Wenn Sie die Funktion `applyBehavior()` schreiben, müssen Sie sich entscheiden, wie das Dokument des Benutzers geändert werden soll. Sie können sich etwa dafür entscheiden, Code in die script-Tags im Hauptteil des Dokuments einzufügen. Sie können dies mithilfe der Standard-APIs für DOM-Bearbeitungen erledigen.

Argumente

uniqueName

Das Argument ist eine ID, die unter den Instanzen aller Verhalten im Benutzerdokument eindeutig ist. Sein Format lautet *functionNameInteger*, wobei *functionName* der Name der Funktion ist, die von `behaviorFunction()` eingefügt wird. Dieses Argument kann nützlich sein, wenn Sie ein Tag in ein Benutzerdokument einfügen und seinem Attribut `NAME` einen eindeutigen Wert zuweisen möchten.

Rückgabewerte

Dreamweaver erwartet einen String mit dem Funktionsaufruf, der in das Benutzerdokument eingefügt werden soll, in der Regel, nachdem der Benutzer Parameter eingegeben hat. Wenn die Funktion `applyBehavior()` feststellt, dass der Benutzer einen ungültigen Eintrag vorgenommen hat, kann die Funktion statt des Funktionsaufrufs einen Fehlerstring zurückgeben. Falls der String leer ist (`return ""`), meldet Dreamweaver keinen Fehler. Ist der String jedoch weder leer noch ein Funktionsaufruf, zeigt Dreamweaver ein Dialogfeld mit dem Text *Invalid input supplied for this behavior:* und dem von `applyBehavior()` zurückgegebenen String an. Wenn der Rückgabewert `null` ist (`return;`), zeigt Dreamweaver an, dass ein Fehler aufgetreten ist, gibt jedoch keine spezifischen Informationen.

Hinweis: Vor den Anführungszeichen (") innerhalb des zurückgegebenen Strings muss ein umgekehrter Schrägstrich (\) eingegeben werden, damit der JavaScript-Interpreter keine Fehler meldet.

Beispiel

Das folgende Beispiel für die Funktion `applyBehavior()` gibt einen Aufruf an die Funktion `MM_openBrWindow()` zurück und übergibt ihr benutzerdefinierte Parameter (Höhe und Breite des Fensters, Vorhandensein von Bildlaufleisten, Symbolleisten und andere Merkmalen sowie die URL, die im Fenster geöffnet werden soll):

```
function applyBehavior() {
    var i,theURL,theName,arrayIndex = 0;
    var argArray = new Array(); //use array to produce correct ~
        number of commas w/o spaces
    var checkBoxNames = new Array("toolbar","location",-
        "status","menubar","scrollbars","resizable");

    for (i=0; i<checkBoxNames.length; i++) {
        theCheckBox = eval("document.theForm." + checkBoxNames[i]);
        if (theCheckBox.checked) argArray[arrayIndex++] = (checkBoxNames[i] + "=yes");
    }
    if (document.theForm.width.value)
        argArray[arrayIndex++] = ("width=" + document.theForm.width.value);
    if (document.theForm.height.value)
        argArray[arrayIndex++] = ("height=" + document.theForm.height.value);
    theURL = escape(document.theForm.URL.value);
    theName = document.theForm.winName.value;
    return "MM_openBrWindow('" + theURL + "','" + theName + "','" + argArray.join(',') + "')";
}
```

behaviorFunction()

Beschreibung

Mit dieser Funktion werden eine oder mehrere Funktionen – die durch die folgenden Tags umgeben sind, sofern sie nicht bereits existieren – in den head-Bereich des Dokuments des Benutzers eingefügt:

```
<SCRIPT LANGUAGE="JavaScript"></SCRIPT>
```

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet entweder einen String, der die JavaScript-Funktionen enthält, oder einen String, der die Namen der Funktionen enthält, die in das Benutzerdokument eingefügt werden sollen. Dieser Wert muss jedes Mal exakt gleich sein, darf also nicht von Benutzereingaben abhängen. Die Funktionen werden nur einmal eingefügt, unabhängig davon, wie oft die Aktion auf Elemente in dem Dokument angewendet wird.

Hinweis: Vor den Anführungszeichen ("") innerhalb des zurückgegebenen Strings muss ein umgekehrter Schrägstrich (\) eingegeben werden, damit der JavaScript-Interpreter keine Fehler meldet.

Beispiel

Die folgende Instanz der Funktion `behaviorFunction()` gibt die Funktion `MM_popupMsg()` zurück:

```
function behaviorFunction() {
    return "+
    \"function MM_popupMsg(theMsg) { //v1.0\\n\"+
    \"alert(theMsg);\\n\"+
    \"}\";
}
```

Der folgende Code entspricht der oben aufgeführten Deklaration `behaviorFunction()`. Bei diesem Code handelt es sich um die Methode, die zum Deklarieren der Funktion `behaviorFunction()` in allen mit Dreamweaver gelieferten Verhalten verwendet wird:

```
function MM_popupMsg(theMsg) { //v1.0
    alert(theMsg);
}

function behaviorFunction() {
    return "MM_popupMsg";
}
```

canAcceptBehavior()

Beschreibung

Die Funktion bestimmt, ob die Aktion für das ausgewählte HTML-Element erlaubt ist, und spezifiziert das Standardereignis, das sie auslösen soll. Kann auch überprüfen, ob bestimmte Objekte (zum Beispiel Shockwave-Filme) im Benutzerdokument vorhanden sind, und die Aktion verbieten, falls diese Objekte nicht vorhanden sind.

Argumente

HTMLElement

Das Argument ist das ausgewählte HTML-Element.

Rückgabewerte

Dreamweaver erwartet einen der folgenden Werte:

- Den Wert `true`, wenn die Aktion erlaubt ist, es aber keine bevorzugten Ereignisse gibt.
- Eine Liste der bevorzugten Ereignisse für diese Aktion (in absteigender Reihenfolge des Vorzugs). Die Angabe von bevorzugten Ereignissen setzt das Standardereignis (gekennzeichnet durch ein Sternchen [*] in der Ereignisdatei) für das ausgewählte Objekt außer Kraft. Siehe Schritt 1 unter „Funktionsweise von Verhalten“ auf Seite 221.
- Den Wert `false`, wenn die Aktion nicht zulässig ist.

Wenn die Funktion `canAcceptBehavior()` den Wert `false` zurückgibt, wird die Aktion im Popupmenü **Aktionen** des Bedienfelds **Verhalten** abgeblendet angezeigt.

Beispiel

Das folgende Beispiel für die Funktion `canAcceptBehavior()` gibt eine Liste bevorzugter Ereignisse für das Verhalten für den Fall zurück, dass das Dokument über benannte Bilder verfügt:

```
function canAcceptBehavior(){
    var theDOM = dreamweaver.getDocumentDOM();
    // Einen Array mit allen Bildern im Dokument abrufen.
    var allImages = theDOM.getElementsByTagName('IMG');
    if (allImages.length > 0){
        return "onMouseOver, onClick, onMouseDown";
    }else{
        return false;
    }
}
```

displayHelp()

Beschreibung

Ist diese Funktion definiert, wird unter den Schaltflächen **OK** und **Abbrechen** im Dialogfeld **Parameter** die Schaltfläche **Hilfe** eingeblendet. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche **Hilfe** klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// die folgende Instanz von displayHelp() öffnet
// in einem Browser eine Datei, die die Funktionsweise
// der Erweiterung erläutert.
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/superDuperHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

deleteBehavior()

Beschreibung

Diese Funktion macht die durch die Funktion `applyBehavior()` durchgeführten Änderungen rückgängig.

Hinweis: Dreamweaver löscht automatisch die mit einem Verhalten verknüpfte Funktionsdeklaration und die Ereignisprozedur, wenn der Benutzer das Verhalten im Bedienfeld **Verhalten** löscht. Daher ist es nur dann nötig, die Funktion `deleteBehavior()` zu definieren, wenn die Funktion `applyBehavior()` zusätzliche Bearbeitungsschritte am Benutzerdokument durchgeführt hat (wenn sie beispielsweise ein Tag eingefügt hat).

Argumente

`applyBehaviorString`

Dieses Argument ist der String, der von der Funktion `applyBehavior()` zurückgegeben wird.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

identifyBehaviorArguments()

Beschreibung

Diese Funktion macht Argumente aus einem Verhaltensfunktionsaufruf als Navigationshyperlinks, abhängige Dateien, URLs, Referenzen im Stil von Netscape 4.0 oder Objektnamen kenntlich. Auf diese Weise können URLs in Verhalten aktualisiert werden, wenn der Benutzer das Dokument an einem anderen Ort speichert, und die verknüpften Dateien können in der Sitemap aufgeführt und beim Hoch- und Herunterladen an einen bzw. von einem Server als abhängige Dateien behandelt werden.

Argumente

theFunctionCall

Dieses Argument ist der String, der von der Funktion `applyBehavior()` zurückgegeben wird.

Rückgabewerte

Dreamweaver erwartet einen String, der eine durch Kommas getrennte Liste der Argumenttypen im Funktionsaufruf enthält. Die Länge der Liste muss der Anzahl der Argumente im Funktionsaufruf entsprechen. Argumenttypen müssen zu einem der folgenden Typen gehören:

- Der Argumenttyp `nav` gibt an, dass es sich bei dem Argument um eine Navigations-URL handelt und es daher in der Sitemap angezeigt werden soll.
- Der Argumenttyp `dep` gibt an, dass es sich bei dem Argument um eine abhängige Datei-URL handelt und es daher genau wie alle anderen abhängigen Dateien berücksichtigt werden soll, wenn ein Dokument mit dem entsprechenden Verhalten von einem Server heruntergeladen oder auf einen Server hochgeladen wird.
- Der Argumenttyp `url` gibt an, dass es sich bei dem Argument um eine Navigations-URL und eine abhängige URL oder um eine URL unbekannten Typs handelt und es daher beim Hochladen auf einen Server oder Herunterladen von einem Server in der Sitemap angezeigt und als abhängige Datei behandelt werden soll.
- Der Argumenttyp `NS4.0ref` gibt an, dass es sich bei dem Argument um eine Objektreferenz im Stil von Netscape 4.0 handelt.
- Der Argumenttyp `IE4.0ref` gibt an, dass es sich bei dem Argument um eine Objektreferenz im Stil von Internet Explorer-DOM 4.0 handelt.
- Der Argumenttyp `objName` gibt an, dass das Argument ein einfacher Objektname ist (wie im Objektattribut `NAME` angegeben). Dieser Typ ist in Dreamweaver 3 neu hinzugekommen.
- Der Argumenttyp `other` gibt an, dass das Argument nicht zu einem der vorstehend aufgeführten Typen gehört.

Beispiel

Das folgende einfache Beispiel für die Funktion `identifyBehaviorArguments()` funktioniert für die Verhaltensaktion **Browserfenster öffnen**, die immer eine Funktion mit drei Argumenten zurückgibt (die zu öffnende URL, den Namen des neuen Fensters und die Liste der Fenstereigenschaften):

```
function identifyBehaviorArguments(fnCallStr) {
    return "URL,other,other";
}
```

Eine komplexere Version der Funktion `identifyBehaviorArguments()` ist für Verhaltensfunktionen erforderlich, deren Argumentanzahl variieren kann (zum Beispiel **Ebene ein-/ausblenden**). Bei dieser Beispielversion der Funktion `identifyBehaviorArguments()` wird davon ausgegangen, dass es eine Mindestanzahl von Argumenten gibt und weitere Argumente immer in Gruppen auftreten, die ein Vielfaches dieser Mindestanzahl enthalten. Eine Funktion, die mindestens vier Argumente haben muss, kann also auch 4, 8 oder 12 Argumente haben, jedoch niemals 10 Argumente:

```
function identifyBehaviorArguments(fnCallStr) {
    var listOfArgTypes;
    var itemArray = dreamweaver.getTokens(fnCallStr, '()', ',');

    // Das Array an Elementen, das von getTokens() zurückgegeben wird, umfasst den
    // Funktionsnamen, sodass die Anzahl der *Argumente* im Array
    // der Länge des Arrays minus 1 entspricht. Dieser Wert wird durch 4 dividiert, um die
    // Anzahl der Argumentgruppen zu erhalten.
```

```

var numArgGroups = ((itemArray.length - 1)/4);
// Für jede Argumentgruppe
for (i=0; i < numArgGroups; i++){

    // werden ein Komma und "NS4.0ref,IE4.0ref,other,dep" zur Liste der Argumenttypen hinzugefügt (da
diese
    // hypothetische Verhaltensfunktion mindestens vier
    // Argumente besitzt - die Netscape-Objektreferenz, die IE-Objekt-
    // -referenz, eine abhängige URL und vielleicht ein Eigenschaftswert
    // wie "show" oder "hide")
    // oder, falls noch keine Liste existiert, wird nur
    // "NS4.0ref,IE4.0ref,other,dep" hinzugefügt
var listOfArgTypes += ((listOfArgTypes)?",":"" ) + ↵
    "NS4.0ref,IE4.0ref,other,dep";
}
}

```

inspectBehavior()

Beschreibung

Diese Funktion untersucht den Funktionsaufruf auf ein zuvor angewendetes Verhalten im Benutzerdokument und setzt die Optionen im Dialogfeld **Parameter** auf die entsprechenden Werte. Wenn die Funktion `inspectBehavior()` nicht definiert ist, werden die Standardwerte der Optionen angezeigt.

Hinweis: Die Funktion `inspectBehavior()` muss nur auf den Informationen basieren, die das Argument `applyBehaviorString` an sie übergibt. Versuchen Sie nicht, im Rahmen dieser Funktion weitere Informationen über das Benutzerdokument abzurufen (zum Beispiel mit `dreamweaver.getDocumentDOM()`).

Argumente

`applyBehaviorString`

Dieses Argument ist der String, der von der Funktion `applyBehavior()` zurückgegeben wird.

Hinweis: Wenn das HTML-Element Code wie `'onClick="someBehavior(); return document.MM_returnValue;"` enthält und Sie ein neues Verhalten aus dem Verhaltensmenü hinzufügen, ruft Dreamweaver `inspectBehavior()` auf, sobald die neue Verhaltensbenutzeroberfläche eingeblendet wird, und übergibt einen leeren String als Parameter. Folglich muss der Parameter `applyBehaviorString` unbedingt geprüft werden. Sehen Sie hierzu das nachfolgende Beispiel:

```

function inspectBehavior(enteredStr){
    if(enteredStr){
        // hier wird die Arbeit vorgenommen
    }
}

```

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Das folgende Beispiel für die Funktion `inspectBehavior()` aus der Datei `Display Status Message.htm` füllt das Meldungsfeld im Dialogfeld **Parameter** mit der Meldung aus, die der Benutzer ausgewählt hat, als das Verhalten ursprünglich angewendet wurde:

```

function inspectBehavior(msgStr){
    var startStr = msgStr.indexOf('"') + 1;
    var endStr = msgStr.lastIndexOf('"');
    if (startStr > 0 && endStr > startStr) {
        document.theForm.message.value = ↵
        unescape(msgStr.substring(startStr,endStr));
    }
}

```

Hinweis: Weitere Informationen über die Funktion `unescapeQuotes()` finden Sie in der Datei **dwscripts.js** im Ordner **Configuration/Shared/Common/Scripts/CMN**.

windowDimensions()

Beschreibung

Diese Funktion legt die Abmessungen für das Dialogfeld **Parameter** fest. Wenn diese Funktion nicht definiert ist, werden die Abmessungen automatisch berechnet.

Hinweis: Verwenden Sie diese Funktion nur dann, wenn das Dialogfeld **Parameter** größer als 640 x 480 Pixel sein soll.

Argumente

platform

Der Wert des Arguments ist entweder "macintosh" oder "windows", abhängig von der verwendeten Plattform.

Rückgabewerte

Ein String der Form "widthInPixels,heightInPixels".

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da die Schaltflächen für **OK** und **Abbrechen** nicht mit einbezogen sind. Wenn mit den zurückgegebenen Abmessungen nicht für alle Optionen Platz vorhanden ist, werden Bildlaufleisten angezeigt.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Parameter-Dialogfelds auf 648 x 520 Pixel eingestellt:

```
function windowDimensions(){  
    return "648,520";  
}
```

Kapitel 17: Serververhalten

In Dreamweaver CS3 können Benutzer ihren Dokumenten verschiedene Serververhalten hinzufügen, um Datensätze entsprechend bestimmten Kriterien zu filtern, durch Datensätze zu navigieren, Detaillisten mit Detailseiten zu verknüpfen, Datensätze in eine Ergebnisliste einzufügen und andere Server-Tasks auszuführen. Fügt ein Dreamweaver-Benutzer mehrmals denselben Laufzeitcode in Dokumente ein, können Sie mithilfe einer neu erstellten Erweiterung die Aktualisierung eines Dokuments mit diesen häufig verwendeten Codeblöcken automatisieren. Ausführliche Informationen zur Arbeit mit dem Serververhaltensgenerator zur Implementierung benutzerdefinierter Serververhalten finden Sie unter „Benutzerdefinierte Serververhalten hinzufügen“ im Handbuch *Erste Schritte mit Dreamweaver*. Im aktuellen Kapitel finden Sie Informationen zum Arbeiten mit den entsprechenden Serververhaltensdateien und Funktionen, die für die Interaktion mit bestehenden Serververhalten verfügbar sind. Informationen zu den einzelnen Funktionen finden Sie unter „Serververhalten-Funktionen“ und im Abschnitt zu den Extension Data Manager-Funktionen im *Dreamweaver API-Referenzhandbuch*. Dreamweaver unterstützt zurzeit Serververhaltenserweiterungen, die Laufzeitcode für die folgenden Servermodelle hinzufügen: ASP.NET/C#, ASP.NET/VisualBasic, ASP/JavaScript, ASP/VBScript, ColdFusion, JSP und PHP/MySQL.

In diesem Abschnitt werden folgende Begriffe verwendet:

Die Serververhaltenserweiterung ist die Schnittstelle zwischen serverbasiertem Code und Dreamweaver. Eine Serververhaltenserweiterung besteht aus JavaScript, HTML und Extension Data Markup Language (EDML), d. h. XML, die eigens für Erweiterungsdaten erstellt wird. Beispieldateien, die nach Servermodell angeordnet sind, finden Sie in Ihrem Installationsverzeichnis im Ordner **Configuration\ServerBehaviors**. Wenn Sie das Skript für eine Erweiterung schreiben, leiten Sie Dreamweaver mit `dwscripts.applySB()` an, die EDML-Dateien zu lesen, die Komponenten Ihrer Erweiterung abzurufen und die entsprechenden Codeblöcke in das Dokument des Benutzers einzufügen.

Serververhaltensinstanz Wenn Dreamweaver Codeabschnitte in das Dokument des Benutzers einfügt, stellt der eingefügte Code eine Instanz des Serververhaltens dar. Die meisten Serververhalten kann der Benutzer mehrmals anwenden, sodass sich mehrere Serververhaltensinstanzen ergeben. Jede Serververhaltensinstanz wird im Bedienfeld **Serververhalten** der Dreamweaver-Benutzeroberfläche aufgeführt.

Laufzeitcode ist die Gruppe der Codeblöcke, die einem Dokument beim Anwenden des Serververhaltens hinzugefügt werden. Diese Codeblöcke enthalten normalerweise auch serverbasierten Code, z. B. ein in die Tags `<% . . . %>` eingeschlossenes ASP-Skript.

Mitglieder Ihre Serververhaltenserweiterung fügt Codeblöcke in das Dokument des Benutzers ein. Ein Codeblock ist ein einzelner, zusammenhängender Skriptblock, wie etwa ein Server-Tag, ein HTML-Tag oder ein Attribut, das einer Webseite spezielle Serverfunktionalität verleiht. Eine EDML-Datei definiert jeden Codeblock als Mitglied. Alle Mitglieder für ein bestimmtes Serververhalten werden in einer Gruppe zusammengefasst.

Hinweis: Weitere Informationen über Mitglieder, Mitgliedergruppen und den Aufbau von Dreamweaver EDML-Dateien finden Sie unter „Extension Data Markup Language“ auf Seite 234.

Dreamweaver-Architektur

Wenn Sie mit dem Serververhaltensgenerator eine Dreamweaver-spezifische Erweiterung erstellen, legt Dreamweaver mehrere Dateien an (EDML- und HTML-Skriptdateien), die das Einfügen von Serververhaltens-Code in ein Dreamweaver-Dokument unterstützen. Einige Verhalten verweisen außerdem auf JavaScript-Dateien und ermöglichen so weitere Funktionalitäten. Die Architektur vereinfacht Ihre API-Implementierung und behandelt Ihren Laufzeitcode unabhängig von dessen Einsatz in Dreamweaver. In diesem Abschnitt finden Sie Informationen zum Bearbeiten dieser Dateien.

Serververhaltensordner und -dateien

Die Benutzeroberfläche (UI) für jedes Serververhalten befindet sich im Ordner **Configuration\ServerBehaviors\Servermodell-Name**, wobei *Servermodell-Name* einer der folgenden Servertypen ist: ASP.NET_Csharp, ASP.NET_VB (Visual Basic), ASP_Js (JavaScript), ASP_Vbs (VBScript), ColdFusion, JSP, PHP_MySQL oder Shared (servermodellübergreifende Implementierungen).

Extension Data Markup Language

Dreamweaver generiert bei Verwendung des Serververhaltensgenerators zwei EDML-Dateien: eine Gruppen-EDML-Datei und eine Mitglieder-EDML-Datei, entsprechend den Namen, die Sie im Serververhaltensgenerator angeben. Die Gruppen-datei enthält die relevanten Mitglieder, d. h. die Codeblöcke, während die Gruppen die Mitglieder definieren, aus denen sich ein Serververhalten zusammensetzt.

Gruppendateien

Im Allgemeinen enthalten die Gruppendateien eine Liste der Mitglieder, während die Mitgliederdateien alle vom jeweiligen Servermodell abhängigen Mitgliederdaten enthalten. Mitgliederdateien können von mehreren Erweiterungen verwendet werden, daher verweisen mehrere Gruppendateien möglicherweise auch auf eine einzige Mitgliederdatei.

Das nachfolgende Beispiel zeigt eine vereinfachte Struktur der Gruppen-EDML-Datei für ein Serververhalten: Eine vollständige Liste der Elemente und Attribute finden Sie unter „EDML-Datei-Tags gruppieren“ auf Seite 246.

```
<group serverBehavior="Go To Detail Page.htm" dataSource="Recordset.htm">
  <groupParticipants selectParticipant="goToDetailPage_attr">
    <groupParticipant name="moveTo_declareParam" 0partType="member"/>
    <groupParticipant name="moveTo_keepParams" 0partType="member"/>
    <groupParticipant name="goToDetailPage_attr" partType="identifizier" />
  </groupParticipants>
</group>
```

Im Block-Tag `groupParticipants` werden durch `groupParticipant`-Tags die einzelnen EDML-Mitgliederdateien angegeben, die die zu verwendenden Codeblöcke enthalten. Der Wert des Attributs `name` gibt den Namen der Mitglieder-datei ohne die Erweiterung „.edml“ an (z. B. das Attribut `moveTo_declareParam`).

Mitgliederdateien

Ein Mitglied repräsentiert einen einzelnen Codeblock auf der Seite, z. B. ein Server-Tag, ein HTML-Tag oder ein Attribut. Eine Mitgliederdatei muss in einer Gruppendatei aufgeführt werden, damit der Autor eines Dreamweaver-Dokuments darauf zugreifen kann. Mehrere Gruppendateien können eine einzelne Mitgliederdatei verwenden.

Die EDML-Datei **moveTo_declareParam.edml** beispielsweise enthält folgenden Code:

```
<participant>
  <quickSearch><![CDATA[MM_paramName]]></quickSearch>
  <insertText location="aboveHTML+80">
<![CDATA[
<% var MM_paramName = ""; %>
]]>
    </insertText>
    <searchPatterns whereToSearch="directive">
      <searchPattern><![CDATA[/var\s*MM_paramName/]]></searchPattern>
    </searchPatterns>
</participant>
```


Wenn Dreamweaver ein Serververhalten in ein Dokument einfügt, benötigt es u. a. detaillierte Informationen zur Position, an der der Code eingefügt werden soll, zum Aufbau des Codes und zu den Parametern bzw. den Daten, die der Autor zur Laufzeit ersetzt hat. Jede Mitglieder-EDML-Datei enthält diese Details für jeden Codeblock. Dazu gehören folgende Daten:

- Der Code und der Einfügeort der eindeutigen Instanz werden durch die Parameter des `insertText`-Tags definiert, wie im folgenden Beispiel dargestellt:

```
<insertText location="aboveHTML+80">
```

- Das folgende Beispiel zeigt, wie Sie Instanzen erkennen, die sich bereits auf der Seite befinden und vom `searchPatterns`-Tag definiert sind:

```
<searchPatterns whereToSearch="directive">
  <searchPattern><![CDATA[/var\s*MM_paramName/]]></searchPattern>
</searchPatterns>
```

Im Block-Tag `searchPatterns` enthält jedes `searchPattern`-Tag ein Suchmuster für die Suche von Laufzeitcode-Instanzen und das Extrahieren bestimmter Parameter. Weitere Informationen finden Sie unter „Serververhaltenstechniken“ auf Seite 268.

Die Skriptdatei

Für jedes Serververhalten wird auch eine HTML-Datei mit den Funktionen und Links zu den Skripten angelegt, mit denen die Integration des zugehörigen Codes mit der Dreamweaver-Benutzeroberfläche verwaltet wird. Informationen zu den Bearbeitungsfunktionen für diese Datei finden Sie unter „Implementierungsfunktionen für das Serververhalten“ auf Seite 242.

Einfaches Beispiel für ein Serververhalten

Dieses Beispiel erläutert die verschiedenen Schritte beim Erstellen eines neuen Serververhaltens, die von Dreamweaver dafür angelegten Dateien und das Bearbeiten dieser Dateien. Ausführliche Informationen zur Arbeit mit dem Serververhaltensgenerator finden Sie unter „Benutzerdefinierte Serververhalten hinzufügen“ im Handbuch *Erste Schritte mit Dreamweaver*. Im Beispiel wird durch einen ASP-Server die Meldung „Hello World“ angezeigt. Bei diesem Beispiel gibt es nur ein Mitglied (ein einziges ASP-Tag), und es werden keine Änderungen oder sonstigen Bearbeitungen auf der Seite vorgenommen.

Zum Erstellen des Verhaltens gehen Sie wie folgt vor:

- „Dynamischen Seiteninhalt erstellen“ auf Seite 235
- „Neues Serververhalten definieren“ auf Seite 236
- „Einzufügenden Code definieren“ auf Seite 236

Dynamischen Seiteninhalt erstellen

Zunächst erstellen Sie ein neues ASP-Dokument.

So erstellen Sie ein neues dynamisches Seitendokument:

- 1 Wählen Sie in Dreamweaver **Datei > Neu**.
- 2 Wählen Sie im Dialogfeld **Neues Dokument** die Option **Kategorie: Dynamische Seite** und **Dynamische Seite: ASP-JavaScript**.
- 3 Klicken Sie auf **Erstellen**.

Neues Serververhalten definieren

Als Nächstes definieren Sie das neue Serververhalten.

Definieren Sie mit dem Serververhaltensgenerator das neue Serververhalten:

Hinweis: Ist das Bedienfeld **Serververhalten** nicht geöffnet und sichtbar, wählen Sie **Fenster > Serververhalten**.

- 1 Klicken Sie im Bedienfeld **Serververhalten** auf die Schaltfläche mit dem Pluszeichen (+), und wählen Sie die Option **Neues Serververhalten**.
- 2 Wählen Sie im Dialogfeld **Neues Serververhalten** die Option **Dokumenttyp: ASP-JavaScript** und **Name: Hello World**. (Lassen Sie die Option **Vorhandenes Serververhalten kopieren** deaktiviert.)
- 3 Klicken Sie auf **OK**.

Einzufügenden Code definieren

Zum Schluss definieren Sie den einzufügenden Code.

So definieren Sie den einzufügenden Code:

- 1 Klicken Sie auf die Schaltfläche mit dem Pluszeichen (+) für **Einzufügende Codeblöcke**.
- 2 Geben Sie im Dialogfeld **Neuen Codeblock erstellen** Folgendes ein: `Hello_World_block1` (in manchen Fällen wird dies von Dreamweaver automatisch vorgenommen).
- 3 Klicken Sie auf **OK**.
- 4 Geben Sie im Feld **Codeblock** Folgendes ein: `<% Response.Write("Hello World") %>.`
- 5 Wählen Sie im Popupmenü **Code einfügen** die Option **Relativ zur Auswahl**, sodass der Benutzer bestimmen kann, an welcher Stelle im Dokument der Code eingefügt wird.
- 6 Wählen Sie im Popupmenü **Relative Position** die Option **Nach der Auswahl**.
- 7 Klicken Sie auf **OK**.

Jetzt wird im Bedienfeld **Serververhalten** im Plusmenü (+) das neue Serververhalten in der Popupliste angezeigt. Das Unterverzeichnis **Configuration\ServerBehaviors\ASP_Js** im Installationsverzeichnis für Ihre Dreamweaver-Dateien enthält nun die folgenden drei Dateien:

- Die Gruppendatei: **Hello World.edml**
- Die Mitgliederdatei: **Hello World_block1.edml**
- Eine Skriptdatei: **Hello World.htm**

Hinweis: Wenn Sie in einer Umgebung mit mehreren Benutzern arbeiten, sind diese Dateien im Ordner **Anwendungsdaten** abgelegt.

API-Funktionen des Serververhaltens aufrufen

Die API-Funktionen des Serververhaltens werden unter folgenden Umständen aufgerufen:

- Die Funktion `findServerBehaviors()` wird aufgerufen, wenn ein Dokument geöffnet wird. Sie wird erneut aufgerufen, wenn ein Mitglied bearbeitet wird. Sie durchsucht das Dokument des Benutzers nach Instanzen des Serververhaltens. Für jede gefundene Instanz erstellt die Funktion `findServerBehaviors()` ein JavaScript-Objekt und verknüpft mithilfe von JavaScript-Eigenschaften Statusinformationen damit.
- Falls diese Funktion implementiert wurde, ruft Dreamweaver die Funktion `analyzeServerBehavior()` für jede Verhaltensinstanz auf, die im Benutzerdokument gefunden wird, nachdem alle `findServerBehaviors()`-Funktionen aufgerufen wurden.

Wenn die Funktion `findServerBehaviors()` ein Verhaltensobjekt erstellt, legt es in der Regel die vier Eigenschaften (`incomplete`, `participants`, `selectedNode` und `title`) fest. In manchen Fällen ist es jedoch einfacher, das Festlegen einiger Eigenschaften so lange zu verzögern, bis alle anderen Serververhalten Instanzen ihrer selbst finden. Das Verhalten

Zum nächsten Datensatz verschieben beispielsweise hat zwei Mitglieder: ein Hyperlinkobjekt und ein Datensatzgruppenobjekt. Anstatt nun das Datensatzgruppenobjekt in seiner Funktion `findServerBehaviors()` zu suchen, ist es einfacher, zu warten, bis die Funktion `findServerBehaviors()` des Datensatzgruppenverhaltens ausgeführt wird, da die Datensatzgruppe alle Instanzen ihrer selbst findet.

Wenn die Funktion `analyzeServerBehavior()` des Verhaltens **Zum nächsten Datensatz verschieben** aufgerufen wird, ruft es ein Array ab, das alle Verhaltensobjekte des Servers in dem Dokument enthält. Die Funktion kann im Array nach seinem Datensatzobjekt suchen.

Manchmal wird während einer Analyse ein einzelnes Tag im Dokument des Benutzers von zwei oder mehr Verhalten als Instanz dieses Verhaltens identifiziert. So könnte zum Beispiel die Funktion `findServerBehaviors()` für das Verhalten **Dynamisches Attribut** eine Instanz dieses Verhaltens entdecken, die im Dokument des Benutzers mit einem `input`-Tag verknüpft ist. Gleichzeitig könnte es sein, dass die Funktion `findServerBehaviors()` für das Verhalten **Dynamisches Textfeld** denselben Namen in demselben `input`-Tag betrachtet und eine Instanz des Verhaltens **Dynamisches Textfeld** gefunden hat. Im Bedienfeld **Serververhalten** werden als Folge sowohl ein Block für das dynamische Attribut als auch das dynamische Textfeld aufgeführt. Um dies zu verhindern, müssen die `analyzeServerBehavior()`-Funktionen alle bis auf eines dieser Serververhalten löschen.

Hierzu kann bei der Funktion `analyzeServerBehavior()` die Eigenschaft `deleted` bei jedem Serververhalten auf `true` gesetzt werden. Hat die Eigenschaft `deleted` auch dann noch den Wert `true`, wenn Dreamweaver die `analyzeServerBehavior()`-Funktionen aufgerufen hat, wird das Verhalten aus der Liste gelöscht.

- Wenn der Benutzer im Bedienfeld **Serververhalten** auf die Schaltfläche mit dem Pluszeichen (+) klickt, wird das Popupmenü eingeblendet.

Dreamweaver sucht zunächst in dem Ordner, in dem sich auch die Verhaltensoptionen befinden, nach der Datei **ServerBehaviors.xml**, um den Inhalt des Menüs zu bestimmen. Die Datei **ServerBehaviors.xml** nimmt Bezug auf die HTML-Dateien, die im Menü angezeigt werden sollen.

Sollte die HTML-Datei, auf die verwiesen wird, ein `<title>`-Tag enthalten, wird der Inhalt des `<title>`-Tags im Menü angezeigt. Enthält die Datei **ServerBehaviors\ASP\Get_Js\Records.htm** beispielsweise das Tag `<title>Get More Records</title>`, wird *Get More Records* im Menü angezeigt.

Enthält die Datei kein `<title>`-Tag, wird der Dateiname im Menü angezeigt. Enthält beispielsweise die Datei **GetRecords.htm** kein `title`-Tag, wird „GetRecords“ im Menü angezeigt.

Sollte die Datei **ServerBehaviors.xml** nicht vorhanden sein oder der Ordner eine oder mehrere HTML-Dateien enthalten, die nicht in **ServerBehaviors.xml** aufgeführt sind, sucht Dreamweaver in jeder Datei nach einem `<title>`-Tag und verwendet dieses oder den Dateinamen im Menü.

Falls eine im Ordner **ServerBehaviors** enthaltene Datei nicht im Menü angezeigt werden soll, fügen Sie die folgende Anweisung als erste Zeile in die HTML-Datei ein:

```
<!-- MENU-LOCATION=NONE -->
```

- Wenn der Benutzer ein Element im Menü auswählt, wird die Funktion `canApplyServerBehavior()` aufgerufen. Wenn diese Funktion den Wert `true` zurückgibt, wird das Dialogfeld angezeigt. Wenn der Benutzer auf **OK** klickt, wird die Funktion `applyServerBehavior()` aufgerufen.
- Wenn der Benutzer auf ein bereits vorhandenes Serververhalten doppelklickt, um es zu bearbeiten, blendet Dreamweaver das Dialogfeld ein, führt ggf. die `onLoad`-Prozedur im `body`-Tag aus und ruft dann die Funktion `inspectServerBehavior()` auf. Die Funktion `inspectServerBehavior()` stellt die aktuellen Argumentwerte für die Formularelemente bereit. Wenn der Benutzer auf **OK** klickt, ruft Dreamweaver die Funktion `applyServerBehavior()` erneut auf.
- Klickt der Benutzer auf die Schaltfläche mit dem Minuszeichen (–), wird die Funktion `deleteServerBehavior()` aufgerufen. Die Funktion `deleteServerBehavior()` entfernt das Verhalten aus dem Dokument.
- Wendet der Benutzer auf ein zuvor ausgewähltes Serververhalten den Befehl **Ausschneiden** oder **Kopieren** an, übergibt Dreamweaver das Objekt, das das Serververhalten repräsentiert, an dessen `copyServerBehavior()`-Funktion. Die Funktion `copyServerBehavior()` fügt dem Serververhaltensobjekt alle zusätzlichen Eigenschaften hinzu, die später benötigt werden, um es einzufügen.

Sobald die Funktion `copyServerBehavior()` diesen Vorgang beendet hat, konvertiert Dreamweaver das Serververhaltensobjekt in ein Format, in dem es in die Zwischenablage kopiert werden kann. Dreamweaver löscht während der Konvertierung des Objekts alle Eigenschaften, die auf Objekte Bezug nehmen. Alle Eigenschaften des Objekts, die keine Zahl, keinen Booleschen Wert und keinen String darstellen, gehen verloren.

Wählt der Benutzer den Befehl **Einfügen**, entpackt Dreamweaver den Inhalt der Zwischenablage und generiert ein neues Serververhaltensobjekt. Das neu erstellte Objekt ist identisch mit dem Original, verfügt aber über keine Eigenschaften, die auf Objekte verweisen. Dreamweaver leitet das neue Serververhaltensobjekt an die Funktion `pasteServerBehavior()` weiter. Die Funktion `pasteServerBehavior()` fügt das Verhalten dem Dokument des Benutzers hinzu. Nachdem die Funktion `pasteServerBehavior()` den Vorgang abgeschlossen hat, ruft Dreamweaver die Funktion `findServerBehaviors()` auf, um eine neue Liste aller Serververhalten im Dokument des Benutzers zu erhalten.

Die Benutzer können Verhalten von einem Dokument in ein anderes kopieren. Die Funktionen `copyServerBehavior()` und `pasteServerBehavior()` sollten sich beim Austausch von Informationen nur auf die Eigenschaften auf dem Serverobjekt stützen.

Die API für Serververhalten

Sie können Serververhalten mit den folgenden API-Funktionen verwalten:

analyzeServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Versetzt Serververhalten in die Lage, ihre `incomplete`- und `deleted`-Eigenschaften zu setzen.

Nachdem die Funktion `findServerBehaviors()` für jedes Serververhalten auf der Seite aufgerufen wurde, wird ein Array aller Verhalten im Dokument des Benutzers angezeigt. Die Funktion `analyzeServerBehavior()` wird für jedes JavaScript-Objekt in diesem Array aufgerufen. Bei einem Verhalten des Typs **Dynamischer Text** würde Dreamweaver beispielsweise die Funktion `analyzeServerBehavior()` in `DynamicText.htm` (oder `DynamicText.js`) aufrufen.

Zum einen dient die Funktion `analyzeServerBehavior()` dazu, das Festlegen aller Eigenschaften (`incomplete`, `participants`, `selectedNode` und `title`) bei dem Verhaltensobjekt zum Abschluss zu bringen. Manchmal ist es einfacher, diese Aufgabe auszuführen, nachdem die Funktion `findServerBehaviors()` die vollständige Liste der Serververhalten im Dokument des Benutzers erstellt hat.

Zum anderen soll die Funktion `analyzeServerBehavior()` erkennen, ob zwei oder mehr Verhalten auf dasselbe Tag im Dokument des Benutzers Bezug nehmen. In diesem Fall löscht die `deleted`-Eigenschaft alle Verhalten bis auf eines aus dem Array.

Angenommen, die Serververhalten **Recordset1**, **DynamicText1** und **DynamicText2** befinden sich auf einer Seite. Beide **DynamicText**-Serververhalten benötigen **Recordset1**, um auf der Seite bestehen zu können. Nachdem die Serververhalten mit der Funktion `findServerBehaviors()` gefunden wurden, ruft Dreamweaver die Funktion `analyzeServerBehavior()` für die drei Serververhalten auf. Wenn die Funktion `analyzeServerBehavior()` für **DynamicText1** aufgerufen wird, durchsucht die Funktion das Array aller Serververhaltensobjekte auf der Seite nach dem Serververhaltensobjekt, das zu **Recordset1** gehört. Kann für **Recordset1** kein Serververhaltensobjekt gefunden werden, wird die `incomplete`-Eigenschaft auf `true` gesetzt. Ein daraufhin im Bedienfeld **Serververhalten** angezeigtes Ausrufezeichen informiert den Benutzer, dass ein Problem vorliegt. Wenn nun die Funktion `analyzeServerBehavior()` für **DynamicText2** aufgerufen wird, sucht die Funktion nach dem Objekt für **Recordset1**. Da **Recordset1** nicht von anderen Serververhalten abhängig ist, braucht es die Funktion `analyzeServerBehavior()` in diesem Beispiel nicht zu definieren.

Argumente

serverBehavior, [*serverBehaviorArray*]

- Das Argument *serverBehavior* ist ein JavaScript-Objekt, das für das zu analysierende Verhalten steht.
- Das Argument [*serverBehaviorArray*] ist ein Array von JavaScript-Objekten, stellvertretend für alle Serververhalten auf einer Seite.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

applyServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Liest Werte von den Formularelementen im Dialogfeld ein und fügt das Verhalten dem Dokument des Benutzers hinzu. Dreamweaver ruft diese Funktion auf, wenn der Benutzer im Dialogfeld **Serververhalten** auf **OK** klickt. Wenn diese Funktion korrekt abläuft, wird das Dialogfeld **Serververhalten** geschlossen. Arbeitet sie nicht korrekt, wird eine Fehlermeldung angezeigt, ohne dass das Dialogfeld **Serververhalten** geschlossen wird. Diese Funktion kann ein Dokument eines Benutzers bearbeiten.

Weitere Informationen finden Sie unter „*dwscripts.applySB()*“ auf Seite 243.

Argumente

serverBehavior

serverBehavior ist ein JavaScript-Objekt, das das Serververhalten repräsentiert. Es ist erforderlich, um ein bestehendes Verhalten zu verändern. Wenn es sich um ein neues Verhalten handelt, lautet das Argument *null*.

Rückgabewerte

Dreamweaver erwartet bei Erfolg einen leeren String oder bei Fehlschlag eine Fehlermeldung.

canApplyServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Bestimmt, ob ein Verhalten angewandt werden kann. Dreamweaver ruft diese Funktion auf, bevor das Dialogfeld **Serververhalten** angezeigt wird. Gibt die Funktion den Wert *true* zurück, wird das Dialogfeld **Serververhalten** angezeigt. Gibt die Funktion den Wert *false* zurück, wird das Dialogfeld **Serververhalten** nicht angezeigt und das Hinzufügen eines Serververhaltens wird abgebrochen.

Argumente

serverBehavior

serverBehavior ist ein JavaScript-Objekt, das für das Verhalten steht. Es ist erforderlich, um ein bestehendes Verhalten zu verändern. Wenn es sich um ein neues Verhalten handelt, lautet das Argument *null*.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert: *true*, wenn das Verhalten angewandt werden kann, andernfalls *false*.

copyServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Das Implementieren der Funktion `copyServerBehavior()` ist optional. Die Benutzer können Instanzen eines bestimmten Serververhaltens kopieren. Im nachfolgenden Beispiel wird diese Funktion für Datensatzgruppen implementiert. Wählt ein Benutzer im Bedienfeld **Serververhalten** oder **Bindungen** ein Serververhalten aus, so kann er über den Befehl **Kopieren** oder **Ausschneiden** das ausgewählte Verhalten in der Zwischenablage ablegen. Für Serververhalten, die diese Funktion nicht unterstützen, sind diese beiden Befehle nicht verfügbar. Weitere Informationen finden Sie unter „API-Funktionen des Serververhaltens aufrufen“ auf Seite 236.

Die Funktion `copyServerBehavior()` sollte zum Austauschen von Informationen mit der Funktion `pasteServerBehavior()` nur Verhaltensobjekteigenschaften berücksichtigen, die in Strings konvertiert werden können. In der Zwischenablage wird nur Rohtext gespeichert, sodass `participant`-Knoten im Dokument aufgelöst und der dabei entstehende Rohtext in einer sekundären Eigenschaft gespeichert werden sollten.

Hinweis: Die Funktion `pasteServerBehavior()` muss ebenfalls implementiert sein, um dem Benutzer die Möglichkeit zu geben, das Verhalten in ein beliebiges Dreamweaver-Dokument einzufügen.

Argumente

`serverBehavior`

- `serverBehavior` ist ein JavaScript-Objekt, das für das Verhalten steht.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das Verhalten erfolgreich in die Zwischenablage kopiert wird, andernfalls `false`.

deleteServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Entfernt das Verhalten aus dem Dokument des Benutzers. Diese Funktion wird aufgerufen, wenn ein Benutzer im Bedienfeld **Serververhalten** auf die Schaltfläche mit dem Minuszeichen (–) klickt. Sie kann das Dokument eines Benutzers bearbeiten.

Weitere Informationen finden Sie unter „`dwscripts.deleteSB()`“ auf Seite 244.

Argumente

`serverBehavior`

- `serverBehavior` ist ein JavaScript-Objekt, das für das Verhalten steht.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

displayHelp()

Beschreibung

Wenn diese Funktion definiert ist, wird die Schaltfläche **Hilfe** unterhalb der Schaltflächen **OK** und **Abbrechen** im Dialogfeld angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche **Hilfe** klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// die folgende Instanz von displayHelp() öffnet  
// in einem Browser eine Datei, die die Funktionsweise  
// der Erweiterung erläutert.  
function displayHelp(){  
    var myHelpFile = dw.getConfigurationPath() +  
        '/ExtensionsHelp/superDuperHelp.htm';  
    dw.browseDocument(myHelpFile);  
}
```

findServerBehaviors()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Durchsucht das Dokument des Benutzers nach Instanzen seiner selbst. Für jede gefundene Instanz erstellt die Funktion `findServerBehaviors()` ein JavaScript-Objekt und verknüpft damit Statusinformationen als JavaScript-Eigenschaften des Objekts.

Die vier erforderlichen Eigenschaften sind `incomplete`, `participants`, `title` und `selectedNode`. Sie können je nach Bedarf weitere Eigenschaften festlegen.

Weitere Informationen finden Sie unter „`dwscripts.findSBs()`“ auf Seite 242 und `dreamweaver.getParticipants()` im *Dreamweaver API-Referenzhandbuch*.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array mit JavaScript-Objekten. Die Länge des Arrays entspricht der Anzahl von Verhaltensinstanzen, die auf der Seite gefunden wurden.

inspectServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Bestimmt die Einstellungen des Dialogfelds **Serververhalten** auf der Grundlage des angegebenen Verhaltensobjekts. Dreamweaver ruft die Funktion `inspectServerBehavior()` auf, wenn ein Benutzer das Dialogfeld **Serververhalten** öffnet. Dreamweaver ruft diese Funktion nur dann auf, wenn ein Benutzer ein bestehendes Verhalten bearbeitet.

Argumente

serverBehavior

Das Argument *serverBehavior* ist ein JavaScript-Objekt, das für das Verhalten steht. Es handelt sich um das gleiche Objekt, das von `findServerBehaviors()` zurückgegeben wird.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

pasteServerBehavior()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Wenn die Funktion implementiert wird, können Benutzer Instanzen eines bestimmten Serververhaltens mit der Funktion `pasteServerBehavior()` einfügen. Wenn der Benutzer das Serververhalten einfügt, strukturiert Dreamweaver die Daten aus der Zwischenablage und generiert ein neues Verhaltensobjekt. Das neue Objekt ist mit dem ursprünglichen Objekt identisch. Die Verweiseigenschaften sind jedoch nicht mehr vorhanden. Dreamweaver leitet das neue Verhaltensobjekt an die Funktion `pasteServerBehavior()` weiter. Die Funktion `pasteServerBehavior()` bestimmt anhand der Eigenschaften des Verhaltensobjekts, was dem Dokument des Benutzers hinzugefügt werden soll. Die Funktion `pasteServerBehavior()` fügt dann das Verhalten dem Dokument des Benutzers hinzu. Nachdem `pasteServerBehavior()` den Vorgang abgeschlossen hat, ruft Dreamweaver die `findServerBehaviors()`-Funktionen auf, um eine neue Liste aller Serververhalten im Dokument des Benutzers zu erhalten.

Das Implementieren der Funktion `pasteServerBehavior()` ist optional. Weitere Informationen finden Sie unter „API-Funktionen des Serververhaltens aufrufen“ auf Seite 236.

Hinweis: Wenn Sie diese Funktion implementieren, müssen Sie auch die Funktion `copyServerBehavior()` implementieren.

Argumente

Verhalten

behavior ist ein JavaScript-Objekt, das das Verhalten repräsentiert.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true`, wenn das Verhalten erfolgreich aus der Zwischenablage eingefügt wurde, andernfalls `false`.

Implementierungsfunktionen für das Serververhalten

Diese Funktionen können innerhalb der HTML-Skriptdateien oder in den JavaScript-Dateien innerhalb dieser hinzugefügt oder bearbeitet werden.

dwscripts.findSBs()

Verfügbarkeit

Dreamweaver MX (diese Funktion ersetzt `findSBs()` aus früheren Versionen von Dreamweaver)

Beschreibung

Findet alle Instanzen eines Serververhaltens und alle Mitglieder auf der aktuellen Seite. Bestimmt den Titel, den Typ, das Mitglieder-Array, das Gewichtungs-Array, das Typen-Array, den `selectedNode`-Wert und das `incomplete`-Flag. Diese Funktion erstellt zudem ein Parameterobjekt, das eine Liste mit benutzerdefinierbaren Eigenschaften, wie Datensatzgruppe, Name und Spaltenname, enthält. Sie können dieses Array aus der Funktion `findServerBehaviors()` zurückgeben.

Argumente

serverBehaviorTitle

Das Argument *serverBehaviorTitle* ist ein optionaler Titelstring, der dann verwendet wird, wenn im EDML-Titel kein Titel angegeben ist (hilfreich für die Lokalisierung).

Rückgabewerte

Dreamweaver erwartet ein Array von JavaScript-Objekten, bei denen die erforderlichen Eigenschaften definiert wurden. Gibt ein leeres Array zurück, wenn keine Instanzen eines Serververhaltens auf der Seite angezeigt werden.

Beispiel

Der Code im folgenden Beispiel sucht nach allen Instanzen eines bestimmten Serververhaltens im aktuellen Benutzerdokument:

```
function findServerBehaviors() {  
    allMySBs = dwscripts.findSBs();  
    return allMySBs;  
}
```

dwscripts.applySB()

Verfügbarkeit

Dreamweaver MX (diese Funktion ersetzt `applySB()` aus früheren Versionen von Dreamweaver).

Beschreibung

Fügt Laufzeitcode für das Serververhalten ein oder aktualisiert ihn. Wenn das *sObj*-Argument `null` beträgt, fügt sie neuen Laufzeitcode ein. Andernfalls aktualisiert sie bestehenden Laufzeitcode, der durch das Objekt *sObj* angezeigt wird. Benutzereigenschaften sollten als Eigenschaften in einem JavaScript-Objekt festgelegt und als *paramObj* eingesetzt werden. Diese Einstellungen sollten mit allen Argumenten übereinstimmen, die als `@@paramName@@` im EDML-Einfügetext deklariert werden.

Argumente

paramObj, *sObj*

- Das Argument *paramObj* ist das Objekt, das die Benutzerparameter enthält.
- Das Argument *sObj* ist das vorherige Serververhaltensobjekt, wenn Sie ein bestehendes Serververhalten aktualisieren, andernfalls ist es `null`.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. `true` wenn das Serververhalten in das Benutzerdokument aufgenommen werden konnte, andernfalls `false`.

Beispiel

Im folgenden Beispiel füllen Sie das Objekt *paramObj* mit der Benutzereingabe und rufen dann `dwscripts.applySB()` auf, wobei Sie die Eingabe und Ihr Serververhalten, *sObj*, einsetzen:

```
function applyServerBehaviors(sObj) {  
    // all UI-Werte hier abrufen...  
    paramObj = new Object();  
    paramObj.rs= rsName.value;  
    paramObj.col = colName.value;  
    paramObj.url = urlPath.value;  
    paramObj.form__tag = formObj;  
    dwscripts.applySB(paramObj, sObj);  
}
```

dwscripts.deleteSB()

Verfügbarkeit

Dreamweaver MX (diese Funktion ersetzt `deleteSB()` aus früheren Versionen von Dreamweaver)

Beschreibung

Löscht alle Mitglieder der Serververhaltensinstanz `sObj`. Das gesamte Mitglied wird gelöscht, es sei denn, die EDML-Datei enthält spezielle Löschanweisungen in Verbindung mit dem `delete`-Tag. Es werden keine Mitglieder gelöscht, die mehreren Serververhaltensinstanzen angehören (Referenzzahl > 1).

Argumente

sObj

- Das Argument *sObj* ist die Objektinstanz des Serververhaltens, die Sie aus dem Dokument des Benutzers entfernen möchten.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Im folgenden Beispiel werden alle Mitglieder des Serververhaltens `sObj` gelöscht. Ausgenommen sind Mitglieder, die vom `delete`-Tag der EDML-Datei geschützt sind.

```
function deleteServerBehavior(sObj) {  
    dwscripts.deleteSB(sObj);  
}
```

EDML-Dateien bearbeiten

Beim Bearbeiten einer Datei müssen die Dreamweaver-Codekonventionen befolgt werden. Insbesondere sind die Abhängigkeiten zwischen den Elementen zu beachten. Wenn Sie beispielsweise die einzufügenden Tags aktualisieren, müssen Sie möglicherweise auch die Suchmuster aktualisieren.

Hinweis: EDML-Dateien wurden in Dreamweaver MX neu eingeführt. Wenn Sie mit früheren Serververhalten arbeiten, finden Sie Informationen in den Handbüchern Dreamweaver erweitern der Vorversionen.

Reguläre Ausdrücke

Sie müssen mit regulären Ausdrücken, die in JavaScript 1.5 eingesetzt werden, vertraut sein. Darüber hinaus müssen Sie ihre Verwendung in Serververhalten-EDML-Dateien kennen. Reguläre Ausdrücke können beispielsweise nicht in `quick-search`-Werten verwendet werden. Sie werden jedoch in `searchPattern`-Tags eingesetzt, um Daten zu suchen und zu extrahieren.

Reguläre Ausdrücke beschreiben Textstrings durch Zeichen, denen spezielle Bedeutungen zugeordnet sind (sog. Metazeichen), um Text zu klassifizieren, aufzugliedern und nach vordefinierten Regeln zu verarbeiten. Reguläre Ausdrücke sind ein leistungsstarkes Hilfsmittel zum Durchsuchen und Bearbeiten von Text, weil sie ein standardisiertes Mittel zur Darstellung von Textmustern sind.

Gute Handbücher zu JavaScript 1.5 widmen regulären Ausdrücken einen eigenen Abschnitt oder ein eigenes Kapitel. In diesem Abschnitt wird erläutert, wie Dreamweaver Serververhalten-EDML-Dateien mithilfe von regulären Ausdrücken Argumente in Ihrem Laufzeitcode finden und deren Werte extrahieren können. Wann immer ein Benutzer ein Serververhalten bearbeitet, müssen die vorherigen Argumentwerte aus den Instanzen des Laufzeitcodes extrahiert werden. Für die Extraktion werden reguläre Ausdrücke verwendet.

Sie sollten mit einigen Metazeichen und Metasequenzen (bestimmte Zeichengruppierungen) vertraut sein, die in Serververhalten-EDML-Dateien eingesetzt werden. Siehe dazu nachfolgende Tabelle:

Reguläre Ausdrücke	Beschreibung
\	Sonderzeichen ignorieren. Beispiel: \. hebt die Metazeichenfunktion des Punkts auf, \\\ hebt die Metazeichenfunktion des Schrägstrichs auf, und \\) hebt die Metazeichenfunktion der Klammer auf. Das Escape-Zeichen bewirkt, dass der Parser das nachfolgende Zeichen nicht als Metazeichen bewertet.
/.../i	Bei der Suche nach der Metasequenz die Groß-/Kleinschreibung ignorieren.
(...)	Einen parenthetischen Teilausdruck innerhalb der Metasequenz erstellen.
\\s*	Nach Leerräumen suchen.

Das EDML-Tag `<searchPatterns whereToSearch="directive">` gibt an, dass Laufzeitcode durchsucht werden muss. Jedes Subtag `<searchPattern>...</searchPattern>` definiert ein zu ermittelndes Muster im Laufzeitcode. Somit verfügt das Beispiel „Redirect If Empty“ über zwei Muster.

Zum Extrahieren von Argumentwerten aus `<% if (@rs@@.EOF) Response.Redirect("@@new__url@@"); %>`, schreiben Sie im folgenden Beispiel einen regulären Ausdruck, der alle vorkommenden `rs-` und `new__url-`Strings ermittelt:

```
<searchPattern paramNames="rs,new__url">
    /if d ((\w+)\.EOF\\) Response\\.Redirect\\("([^\r\n]*)")\\)/i
</searchPattern>
```

Bei diesem Prozess wird das Dokument des Benutzers durchsucht, und bei Übereinstimmungen werden die jeweiligen Argumentwerte extrahiert. Der erste parenthetische Teilausdruck `(\\w+)` extrahiert den Wert für `rs`. Der zweite Teilausdruck `([^\r\n]*)` extrahiert den Wert für `new__url`.

Hinweis: Die Zeichenfolge `"([^\r\n]*)"` stimmt mit allen Zeichen überein, die nicht für einen Zeilenvorschub stehen (Macintosh- und Windows-Systeme).

Hinweise zur EDML-Struktur

Sie sollten für Ihre Serververhaltensgruppe immer einen eindeutigen Dateinamen vergeben. Wenn nur eine Gruppendatei eine verknüpfte Mitgliederdatei verwendet, muss der Name der Mitgliederdatei mit dem Gruppennamen übereinstimmen. Wenn diese Konvention befolgt wird, arbeitet die Gruppendatei `updateRecord.edml` mit der Mitgliederdatei `updateRecord_init.edml`. Wird eine Mitgliederdatei von mehreren Serververhaltensgruppen verwendet, müssen Sie eindeutige, beschreibende Namen vergeben.

Hinweis: Die Zuordnung der EDML-Dateien ist unabhängig von der Ordnerstruktur. Stellen Sie sicher, dass eindeutige Dateinamen verwendet werden. Bedingt durch Einschränkungen für Macintosh sollten Dateinamen einschließlich der `.edml`-Erweiterung nicht länger als 31 Zeichen sein.

Der Laufzeitcode für Ihr Serververhalten ist in den EDML-Dateien gespeichert. Der EDML-Parser muss Ihren Laufzeitcode von EDML-Markup unterscheiden können. Aus diesem Grund muss vor und nach dem Laufzeitcode das Tag `CDATA` stehen. Das Tag `CDATA` steht für „Character Data“ (Zeichendaten), d. h. für jeglichen Text, der kein EDML-Markup ist. Wenn Sie das Tag `CDATA` verwenden, interpretiert der EDML-Parser den enthaltenen Text nicht als Markup, sondern als Block aus einfachem Text. Die mit `CDATA` markierten Blöcke beginnen mit `<![CDATA[` und enden mit `]]>`.

Wenn Sie den Text „Hello, World“ einfügen, können Sie ganz einfach Ihr EDML festlegen (siehe dazu nachfolgendes Beispiel):

```
<insertText>Hello, World</insertText>
```

Wenn Sie jedoch Text mit Tags einfügen, wie z. B. `<img src='foo.gif'>`, kann der EDML-Parser ihn nicht immer richtig auswerten. In diesem Fall empfiehlt es sich, den Text, wie nachfolgend beschrieben, in `CDATA` zu integrieren:

```
<insertText><![CDATA[<img src='foo.gif'>]]></insertText>
```

Der ASP-Laufzeitcode wird hier von einem CDATA-Tagpaar umgeben:

```
<![CDATA [  
    <% if (@rs@@.EOF) Response.Redirect("@@new__url@"); %>  
]]
```

Aufgrund des Tags CDATA werden die ASP-Tags <%= %> und deren Inhalt nicht verarbeitet. Stattdessen erhält der Extension Data Manager (EDM) folgenden, nicht interpretierten Text:

```
<% if (Recordset1.EOF) Response.Redirect("http://www.Adobe.com"); %>
```

Bei den nachfolgenden Beispielen von EDML-Definitionen sind die empfohlenen Positionen für CDATA-Tags angegeben.

EDML-Datei-Tags gruppieren

Diese Tags und Attribute sind innerhalb der EDML-Gruppendateien gültig.

<group>

Beschreibung

Dieses Tag enthält alle Spezifikationen für eine Mitgliedergruppe.

Übergeordnet

Keine

Typ

Block-Tag

Erforderlich

Ja

<group>-Attribute

Die folgenden Elemente sind gültige Attribute des group-Tags.

version

Beschreibung

Dieses Attribut definiert, auf welche Version der Dreamweaver-Serververhaltensverarbeitung das aktuelle Serververhalten abzielt. Bei Dreamweaver CS3 lautet die Versionsnummer 9. Wenn keine Version angegeben ist, geht Dreamweaver von Version 7 aus. Bei dieser Version von Dreamweaver verfügen alle Gruppen und Mitglieder, die vom Serververhaltensgenerator erstellt werden, über das Versionsattribut 9.0. Die Gruppenversion dieses Attributs ist derzeit nicht gültig.

Übergeordnet

group

Typ

Attribut

Erforderlich

Nein

serverBehavior

Beschreibung

Das Attribut `serverBehavior` gibt an, welches Serververhalten die Gruppe verwenden darf. Wird ein beliebiger `quick-search`-String der Gruppenmitglieder im Dokument gefunden, veranlasst das vom Attribut `serverBehavior` bezeichnete Serververhalten, dass Dreamweaver die Funktion `findServerBehaviors()` aufruft.

Sind einem Serververhalten mehrere Gruppen zugeordnet, muss das Serververhalten entscheiden, welche Gruppe zu verwenden ist.

Übergeordnet

group

Typ

Attribut

Erforderlich

Nein

Wert

Der Wert ist der genaue Name (ohne Pfadangabe) einer beliebigen Serververhalten-HTML-Datei im Ordner **Configuration\ServerBehaviors**:

```
<group serverBehavior="redirectIfEmpty.htm">
```

dataSource

Beschreibung

Diese erweiterte Funktion unterstützt neue Datenquellen, die in Dreamweaver aufgenommen werden können.

Je nach verwendeter Datenquelle können die verschiedenen Versionen eines Serververhaltens voneinander abweichen. Beispielsweise ist das Serververhalten „Repeat Region“ für die Standarddatenquelle `Recordset.htm` ausgelegt. Wird Dreamweaver um einen neuen unterstützten Datenquellentyp erweitert (z. B. um COM-Objekte), können Sie in einer Gruppen-datei mit einer anderen Implementierung von „Repeat Region“ die neue Datenquelle deklarieren (beispielsweise mit `dataSource="COM.htm"`). Bei Auswahl der neuen Datenquelle wendet das Serververhalten „Repeat Region“ dann die neue Implementierung von „Repeat Region“ an.

Übergeordnet

group

Typ

Attribut

Erforderlich

Nein

Wert

Der genaue Name einer Datenquellendatei im Ordner **Configuration\DataSourcees**:

```
<group serverBehavior="Repeat Region.htm" dataSource="myCOMdataSource.htm">
```

Diese Gruppe definiert eine neue Implementierung des Serververhaltens „Repeat Region“, die bei Auswahl der COM-Datenquelle zu verwenden ist. In der Funktion `applyServerBehaviors()` können Sie über die Eigenschaft `MM_dataSource` für das Parameterobjekt festlegen, dass diese Gruppe angewendet werden soll:

```
function applyServerBehavior(ssRec) {
    var paramObj = new Object();
    paramObj.rs = getComObjectName();
    paramObj.MM_dataSource = "myCOMdataSource.htm";

    dwscripts.applySB(paramObj, sbObj);
}
```

subType**Beschreibung**

Diese erweiterte Funktion unterstützt mehrere Implementierungen eines Serververhaltens.

Je nach Auswahl des Benutzers können die verschiedenen Versionen eines Serververhaltens voneinander abweichen. Wird bei mehreren relevanten Gruppdateien ein Serververhalten angewendet, kann die richtige Gruppdatei durch Übergabe eines `subType`-Werts festgelegt werden. Angewendet wird dann die Gruppe mit dem betreffenden Wert `subType`.

Übergeordnet

```
group
```

Typ

Attribut

Erforderlich

Nein

Wert

Der Wert ist ein eindeutiger String, der festlegt, welche Gruppe angewendet wird:

```
<group serverBehavior="myServerBehavior.htm" subType="longVersion">
```

Dieses Gruppenattribut definiert die lange Version des Subtyps `myServerBehavior`. Es liegt auch eine Version mit dem Attribut `subType="shortVersion"` vor. In der Funktion `applyServerBehaviors()` können Sie mithilfe der Eigenschaft `MM_subType` für das Parameterobjekt festlegen, welche Gruppe angewendet werden soll:

```
function applyServerBehavior(ssRec) {
    var paramObj = new Object();
    if (longVersionChecked) {
        paramObj.MM_subType = "longVersion";
    } else {
        paramObj.MM_subType = "shortVersion";
    }
    dwscripts.applySB(paramObj, sbObj);
}
```

<title>

Beschreibung

Der String, der im Bedienfeld **Serververhalten** für die einzelnen Instanzen des im aktuellen Dokument gefundenen Serververhalten angezeigt wird.

Übergeordnet

group

Typ

Block-Tag

Erforderlich

Nein

Wert

Der Wert ist ein einfacher Textstring, der Argumentnamen enthalten kann, mit dem jede Instanz eindeutig bezeichnet wird:

```
<title>Redirect If Empty (@@recordsetName@@)</title>
```

<groupParticipants>

Beschreibung

Dieses Tag enthält ein Array von groupParticipant-Deklarationen.

Übergeordnet

group

Typ

Block-Tag

Erforderlich

Ja.

<groupParticipants>-Attribute

Die folgenden Elemente sind gültige Attribute des groupParticipants-Tags.

selectParticipant

Beschreibung

Gibt an, welches Mitglied im Dokument ausgewählt und hervorgehoben werden soll, wenn eine Instanz im Bedienfeld **Serververhalten** ausgewählt wird. Die in diesem Bedienfeld aufgeführten Serververhalten-Instanzen sind nach dem ausgewählten Mitglied sortiert. Legen Sie daher das Attribut selectParticipant fest, auch wenn das Mitglied nicht sichtbar ist.

Übergeordnet

groupParticipants

Typ

Attribut

Erforderlich

Nein

Wert

Der Wert *participantName* ist der genaue Name einer Mitgliederdatei (ohne die .edml-Erweiterung), die als Teil einer Gruppe aufgeführt wird: Weitere Informationen finden Sie unter „name“ auf Seite 250.

```
<groupParticipants selectParticipant="redirectIfEmpty_link">
```

<groupParticipant>**Beschreibung**

Dieses Tag steht für die Einbeziehung eines einzelnen Mitglieds in die Gruppe.

Übergeordnet

groupParticipants

Typ

Tag

Erforderlich

Ja (mindestens eines)

<groupParticipant>-Attribute

Die folgenden Elemente sind gültige Attribute des groupParticipants-Tags.

name**Beschreibung**

Dieses Attribut benennt ein bestimmtes Mitglied, das der Gruppe angehört. Das name-Attribut des groupParticipant - Tags muss mit dem Dateinamen des Mitglieds (ohne die Dateinamenserweiterung .edml) übereinstimmen.

Übergeordnet

groupParticipant

Typ

Attribut

Erforderlich

Ja.

Wert

Der Wert ist der genaue Name einer Mitgliederdatei (ohne die Dateinamenserweiterung .edml):

```
<groupParticipant name="redirectIfEmpty_init">
```

Dieses Beispiel bezieht sich auf die EDML-Datei redirectIfEmpty_init.

partType**Beschreibung**

Dieses Attribut kennzeichnet den Typ des Mitglieds.

Übergeordnet

groupParticipant

Typ

Attribut

Erforderlich

Nein

Werte

identifier, member, option, multiple, data

- Der Wert *identifier* ist ein Mitglied, das die gesamte Gruppe kennzeichnet. Wird dieses Mitglied im Dokument gefunden, gilt die Gruppe als vorhanden, unabhängig davon, ob andere Gruppenmitglieder gefunden werden. Dies ist der Standardwert, wenn das Attribut `partType` nicht angegeben ist.
- Der Wert *member* ist ein normales Element einer Gruppe. Wenn nur dieses alleine gefunden wird, steht es nicht für eine Gruppe. Wenn es in einer Gruppe nicht gefunden wird, gilt die Gruppe als unvollständig.
- Der Wert *option* gibt an, dass das Mitglied optional ist. Wenn es nicht gefunden wird, gilt die Gruppe dennoch als vollständig (im Bedienfeld **Serververhalten** wird dann kein incomplete-Flag gesetzt).
- Der Wert *multiple* gibt an, dass das Mitglied optional ist und dem Serververhalten mehrere Kopien davon zugeordnet sein können. Nur für dieses Mitglied festgelegte Argumente werden beim Gruppieren von Mitgliedern nicht berücksichtigt, da sie möglicherweise abweichende Werte aufweisen.
- Der Wert *data* ist ein besonderes Mitglied, das von Programmierern als Speicher für zusätzliche Gruppendaten verwendet wird. Ansonsten wird dieses Mitglied ignoriert.

Mitglieder-EDML-Dateien

Folgende Tags und Attribute sind für Mitglieder-EDML-Dateien gültig:

<participant>**Beschreibung**

Dieses Tag enthält alle Spezifikationen für ein einzelnes Mitglied.

Übergeordnet

Keiner

Typ

Block-Tag

Erforderlich

Ja.

<participant>-Attribute

Die folgenden Elemente sind gültige Attribute des participant-Tags.

version**Beschreibung**

Dieses Attribut definiert, auf welche Version der Dreamweaver-Serververhaltensverarbeitung das aktuelle Serververhalten abzielt. Bei Dreamweaver CS3 lautet die Versionsnummer 9. Wenn keine Version angegeben ist, geht Dreamweaver von Version 7 aus. Bei dieser Version von Dreamweaver verfügen alle Gruppen und Mitglieder, die vom Serververhaltensgenerator erstellt werden, über das Versionsattribut 0.0.

Hinweis: Das Mitglieds-Versionsattribut überschreibt das Gruppen-Versionsattribut, wenn es sich von diesem unterscheidet. Die Mitgliedsdatei verwendet jedoch das Gruppen-Versionsattribut, wenn für das Mitglied kein Versionsattribut angegeben ist.

Dieses Attribut legt fest, ob bei Mitgliederdateien Blöcke zusammengeführt werden sollen. Bei Mitgliedern ohne dieses Attribut oder Mitgliedern, bei denen dieses Attribut auf den Wert 4 oder darunter gesetzt ist, werden die eingefügten Codeblöcke nicht mit anderen Blöcken auf der Seite zusammengeführt. Bei Mitgliedern, bei denen das Attribut auf Version 5 oder höher gesetzt ist, werden Codeblöcke mit anderen Blöcken der Seite zusammengeführt, soweit dies möglich ist. Beachten Sie, dass dieses Zusammenführen nur für Mitglieder vor oder nach dem HTML-Tag durchgeführt wird.

Übergeordnet

participant

Typ

Attribut

Erforderlich

Nein

<quickSearch>**Beschreibung**

Bei diesem Tag handelt es sich um einen einfachen Suchstring, der aus leistungsbedingten Gründen eingesetzt wird. Er darf kein regulärer Ausdruck sein. Wird der String im aktuellen Dokument gefunden, werden die verbleibenden Suchmuster aufgerufen, um bestimmte Instanzen zu suchen. Ist dieser String leer, werden immer die Suchmuster verwendet.

Übergeordnet

participant

Typ

Block-Tag

Erforderlich

Nein

Wert

Der Wert *searchString* ist ein expliziter String, der auf der Seite vorhanden ist, wenn das Mitglied existiert. Der String muss möglichst eindeutig sein, um die Leistung zu optimieren, er muss jedoch nicht auf jeden Fall eindeutig sein. Die Groß-/Kleinschreibung spielt dabei keine Rolle. Achten Sie aber auf Leerräume, die vom Benutzer geändert werden könnten:

```
<quickSearch>Response.Redirect</quickSearch>
```

Ist das Tag `quickSearch` leer, gilt dies als Übereinstimmung. Für die genauere Suche werden dann die in den `search-Pattern`-Tags definierten regulären Ausdrücke verwendet. Diese Methode ist sinnvoll, wenn zur richtigen Darstellung des Suchmusters ein einfacher String nicht ausreicht und reguläre Ausdrücke benötigt werden.

<insertText>

Beschreibung

Dieses Tag gibt an, welche Informationen an welcher Stelle in das Dokument einzufügen sind. Es enthält den einzufügenden Text. Benutzerdefinierte Teile des Texts müssen durch @@parameterName@@ gekennzeichnet werden.

Bei übersetzerspezifischen Mitgliedern und ähnlichen Fällen wird dieses Tag möglicherweise nicht benötigt.

Übergeordnet

implementation

Typ

Block-Tag

Erforderlich

Nein

Wert

Der Wert ist der in das Dokument einzufügende Text. Sollen Teile des Texts angepasst werden, können sie später über Parameter eingefügt werden. Argumente müssen zwischen zwei @-Zeichen gesetzt werden. Da dieser Text möglicherweise die EDML-Struktur stört, sollte dafür ein CDATA-Tag verwendet werden:

```
<insertText location="aboveHTML">
    <![CDATA[<%= @@recordset@@.cursorType %>]]>
</insertText>
```

Beim Einfügen des Texts wird das Argument @@recordset@@ durch einen vom Benutzer angegebenen Datensatzgruppennamen ersetzt. Weitere Informationen zu konditionellen und mehrfach eingesetzten Codeblöcken finden Sie unter „Benutzerdefinierte Serververhalten hinzufügen“ im Handbuch *Erste Schritte mit Dreamweaver*.

<insertText>-Attribute

Die folgenden Elemente sind gültige Attribute des insertText-Tags.

location

Beschreibung

Dieses Attribut gibt an, wo der Mitgliedertext eingefügt werden soll. Beachten Sie, dass die Einfügeposition vom Attribut whereToSearch des Tags searchPatterns abhängt. Beide Werte sind daher sorgfältig zu wählen (siehe „whereToSearch“ auf Seite 256).

Übergeordnet

insertText

Typ

Attribut

Erforderlich

Ja.

Werte

aboveHTML[+weight], belowHTML[+weight], beforeSelection, replaceSelection, wrapSelection, afterSelection, beforeNode, replaceNode, afterNode, firstChildOfNode, lastChildOfNode, nodeAttribute[+attribute]

- Der Wert *aboveHTML[+weight]* fügt den Text oberhalb des Tags `HTML` ein (nur für Servercode geeignet). Die Gewichtung kann eine Ganzzahl zwischen 1 und 99 sein. Mit ihr wird die relative Reihenfolge der verschiedenen Mitglieder festgelegt. Datensatzgruppen haben eine Gewichtung von 50. Verweist ein Mitglied auf Datensatzgruppenvariablen, benötigt es daher eine höhere Gewichtung, z. B. 60, damit der Code unterhalb der Datensatzgruppe eingefügt wird.

```
<insert location="aboveHTML+60">
```

Wurde keine Gewichtung angegeben, wird ihr intern der Wert 100 zugewiesen und die Datensatzgruppe unterhalb der bereits gewichteten Mitglieder eingefügt:

```
<insert location="aboveHTML">
```

- Der Wert *belowHTML[+weight]* entspricht der Position *aboveHTML*. Diese Mitglieder werden jedoch unterhalb des Schluss-Tags `/HTML` eingefügt.
- Der Wert *beforeSelection* fügt den Text vor der aktuellen Auswahl oder vor der Einfügemarke ein. Liegt keine Auswahl vor, wird der Text am Ende des Tags `body` eingefügt.
- *replaceSelection* ersetzt die aktuelle Auswahl durch den Text. Liegt keine Auswahl vor, wird der Text am Ende des Tags `body` eingefügt.
- *wrapSelection* berücksichtigt die aktuelle Auswahl, fügt den Block-Tag vor der Auswahl ein und fügt nach der Auswahl das entsprechende Schluss-Tag ein.
- *afterSelection* fügt den Text nach der aktuellen Auswahl oder nach der Einfügemarke ein. Liegt keine Auswahl vor, wird der Text am Ende des Tags `body` eingefügt.
- *beforeNode* fügt den Text vor einem Knoten ein (einer bestimmten Position im DOM). Wird die Funktion `dwscripts.applySB()` aufgerufen, um die Einfügung durchzuführen, muss der Zeiger auf den Knoten als Parameter von *paramObj* übergeben werden. Der benutzerdefinierbare Name dieses Parameters muss über das Attribut *nodeName* angegeben werden (siehe „nodeName“ auf Seite 255).

Zusammenfassend gilt: Enthält die Position das Wort *node*, müssen Sie das Tag *nodeName* deklarieren.

- *replaceNode* ersetzt einen Knoten durch den Text.
- *afterNode* fügt den Text nach einem Knoten ein.
- *firstChildOfNode* fügt den Text als erstes untergeordnetes Element eines Block-Tags ein. Hiermit können Sie beispielsweise Inhalte am Anfang eines `FORM`-Tags einfügen.
- *lastChildOfNode* fügt den Text als letztes untergeordnetes Element eines Block-Tags ein. Hiermit können Sie beispielsweise Inhalte am Ende eines `FORM`-Tags einfügen. Dies ist beim Hinzufügen von verborgenen Formularfeldern hilfreich.
- *nodeAttribute[+Attribut]* legt ein Attribut eines Tag-Knotens fest. Falls das Attribut nicht bereits existiert, wird es erstellt.

Mit `<insert location="nodeAttribute+ACTION" nodeName="form">` können Sie beispielsweise das `ACTION`-Attribut eines Formulars festlegen. Das `FORM`-Tag des Benutzers wird damit von `<form>` in `<form action="myText">` geändert.

Wenn Sie kein Attribut angeben, bewirkt die Position *nodeAttribute*, dass der Text direkt dem offenen Tag hinzugefügt wird. Mit `insert location="nodeAttribute"` können Sie beispielsweise ein optionales Attribut in ein Tag aufnehmen. Auf diese Weise können Sie das `INPUT`-Tag eines Benutzers ändern von `<input type="checkbox">` zu `<input type="checkbox" <%if (foo) Reponse.Write ("CHECKED") %>>`.

Hinweis: Beim Attributwert `location="nodeAttribute"` wird das letzte Suchmuster verwendet, um zu bestimmen, wo das Attribut beginnt und endet. Vergewissern Sie sich, dass das letzte Suchmuster die gesamte Anweisung findet.

nodeParamName**Beschreibung**

Dieses Attribut wird nur für knotenbezogene Einfügepositionen verwendet. Es bezeichnet den Parameter, mit dem der Knoten zum Einfügezeitpunkt übergeben wird.

Übergeordnet

insertText

Typ

Attribut

Erforderlich

Dieses Attribut ist nur erforderlich, wenn der Einfügeort das Wort `node` enthält.

Wert

Der Wert `tagtype_Tag` ist ein benutzerspezifischer Name für den mit dem Parameterobjekt an die Funktion `dwscripts.applySB()` übergebenen Knoten-Parameter. Angenommen, Sie verwenden zum Einfügen von Text in ein Formular einen Parameter mit dem Namen `form_tag`. In Ihrer Serververhaltensfunktion `applyServerBehavior()` können Sie dann mit dem Parameter `form_tag` das zu aktualisierende Formular genau bezeichnen:

```
function applyServerBehavior(ssRec) {
    var paramObj = new Object();
    paramObj.rs = getRecordsetName();
    paramObj.form_tag = getFormNode();
    dwscripts.applySB(paramObj, sbObj);
}
```

Sie können den Knoten-Parameter `form_tag` in Ihrer EDML-Datei festlegen, wie im folgenden Beispiel dargestellt:

```
<insertText location="lastChildOfNode" nodeParamName="form_tag">
    <![CDATA[<input type="hidden" name="MY_DATA">]]>
</insertText>
```

Der Text wird dann als Wert `lastChildOfNode` eingefügt und der ausgewählte Knoten mithilfe der `form_tag`-Eigenschaft des Parameterobjekts übergeben.

<searchPatterns>**Beschreibung**

Dieses Tag bietet Informationen zum Auffinden von Mitgliedertext im Dokument und enthält eine Liste von Suchmustern, die bei der Suche nach einem Mitglied eingesetzt werden. Sind mehrere Suchmuster definiert, müssen sie alle innerhalb des durchsuchten Texts vorkommen (die Suchmuster sind durch logisches AND verknüpft), es sei denn, sie sind über das Flag `isOptional` als optional gekennzeichnet.

Übergeordnet

implementation

Typ

Block-Tag

Erforderlich

Nein

<searchPatterns>-Attribute

Die folgenden Elemente sind gültige Attribute des searchPatterns-Tags.

whereToSearch

Beschreibung

Dieses Attribut gibt an, wo nach dem Mitgliedertext gesucht werden soll. Dieses Attribut hängt von der Einfügeposition ab. Beide Werte sind daher sorgfältig zu wählen (siehe „location“ auf Seite 253).

Übergeordnet

searchPatterns

Typ

Attribut

Erforderlich

Ja.

Werte

directive, tag+tagName, tag+, comment, text*

- Der Wert *directive* durchsucht alle Server-Direktiven (serverspezifische Tags). Bei ASP und JSP sind dies alle <% ... %>-Skriptblöcke.

Hinweis: Tag-Attribute werden nicht durchsucht, auch wenn sie Direktiven enthalten.

- Der Wert *tag+tagName* durchsucht ein bestimmtes Tag:

```
<searchPatterns whereToSearch="tag+FORM">
```

Dieses Beispiel durchsucht nur form-Tags. Standardmäßig wird der gesamte outerHTML-Knoten durchsucht. Den Typ von INPUT-Tags geben Sie nach dem Schrägstrich (/) an. Um beispielsweise alle Schaltflächen **Absenden** zu durchsuchen, geben Sie Folgendes ein:

```
<searchPatterns whereToSearch="tag+INPUT/SUBMIT">.
```

- Der Wert *tag+** durchsucht den Inhalt eines beliebigen Tags:

```
<searchPatterns whereToSearch="tag+*">
```

Dieses Beispiel durchsucht alle Tags.

- Der Wert *comment* durchsucht nur die HTML-Kommentare <!-- ... -->, wie im folgenden Beispiel dargestellt:

```
<searchPatterns whereToSearch="comment">
```

Dieses Beispiel durchsucht Tags wie <!-- my comment here -->.

- Der Wert *text* sucht nur in reinen Textabschnitten:

```
<searchPatterns whereToSearch="text">
  <searchPattern>XYZ</searchPattern>
</searchPatterns>
```

Dieses Beispiel findet einen Textknoten, der den Text XYZ enthält.

<searchPattern>

Beschreibung

Dieses Tag ist ein Muster, mit dem Mitgliedertext identifiziert und Parameterwerte daraus extrahiert werden. Jeder Parameter-Teilausdruck muss in Klammern gesetzt werden.

Die Muster können ohne Parameter (zur reinen Identifizierung von Mitgliedertext), mit einem oder mit mehreren Parametern vorliegen. Alle nicht optionalen Muster müssen vorhanden sein. Jeder Parameter muss benannt sein und darf nur genau einmal vorhanden sein.

Weitere Informationen zur Verwendung des `searchPattern`-Tags finden Sie unter „Serververhalten suchen“ auf Seite 268.

Übergeordnet

`searchPatterns`

Typ

Block-Tag

Erforderlich

Ja.

Werte

`searchString`, `/regularExpression/`, `<empty>`

- Der Wert `searchString` ist ein einfacher Suchstring, bei dem nach Groß-/Kleinschreibung unterschieden wird. Er kann nicht zum Extrahieren von Parametern verwendet werden.
- Der Wert `/regularExpression/` ist ein Suchmuster mit regulären Ausdrücken.
- Der Wert `<empty>` wird verwendet, wenn kein Muster angegeben ist. Es wird immer als Übereinstimmung ausgewertet, und der gesamte Wert wird dem ersten Parameter zugewiesen.

Um beispielsweise den Mitgliedertext `<%= RS1.Field.Items("author_id") %>` zu identifizieren, können Sie ein einfaches Muster definieren, gefolgt von einem genauen Muster, das auch die beiden Parameterwerte extrahiert:

```
<searchPattern>Field.Items</searchPattern>
<searchPattern paramNames="rs,col">
  <![CDATA[
    /<%= \s* (\w+) \.Field\.Items\ ( " (\w+) " ) \/
  ]]>
</searchPattern>
```

Dies ergibt eine genaue Übereinstimmung mit dem Muster und weist den Wert des ersten Teilausdrucks (`\w+`) dem Parameter `rs` und den zweiten Teilausdruck (`\w+`) dem Parameter `col`.

Hinweis: Es ist sehr wichtig, dass der reguläre Ausdruck mit einem Schrägstrich (`/`) beginnt und endet. Andernfalls gilt er als expliziter Suchstring. Regulären Ausdrücken kann der Parameter `"i"` nachgestellt werden, um anzugeben, dass die Groß-/Kleinschreibung beachtet werden soll (z. B. `/pattern/i`). VBScript berücksichtigt die Groß-/Kleinschreibung nicht; es sollte daher `/pattern/i` verwendet werden. JavaScript berücksichtigt die Groß-/Kleinschreibung; hier sollte daher `/pattern/` verwendet werden.

Gelegentlich soll der gesamte Inhalt der begrenzten Suchposition einem Parameter zugewiesen werden. In diesem Fall wird kein Muster angegeben:

```
<searchPatterns whereToSearch="tag+OPTION">
  <searchPattern>MY_OPTION_NAME</searchPattern>
  <searchPattern paramNames="optionLabel" limitSearch="innerOnly">
    </searchPattern>
</searchPatterns>
```

Bei diesem Beispiel wird dem Parameter `optionLabel` der gesamte Inhalt von `innerHTML` eines `OPTION`-Tags zugewiesen.

<searchPattern>-Attribute

Die folgenden Elemente sind gültige Attribute des `searchPatterns`-Tags.

paramNames

Beschreibung

Dieses Attribut ist eine Liste mit den durch Kommas getrennten Namen der Parameter, deren Werte extrahiert werden. Diese Parameter werden in der Reihenfolge des Teilausdrucks zugewiesen. Sie können wahlweise einzelne Parameter zuweisen oder mithilfe einer kommagetrennten Liste mehrere Parameter zuweisen. Werden andere parenthetische Ausdrücke verwendet, aber keine Parameter angegeben, können in der Liste der Parameternamen zusätzliche Kommas als Platzhalter eingesetzt werden.

Die Parameternamen sollten mit denen des Einfügetexts und der Aktualisierungsparameter übereinstimmen.

Übergeordnet

`searchPattern`

Typ

Attribut

Erforderlich

Ja.

Werte

paramName1, paramName2, ...

Jeder Parametername muss genau mit dem Namen eines Parameters im Einfügetext übereinstimmen. Enthält der Einfügetext beispielsweise `@p1@@`, müssen Sie genau einen Parameter mit diesem Namen definieren:

```
<searchPattern paramNames="p1">patterns</searchPattern>
```

Um mehrere Parameter mit einem einzelnen Muster zu extrahieren, verwenden Sie eine Liste der durch Kommas getrennten Parameternamen in der Reihenfolge der Teilausdrücke im Muster. Nehmen wir an, Sie verwenden folgendes Suchmuster:

```
<searchPattern paramName="p1,,p2">/(\w+)_ (BIG|SMALL)_ (\w+)/</searchPattern>
```

Es gibt zwei Parameter zu extrahieren, zwischen denen Text steht. Bei folgendem Text: `<%= a_BIG_b %>` entspricht der erste Teilausdruck im Suchmuster `a`, daraus ergibt sich `p1="a"`. Der zweite Teilausdruck wird ignoriert (beachten Sie das `,` im Wert `paramName`). Der dritte Teilausdruck stimmt überein mit `b`, daraus ergibt sich `p2="b"`.

limitSearch

Beschreibung

Dieses Attribut begrenzt die Suche auf einen Teil des Tags `whereToSearch`.

Übergeordnet

`searchPattern`

Typ

Attribut

Erforderlich

Nein

Werte

all, attribute+attribName, tagOnly, innerOnly

- Der Wert *all* (Standard) durchsucht das gesamte im Attribut *whereToSearch* angegebene Tag.
- Der Wert *attribute+attribName* durchsucht nur innerhalb des Wertes des angegebenen Attributs:

```
<searchPatterns whereToSearch="tag+FORM">
  <searchPattern limitSearch="attribute+ACTION">
    /MY_PATTERN/
  </searchPattern>
</searchPatterns>
```

Dieses Beispiel gibt an, dass nur der Wert des Attributs *ACTION* von *FORM*-Tags durchsucht werden soll. Ist dieses Attribut nicht definiert, wird das Tag ignoriert.

- Der Wert *tagOnly* durchsucht nur das äußere Tag und ignoriert das Tag *innerHTML*. Dieser Wert ist nur zulässig, wenn es sich bei *whereToSearch* um ein Tag handelt.
- Der Wert *innerOnly* durchsucht nur das Tag *innerHTML* und ignoriert das äußere Tag. Dieser Wert ist nur zulässig, wenn es sich bei *whereToSearch* um ein Tag handelt.

isOptional

Beschreibung

Dieses Attribut zeigt an, dass das Suchmuster zum Auffinden des Mitglieds nicht erforderlich ist. Es kann bei komplexen Mitgliedern verwendet werden, bei denen ggf. unkritische Parameter zu extrahieren sind. Sie können zunächst Muster erstellen, mit denen ein Mitglied eindeutig identifiziert wird, und anschließend mit optionalen Mustern die unkritischen Parameter extrahieren.

Übergeordnet

searchPattern

Typ

Attribut

Erforderlich

Nein

Werte

true, false

- Der Wert lautet *true*, wenn *searchPattern* zur Identifizierung des Mitglieds nicht erforderlich ist.
- Der Wert lautet *false* (Standard), wenn das Tag *searchPattern* erforderlich ist.

Betrachten Sie beispielsweise folgenden einfachen Datensatzgruppen-String:

```
<%
var Recordset1 = Server.CreateObject("ADODB.Recordset");
Recordset1.ActiveConnection = "dsn=andescOFFEE";
Recordset1.Source = "SELECT * FROM PressReleases";
Recordset1.CursorType = 3;
Recordset1.Open();
%>
```

Die Suchmuster müssen das Mitglied identifizieren und diverse Parameter extrahieren. Auch wenn der Parameter *cursorType* nicht gefunden wird, sollte bei diesem Muster dennoch von einer Datensatzgruppe ausgegangen werden. Der Parameter *cursor* ist optional. In EDML könnten Suchmuster beispielsweise so aussehen:

```
<searchPattern paramNames="rs">/var (\w+) = Server.CreateObject/
</searchPattern>
<searchPattern paramNames="src">/ActiveConnection = "([^\r\n]*)"/</searchPattern>
<searchPattern paramNames="conn">/Source = "([^\r\n]*)"/</searchPattern>
<searchPattern paramNames="cursor" isOptional="true">/CursorType = (\d+)/
</searchPattern>
```

Die ersten drei Muster werden benötigt, um die Datensatzgruppe zu identifizieren. Auch wenn der letzte Parameter nicht gefunden wird, ist die Datensatzgruppe dennoch identifiziert.

<updatePatterns>

Beschreibung

Mithilfe dieser optionalen erweiterten Funktion können Sie das Mitglied genau aktualisieren. Ohne dieses Tag wird das Mitglied automatisch aktualisiert, indem jedes Mal der gesamte Mitgliedtext ersetzt wird. Wenn Sie ein `updatePatterns`-Tag angeben, muss es bestimmte Muster enthalten, um im Mitglied die einzelnen Parameter zu finden und zu ersetzen.

Dieses Tag ist hilfreich, wenn ein Benutzer den Mitgliedtext bearbeitet. Die präzisen Aktualisierungen erfolgen nur bei den zu ändernden Textabschnitten.

Übergeordnet

implementation

Typ

Block-Tag

Erforderlich

Nein

<updatePattern>

Beschreibung

Dieses Tag ist ein bestimmter Typ eines regulären Ausdrucks und ermöglicht Ihnen die genaue Aktualisierung des Mitgliedertexts. Für jeden eindeutigen im Einfügetext deklarierten Parameter muss mindestens eine Definition eines Aktualisierungsmusters vorliegen (im Format @@paramName@@).

Übergeordnet

updatePatterns

Typ

Block-Tag

Erforderlich

Ja (mindestens eines, wenn das Tag `updatePatterns` deklariert ist)

Werte

Ein regulärer Ausdruck zur Suche nach Parametern, die zwischen zwei parenthetische Teilausdrücke gesetzt sind, im Format `/(pre-pattern) Parameter-pattern (post-pattern)/`. Für jedes im Einfügetext definierte eindeutige `@@paramName@@` wird mindestens ein Aktualisierungsmuster benötigt. Nachfolgend ein Beispiel für einen Einfügetext:

```
<insertText location="afterSelection">
    <![CDATA[<%= @@rs@@.Field.Items("@@col@@") %>]]>
</insertText>
```

Eine bestimmte Instanz des Einfügetexts auf der Seite kann wie folgt aussehen:

```
<%= RS1.Field.Items("author_id") %>
```

Es liegen die beiden Parameter `rs` und `col` vor. Um diesen Text nach dem Einfügen auf der Seite zu aktualisieren, benötigen Sie zwei Definitionen für Aktualisierungsmuster:

```
<updatePattern paramName="rs" >
    /(\b)\w+(\.Field\.Items)/
</updatePattern>
<updatePattern paramName="col">
    /(\bItems\)("\)\w+(\.))/
</updatePattern>
```

Bei Klammern, ebenso wie bei den anderen, für reguläre Ausdrücke verwendeten speziellen Zeichen, wird durch Voranstellen eines umgekehrten Schrägstrichs die besondere Funktion aufgehoben. Der mittlere Ausdruck, hier als `\w+` definiert, wird mit dem zuletzt für die Parameter `rs` bzw. `col` übergebenen Wert aktualisiert. Die Werte `RS1` und `author_id` können mit neuen Werten präzise aktualisiert werden.

Um mehrere Vorkommen desselben Musters gleichzeitig zu aktualisieren, kann nach dem schließenden Schrägstrich des regulären Ausdrucks das globale Flag `g` gesetzt werden, z. B. `/pattern/g`.

Bei langen und komplexen Mitgliedertexten benötigen Sie möglicherweise mehrere Muster, um einen einzelnen Parameter zu aktualisieren:

```
<% ...
    Recordset1.CursorType = 0;
    Recordset1.CursorLocation = 2;
    Recordset1.LockType = 3;
%>
```

Um den Namen der Datensatzgruppe an allen drei Positionen zu aktualisieren, benötigen Sie drei Aktualisierungsmuster für einen einzelnen Parameter:

```
<updatePattern paramName="rs">
    /(\b)\w+(\.CursorType)/
</updatePattern>
<updatePattern paramName="rs">
    /(\b)\w+(\.CursorLocation)/
</updatePattern>
<updatePattern paramName="rs">
    /(\b)\w+(\.LockType)/
</updatePattern>
```

Jetzt können Sie einen neuen Wert für die Datensatzgruppe übergeben. Dieser wird dann an drei Positionen präzise aktualisiert.

<updatePattern>-Attribute

Die folgenden Elemente sind gültige Attribute des `updatePattern`-Tags.

paramName

Beschreibung

Dieses Attribut gibt den Namen des Parameters an, dessen Wert zur Aktualisierung des Mitglieds verwendet wird. Dieser Parameter sollte mit den Einfügetext- und Suchparametern übereinstimmen.

Übergeordnet

`updatePattern`

Typ

Attribut

Erforderlich

Ja.

Werte

Der Wert ist der genaue Name eines Parameters, der im Einfügetext verwendet wird. Enthält der Einfügetext beispielsweise den Wert `@@rs@@`, muss der Name des Parameters wie folgt lauten:

```
<updatePattern paramName="rs">pattern</updatePattern>
```

<delete>

Beschreibung

Dieses Tag ist eine optionale erweiterte Funktion, die es Ihnen ermöglicht, das Löschen eines Mitglieds zu steuern. Ohne dieses Tag wird das Mitglied vollständig gelöscht (allerdings nur, wenn kein Serververhalten darauf verweist). Über ein `delete`-Tag können Sie festlegen, dass es nie gelöscht werden soll oder dass nur Teile davon gelöscht werden.

Übergeordnet

`implementation`

Typ

Tag

Erforderlich

Nein

<delete>-Attribute

Die folgenden Elemente sind gültige Attribute des `delete`-Tags.

deleteType

Beschreibung

Dieses Attribut gibt die Art der auszuführenden Löschung an. Die genaue Bedeutung hängt davon ab, ob das Mitglied eine Direktive, ein Tag oder ein Attribut ist. Standardmäßig wird das gesamte Mitglied gelöscht.

Übergeordnet

`delete`

Typ

Attribut

Erforderlich

Nein

Werte*all, none, tagOnly, innerOnly, attribute+attribName, attribute+**

- *all* (Standard) bewirkt, dass die gesamte Direktive bzw. das gesamte Tag gelöscht wird. Bei Attributen wird die gesamte Definition gelöscht.
- *none* bewirkt, dass dieses Mitglied nie automatisch gelöscht wird.
- *tagOnly* bewirkt, dass nur das äußere Tag entfernt, der Inhalt des Tags `innerHTML` aber beibehalten wird. Bei Attributen wird auch das äußere Tag entfernt, soweit es sich um ein Block-Tag handelt. Bei Direktiven ohne Bedeutung.
- *innerOnly* entfernt bei Anwendung auf Tags nur den Inhalt (`innerHTML`). Bei Attributen wird nur der Wert entfernt. Bei Direktiven ohne Bedeutung.
- *attribute+attribName* entfernt bei Anwendung auf Tags nur das angegebene Attribut. Bei Direktiven und Attributen ohne Bedeutung.
- Der Wert *attribute+** entfernt alle Attribute von Tags. Bei Direktiven und Attributen ohne Bedeutung.

Wenn das Serververhalten beispielsweise ausgewählten Text in einen Hyperlink konvertiert, können Sie den Hyperlink entfernen, wenn Sie die Löschung nur durch Entfernen des äußeren Tags durchführen:

```
<delete deleteType="tagOnly"/>
```

Hierdurch wird ein Hyperlink-Mitglied von `<A HREF="...">HELLO</A>` zu `HELLO` geändert.

<translator>**Beschreibung**

Dieses Tag liefert Informationen zum Übersetzen eines Mitglieds, um es ggf. anders anzuzeigen und mit einem benutzerdefinierten Eigenschafteninspektor zu versehen.

Übergeordnet*implementation***Typ**

Block-Tag

Erforderlich

Nein

<searchPatterns>**Beschreibung**

Dieses Tag bewirkt, dass Dreamweaver jede angegebene Instanz in einem Dokument findet. Sind mehrere Suchmuster definiert, müssen sie alle innerhalb des durchsuchten Texts vorkommen (die Suchmuster sind durch logisches AND verknüpft), es sei denn, sie sind über das Flag `isOptional` als optional gekennzeichnet.

Übergeordnet*translator***Typ**

Block-Tag

Erforderlich

Ja.

<translations>

Beschreibung

Dieses Tag enthält eine Liste von Übersetzungsanweisungen, von denen jede angibt, wo nach dem Mitglied gesucht werden und wie damit verfahren werden soll.

Übergeordnet

translator

Typ

Block-Tag

Erforderlich

Nein

<translation>

Beschreibung

Dieses Tag enthält eine einzige Übersetzungsanweisung, die die Position für das Mitglied, die Art der auszuführenden von Übersetzung und den Inhalt angibt, durch den der Mitgliedertext ersetzt werden soll.

Übergeordnet

translations

Typ

Block-Tag

Erforderlich

Nein

<translation>-Attribute

Die folgenden Elemente sind gültige Attribute des translation-Tags.

whereToSearch

Beschreibung

Dieses Attribut gibt an, wo nach dem Text gesucht werden soll, was der Einfügeposition entspricht. Beide Werte sind daher sorgfältig zu wählen (siehe „location“ auf Seite 253).

Übergeordnet

translation

Typ

Attribut

Erforderlich

Ja.

limitSearch**Beschreibung**

Dieses Attribut begrenzt die Suche auf einen Teil des Tags `whereToSearch`.

Übergeordnet

`translation`

Typ

Attribut

Erforderlich

Nein

translationType**Beschreibung**

Dieses Attribut gibt den Typ der auszuführenden Übersetzung an. Diese Typen sind voreingestellt und legen eine bestimmte Funktionalität für die Übersetzung fest. Wenn Sie beispielsweise `dynamic data` angeben, müssen zu übersetzende Daten sich verhalten wie dynamische Daten von Dreamweaver, d. h., sie sollten in der Entwurfsansicht im Bedienfeld **Serververhalten** mit Platzhaltern für dynamische Daten (geschweifte Klammern `{}`) mit dynamischer Hintergrundfarbe) angezeigt werden.

Übergeordnet

`translation`

Typ

Attribut

Erforderlich

Ja.

Werte

dynamic data, dynamic image, dynamic source, tabbed region start, tabbed region end, custom

- *dynamic data* zeigt an, dass die übersetzten Direktiven im Aussehen und Verhalten den dynamischen Daten von Dreamweaver entsprechen:

```
<translation whereToSearch="tag+IMAGE"
    limitSearch="attribute+SRC"
    translationType="dynamic data">
```

- Der Wert *dynamic image* zeigt an, dass die übersetzten Attribute in Aussehen und Verhalten den dynamischen Bildern von Dreamweaver entsprechen:

```
<translation whereToSearch="IMAGE+SRC"
    translationType="dynamic image">
```

- *dynamic source* zeigt an, dass die übersetzten Direktiven im Verhalten den dynamischen Quellen von Dreamweaver entsprechen:

```
<translation whereToSearch="directive"
    translationType="dynamic source">
```

- *tabbed region start* zeigt an, dass die übersetzten `<CFLOOP>`-Tags den Anfang einer Tabellenkontur definieren:

```
<translation whereToSearch="CFLOOP"
    translationType="tabbed region start">
```

- *tabbed region end* zeigt an, dass die übersetzten `</CFLOOP>`-Tags das Ende einer Tabellenkontur definieren:

```
<translation whereToSearch="CFLOOP"
    translationType="tabbed region end">
```

- *custom* ist der Standardfall, bei dem der Übersetzung keine interne Dreamweaver-Funktionalität hinzugefügt wird. Es wird oft beim Festlegen eines Tags zum Einfügen eines benutzerdefinierten Eigenschafteninspektors verwendet.

```
<translation whereToSearch="directive"  
    translationType="custom">
```

<openTag>

Beschreibung

Ein optionales Tag, das zu Anfang des Übersetzungsabschnitts eingefügt werden kann. Hierdurch können bestimmte andere Erweiterungen, wie z. B. benutzerdefinierte Eigenschafteninspektoren, die Übersetzung finden.

Übergeordnet

translation

Typ

Block-Tag

Erforderlich

Nein

Werte

tagName ist ein gültiger Tag-Name. Er sollte möglichst eindeutig sein, um Konflikte mit bekannten Tag-Typen zu vermeiden. Wenn Sie beispielsweise `<openTag>MM_DYNAMIC_CONTENT</openTag>` festlegen, werden die dynamischen Daten in das Tag `MM_DYNAMIC_CONTENT` übersetzt.

<Attribute>

Beschreibung

Dieses Tag enthält eine Liste mit Attributen, die dem übersetzten Tag hinzuzufügen sind, welches durch `openTag` bestimmt wird. Ist hingegen `openTag` nicht definiert und `tag` durch `searchPattern` angegeben, enthält dieses Tag eine Liste der übersetzten Attribute, die dem gefundenen Tag hinzuzufügen sind.

Übergeordnet

translation

Typ

Block-Tag

Erforderlich

Nein

<Attribut>

Beschreibung

Dieses Tag gibt ein einzelnes Attribut (oder ein übersetztes Attribut) an, das dem übersetzten Tag hinzuzufügen ist.

Übergeordnet

Attribute

Typ

Block-Tag

Erforderlich

Ja (mindestens eines)

Werte

attributeName="attributeValue" ist ein auf einen Wert gesetztes Attribut. In der Regel ist der Attributname festgelegt, und der Wert enthält einige Parameterreferenzen, die durch die Parametermuster extrahiert werden:

```
<attribute>SOURCE="@rs@"</attribute>
<attribute>BINDING="@col@"</attribute>
```

oder

```
<Attribut>
    mmTranslatedValueDynValue="VALUE={@rs@.@col@}"
</attribute>
```

<display>**Beschreibung**

Dieses Tag ist ein optionaler Anzeigestring, der in die Übersetzung eingefügt werden soll.

Übergeordnet

translation

Typ

Block-Tag

Erforderlich

Nein

Werte

displayString ist ein beliebiger String, der Text und HTML enthalten kann. Er kann Parameterreferenzen enthalten, die durch Parametermuster extrahiert werden. Beispielsweise wird mit `<display>{@rs@.@col@}</display>` die Übersetzung als `{myRecordset.myCol}` wiedergegeben.

<closeTag>**Beschreibung**

Ein optionales Tag, das am Ende des übersetzten Abschnitts eingefügt werden soll. Hierdurch können bestimmte andere Erweiterungen, wie z. B. benutzerdefinierte Eigenschafteninspektoren, die Übersetzung finden.

Übergeordnet

translation

Typ

Block-Tag

Erforderlich

Nein

Werte

tagName ist ein gültiger Tag-Name, der mit einem *openTag* der Übersetzung übereinstimmen sollte.

Beispiel

Wenn Sie den Wert `<closeTag>MM_DYNAMIC_CONTENT</closeTag>` angeben, endet die Übersetzung der dynamischen Daten mit dem Tag `</MM_DYNAMIC_CONTENT>`.

Serververhaltenstechniken

Dieser Abschnitt behandelt die allgemeinen und erweiterten Techniken zum Erstellen und Bearbeiten von Serververhalten. Die meisten Verfahren setzen bestimmte Einstellungen in den EDML-Dateien voraus.

Serververhalten suchen

Suchmuster schreiben Um Serververhalten aktualisieren oder löschen zu können, muss Dreamweaver jede Instanz in einem Dokument finden können. Hierzu ist ein `quickSearch`-Tag und mindestens ein `searchPattern`-Tag (innerhalb des `searchPatterns`-Tags) erforderlich.

Das `quickSearch`-Tag sollte ein String (kein regulärer Ausdruck) sein und darauf hinweisen, dass das Serververhalten möglicherweise auf der Seite vorhanden ist. Die Groß-/Kleinschreibung spielt hierfür keine Rolle. Es sollte kurz und eindeutig sein und keine Leerräume oder sonstigen Abschnitte enthalten, die vom Benutzer geändert werden könnten. Nachfolgend ein Beispiel für ein Mitglied, das aus einem einfachen ASP JavaScript-Tag besteht:

```
<% if (Recordset1.EOF) Response.Redirect("some_url_here") %>
```

Im folgenden Beispiel sucht der String `quickSearch` nach dem Tag:

```
<quickSearch>Response.Redirect</quickSearch>
```

Aus leistungsbedingten Gründen bildet das `quickSearch`-Muster den Anfang des Prozesses zum Auffinden von Serververhaltensinstanzen. Wenn dieser String im Dokument gefunden wird und das Mitglied ein Serververhalten identifiziert (in der Gruppendatei `partType="identifier"` für dieses Mitglied), werden die entsprechenden Serververhaltensdateien geladen, und `findServerBehaviors()` wird aufgerufen. Wenn das Mitglied keine eindeutigen Strings enthält, nach denen gesucht werden kann (oder anhand derer eine Fehlerbereinigung durchgeführt werden kann), können Sie den `quickSearch`-String leer lassen:

```
<quickSearch></quickSearch>
```

In diesem Beispiel wird das Serververhalten immer geladen und kann das Dokument durchsuchen.

Dann durchsucht das `searchPattern`-Tag das Dokument genauer als das Tag `quickSearch` und extrahiert Parameterwerte aus dem Mitgliedscode. In den Suchmustern (Attribut `whereToSearch`) ist festgelegt, an welcher Stelle mit einer Reihe von `searchPattern`-Tags, die spezifische Muster enthalten, gesucht werden soll. Diese Muster können einfache Strings oder reguläre Ausdrücke verwenden. Der oben gezeigte Beispielcode ist eine ASP-Direktive. Somit dienen die Spezifikation `whereToSearch="directive"` und ein regulärer Ausdruck zum Identifizieren der Direktive und zum Extrahieren der Parameter, wie in folgendem Beispiel:

```
<quickSearch>Response.Write</quickSearch>
<searchPatterns whereToSearch="directive">
  <searchPattern paramNames="rs,new_url">
    /if\s*\((\w+)\.EOF\)\s*Response\.Redirect\("([^\r\n]*)")/i
  </searchPattern>
</searchPatterns>
```

Der Suchstring ist durch den Schrägstrich (/) am Anfang und am Ende als regulärer Ausdruck definiert. Durch das darauf folgende `i` wird angegeben, dass nicht zwischen Groß- und Kleinschreibung unterschieden werden soll. Innerhalb des regulären Ausdrucks werden Sonderzeichen wie Klammern () und Punkte (.) durch einen vorausgehenden umgekehrten Schrägstrich (\) ausgeschaltet. Die zwei Parameter `rs` und `new_url` werden durch parenthetische Teilausdrücke aus dem String in Klammern extrahiert (die Ausdrücke müssen in Klammern eingeschlossen sein). In diesem Beispiel werden Sie durch `(\w+)` und `([^\r\n]*)` angegeben: Diese Werte entsprechen Werten regulärer Ausdrücke, die normalerweise von §1 und §2 zurückgegeben werden.

Optionale Suchmuster In manchen Fällen möchten Sie Mitglieder identifizieren, obwohl einige Parameter nicht gefunden werden. Es könnte ein Mitglied vorhanden sein, das einige optionale Informationen wie etwa eine Telefonnummer speichert. Sie könnten dann folgenden ASP-Code einsetzen:

```
<% //Adressblock
    LNAME = "joe";
    FNAME = "smith";
    PHONE = "123-4567";
%>
```

Sie könnten folgende Suchmuster verwenden:

```
<quickSearch>address</quickSearch>
<searchPatterns whereToSearch="directive">
    <searchPattern paramNames="lname">/LNAME\s*=\s*"([^\r\n]*)"/i</searchPattern>
    <searchPattern paramNames="fname">/FNAME\s*=\s*"([^\r\n]*)"/i</searchPattern>
    <searchPattern paramNames="phone">/PHONE\s*=\s*"([^\r\n]*)"/i</searchPattern>
</searchPatterns>
```

Im vorhergehenden Beispiel musste die Telefonnummer angegeben werden. Durch Aufnehmen des Attributs `isOptional` können Sie das Eingeben der Telefonnummer als optional festlegen:

```
<quickSearch>address</quickSearch>
<searchPatterns whereToSearch="directive">
    <searchPattern paramNames="lname">/LNAME\s*=\s*"([^\r\n]*)"/i</searchPattern>
    <searchPattern paramNames="fname">/FNAME\s*=\s*"([^\r\n]*)"/i</searchPattern>
    <searchPattern paramNames="phone" isOptional="true">/PHONE\s*=\s*"([^\r\n]*)"/i</searchPattern>
</searchPatterns>
```

Nun wird das Mitglied auch dann erkannt, wenn die Telefonnummer nicht gefunden wurde.

Zusammengehörende Mitglieder erkennen Hat ein Serververhalten mehrere Mitglieder, müssen diese Mitglieder im Dokument des Benutzers identifiziert und einander zugeordnet werden. Wenn der Benutzer mehrere Instanzen des Serververhaltens auf ein Dokument anwendet, muss somit jede Mitgliedergruppe erkannt werden. Damit Mitglieder richtig erkannt werden, müssen Sie eventuell einige Parameter ändern oder hinzufügen und die Mitglieder so konstruieren, dass sie sich eindeutig identifizieren lassen.

Beim Zuordnen müssen einige Regeln befolgt werden. Mitglieder werden dann zugeordnet, wenn alle gleichnamigen Parameter denselben Wert haben. Ober- und unterhalb des `html`-Tags darf nur eine Instanz eines Mitglieds mit einer bestimmten Gruppe von Parameterwerten vorhanden sein. Innerhalb der `html.../html`-Tags werden Mitglieder auch anhand ihrer relativen Position in Bezug auf die Auswahl oder auf gemeinsame, für Einfügungen verwendete Knoten erkannt.

Mitglieder ohne Parameter werden automatisch zugeordnet, wie in diesem Beispiel eines Serververhaltens mit einer Gruppendatei:

```
<group serverBehavior="test.htm">
    <title>Test</title>
    <groupParticipants>
        <groupParticipant name="test_p1" partType="identifizier" />
        <groupParticipant name="test_p2" partType="identifizier" />
    </groupParticipants>
</group>
```

Bei diesem Beispiel werden zwei einfache Mitglieder oberhalb des Tags `html` eingefügt:

```
<% //test_p1 %>
<% //test_p2 %>
<html>
```

Diese Mitglieder werden gefunden und zugeordnet, und *Test* wird einmal im Bedienfeld **Serververhalten** angezeigt. Wenn Sie das Serververhalten erneut hinzufügen, wird nichts hinzugefügt, weil das Mitglied bereits existiert.

Wenn die Mitglieder eindeutige Parameter haben, können mehrere Instanzen oberhalb des `html`-Tags eingefügt werden. Indem ein Benutzer dem Mitglied beispielsweise einen Namensparameter hinzufügt, kann er einen eindeutigen Namen im Serververhalten-Dialogfeld **Test** eingeben. Gibt der Benutzer den Namen "aaa" ein, werden folgende Mitglieder hinzugefügt:

```
<% //test_p1 name="aaa" %>
<% //test_p2 name="aaa" %>
<html>
```

Wenn Sie das Serververhalten erneut mit einem anderen Namen hinzufügen, z. B. **bbb**, sieht das Dokument folgendermaßen aus:

```
<% //test_p1 name="aaa" %>
<% //test_p2 name="aaa" %>
<% //test_p1 name="bbb" %>
<% //test_p2 name="bbb" %>
<html>
```

In diesem Fall werden zwei Instanzen von „Test“ im Bedienfeld **Serververhalten** aufgeführt. Wenn der Benutzer versucht, der Seite eine dritte Instanz mit dem Namen **aaa** hinzuzufügen, wird nichts hinzugefügt, da diese Instanz bereits existiert.

Innerhalb des `html`-Tags werden zum Erkennen eventuell auch Positionsinformationen verwendet. Im nachfolgenden Beispiel wird je ein Mitglied vor und nach der Auswahl hinzugefügt:

```
<% if (expression) { //mySBName %>
```

hier eine zufällige HTML-Auswahl:

```
<% } //Ende von mySBName %>
```

Diese beiden Mitglieder haben keine Parameter und werden daher zu einer Gruppe zusammengefasst. Sie können jedoch an einer anderen Stelle im HTML-Bereich eine andere Instanz dieses Serververhaltens hinzufügen:

```
<% if (expression) { //mySBName %>
```

hier eine zufällige HTML-Auswahl:

```
<% } //Ende von mySBName %>
```

Hier weiterer HTML-Code:

```
<% if (expression) { //mySBName %>
```

Hier eine weitere HTML-Auswahl:

```
<% } //Ende von mySBName %>
```

Nun sind zwei identische Instanzen der einzelnen Mitglieder vorhanden (was innerhalb des HTML-Bereichs zulässig ist). Die Zuordnung der Instanzen erfolgt in der Reihenfolge, in der sie im Dokument angezeigt werden.

Nachfolgend ein Beispiel für ein Zuordnungsproblem und Wege, es zu vermeiden. Angenommen, Sie erstellen ein Mitglied, das die Steuer anhand einiger dynamischer Daten errechnet und das Ergebnis bei der aktuellen Auswahl anzeigt.

```
<% total = Recordset1.Fields.Item("itemPrice").Value * 1.0825 %>
<html>
<body>
    Die Gesamtsumme (inkl. Steuer) beträgt $<%=total%>
</body>
</html>
```

Die beiden Mitglieder werden zugeordnet, weil sie keine gemeinsamen Parameter haben. Wenn Sie jedoch eine zweite Instanz dieses Serververhaltens hinzufügen, erhalten Sie folgenden Code:

```
<% total = Recordset1.Fields.Item("itemPrice").Value * 1.0825 %>
<% total = Recordset1.Fields.Item("salePrice").Value * 1.0825 %>
<html>
<body>
    Die Gesamtsumme (inkl. Steuer) beträgt $<%=total%>
    Der Verkaufspreis (inkl. Steuer) beträgt $<%=total%>
</body>
</html>
```

Dieses Serververhalten funktioniert nicht mehr korrekt, weil nur ein Parameter mit dem Namen `total` vorhanden ist. Sie können dieses Problem umgehen, indem Sie sicherstellen, dass es einen Parameter mit einem eindeutigen Wert gibt, mit dem Mitglieder zugeordnet werden können. Im nachfolgenden Beispiel können Sie den Variablennamen `total` mit einem Spaltennamen eindeutig kennzeichnen:

```
<% itemPrice_total = Recordset1.Fields.Item("itemPrice").Value * 1.0825 %>
<% salePrice_total = Recordset1.Fields.Item("salePrice").Value * 1.0825 %>
<html>
<body>
    Die Gesamtsumme (inkl. Steuer) beträgt $<%=itemPrice_total%>
    Der Verkaufspreis (inkl. Steuer) beträgt $<%=salePrice_total%>
</body>
</html>
```

Die Suchmuster können die Mitglieder nun eindeutig erkennen und zuordnen.

Auflösung von Suchmustern

Dreamweaver unterstützt folgende Aktionen durch Verwenden der Mitgliederfunktionalität `searchPatterns`:

- Abhängigkeiten bei der Dateiübertragung
- Aktualisieren der Dateipfade für alle Dateireferenzen (wie beispielsweise für Include-Dateien)

Wenn Dreamweaver Servermodelle anlegt, stellt es Listen mit Mustern zusammen, indem es die Mitglieder nach besonderen `paramNames` durchsucht. Dreamweaver verwendet alle `searchPattern`-Tags, in dem das `paramNames`-Attribut mit `_url` endet, zum Überprüfen von URLs und zum Berichtigen des Pfadnamens. In einem einzigen `searchPattern`-Tag können mehrere URLs verwendet werden.

Für jedes Übersetzer-`searchPattern`-Tag mit einem `paramNames`-Attributwert, der auf `_includeUrl` endet, verwendet Dreamweaver dieses `searchPattern`, um Include-Datei-Anweisungen auf der Seite zu übersetzen. Dreamweaver identifiziert Include-Datei-URLs mithilfe eines anderen Suffix-Strings, da nicht alle URL-Referenzen übersetzt werden. Hinzu kommt, dass nur eine einzelne URL als Include-Datei übersetzt werden kann.

Beim Auflösen eines `searchPatterns`-Tags verwendet Dreamweaver folgenden Algorithmus:

- 1 Suchen nach dem `whereToSearch`-Attribut innerhalb des `searchPatterns`-Tags.
- 2 Beginnt der Attributwert mit `tag+`, wird angenommen, dass der Rest des Strings der Tag-Name ist. Dieser Tag-Name darf keine Leerzeichen enthalten.
- 3 Suche nach dem `limitSearch`-Attribut innerhalb des `searchPattern`-Tags.
- 4 Beginnt der Attributwert mit `attribute+`, wird angenommen, dass der Rest des Strings der Attributname ist. Dieser Attributname darf keine Leerzeichen enthalten.

Wenn diese vier Schritte erfolgreich durchgeführt werden, geht Dreamweaver von einer Tag-Attribut-Kombination aus. Anderenfalls sucht Dreamweaver nach `searchPattern`-Tags mit einem `paramName`-Attribut, das über das Suffix `_url` verfügt, sowie nach einem definierten Ausdruck. (Weitere Informationen zu regulären Ausdrücken finden Sie unter „Reguläre Ausdrücke“ auf Seite 244.)

Das `searchPatterns`-Tag im folgenden Beispiel verfügt nicht über ein Suchmuster, da es ein Tag (`cfinclude`) mit einem Attribut (`template`) kombiniert, um so die URL zum Überprüfen der Dateiabhängigkeiten, Korrigieren des Pfads etc. zu isolieren:

```
<searchPatterns whereToSearch="tag+cfinclude">
    <searchPattern paramNames="include_url" limitSearch="attribute+template" />
</searchPatterns>
```

Die erwähnte Tag-Attributkombination kann nicht auf Übersetzungen angewendet werden, weil Dreamweaver in der JavaScript-Ebene in reinen Text übersetzt. Funktionen wie das Überprüfen der Dateiabhängigkeiten und das Korrigieren des Pfads hingegen finden immer auf der C-Ebene statt. Auf der C-Ebene teilt Dreamweaver das Dokument in Direktiven (reinen Text) und Tags (in einer effizienten Strukturansicht) auf.

Serververhalten aktualisieren

Ersatzaktualisierung Standardmäßig haben EDML-Mitgliederdateien kein `<updatePatterns>`-Tag, und Instanzen der Mitglieder werden in dem Dokument beim Aktualisieren vollständig ersetzt. Bearbeitet ein Benutzer ein bestehendes Serververhalten und bestätigt die Änderung mit **OK**, werden alle Mitglieder gelöscht, deren Parameterwerte sich geändert haben, und mit dem neuen Wert an derselben Position wieder eingefügt.

Ändert ein Benutzer den Mitgliedercode im Dokument individuell ab, kann es sein, dass das Mitglied nicht mehr erkannt wird, wenn mit den Suchmustern nach dem alten Code gesucht wird. Mit kürzeren Suchmustern kann der Benutzer den Mitgliedercode in seinem Dokument anpassen. Durch Aktualisieren der Serververhaltensinstanz wird das Mitglied jedoch möglicherweise ersetzt werden, wodurch die Anpassungen dann verloren gehen könnten.

Präzise Aktualisierung In manchen Fällen ist es günstig, die Benutzer den Mitgliedercode anpassen zu lassen, nachdem er in das Dokument eingefügt wurde. Zu diesem Zweck können die Suchparameter eingeschränkt und Aktualisierungsmuster in der EDML-Datei bereitgestellt werden. Nachdem Sie das Mitglied zur Seite hinzugefügt haben, aktualisiert das Serververhalten nur bestimmte Teile davon. Nachfolgend ein Beispiel für ein einfaches Mitglied mit zwei Parametern:

```
<% if (Recordset1.EOF) Response.Redirect("some_url_here") %>
```

Dabei könnten folgende Suchmuster eingesetzt werden:

```
<quickSearch>Response.Write</quickSearch>
<searchPatterns whereToSearch="directive">
    <searchPattern paramNames="rs,new__url">
        /if\s*\((\w+)\.EOF\) \s*Response\.Redirect\("([^\r\n]*)"\)/i
    </searchPattern>
</searchPatterns>
```

Der Benutzer könnte dann einen anderen Text einer bestimmten Instanz dieses Codes hinzufügen:

```
<% if (Recordset1.EOF || x > 2) Response.Redirect("some_url_here") %>
```

Nun funktionieren die Suchmuster nicht mehr, denn sie suchen nach einer Klammer nach `EOF`. Um die Toleranz der Suchmuster zu erhöhen, können Sie sie in mehrere kürzere Komponenten aufteilen:

```
<quickSearch>Response.Write</quickSearch>
<searchPatterns whereToSearch="directive">
    <searchPattern paramNames="rs">/(\w+)\.EOF/</searchPattern>
    <searchPattern paramNames="new__url">
        /if\s*\("[^\r\n"]*\)\s*Response\.Redirect\("([^\r\n]*)"\)/i
    </searchPattern>
</searchPatterns>
```

Diese kürzeren Suchmuster sind flexibler, sodass der Benutzer dem Code weitere Elemente hinzufügen kann. Wenn durch das Serververhalten der URL verändert wird, nachdem der Benutzer auf **OK** klickt, wird das Mitglied ersetzt, und die Änderungen gehen verloren. Eine präzisere Aktualisierung erreichen Sie durch Hinzufügen eines `updatePatterns`-Tags, das ein Muster zum Aktualisieren jedes Parameters enthält.

```
<updatePatterns>
    <updatePattern paramNames="rs">/(\b)\w+(\.EOF)/</updatePattern>
    <updatePattern paramNames="new__url">
        /(Response\.Redirect\("([^\r\n]*)"\)/i
    </updatePattern>
</updatePatterns>
```

In Aktualisierungsmustern werden die Klammern in umgekehrter Weise verwendet und um den Text vor und nach dem Parameter gelegt. Verwenden Sie für Suchmuster den Parameter `textBeforeParam(param)textAfterParam`. Verwenden Sie für Aktualisierungsmuster den Parameter `(textBeforeParam)param(textAfterParam)`. Der gesamte Text zwischen zwei parenthetischen Teilausdrücken wird durch den neuen Parameterwert ersetzt.

Serververhalten löschen

Standardmäßige Löschung und Abhängigkeitszahlen Der Benutzer kann eine im Bedienfeld **Serververhalten** ausgewählte Instanz löschen, indem er auf die Schaltfläche mit dem Minuszeichen (–) klickt oder die Taste <Entf> drückt. Alle Mitglieder werden gelöscht. Ausgenommen sind Mitglieder, auf die auch andere Serververhalten zugreifen. Wenn bei mehreren Serververhalten ein Mitgliederverweis auf derselbe Knoten vorhanden ist, wird der Knoten nicht gelöscht.

Zum Löschen der Mitglieder wird standardmäßig das gesamte Tag entfernt. Wenn die Einfügeposition "wrapSelection" lautet, wird nur das äußere Tag entfernt. Bei Attributen wird die gesamte Attributdeklaration entfernt. Das folgende Beispiel zeigt ein Mitglied des ACTION-Attributs eines `form`-Tags:

```
<form action="<% my_participant %">
```

Nach dem Löschen verbleibt nur `form`.

Mitgliederlöschung durch delete-Tags einschränken In manchen Fällen kann es wünschenswert sein, die Art, wie Mitglieder gelöscht werden, einzuschränken. Hierzu kann der EDML-Datei ein `delete`-Tag hinzugefügt werden. Nachfolgend ein Beispiel für ein Mitglied, das ein `href`-Attribut eines Hyperlinks ist:

```
<a href="<%=MY_URL%>">Link Text</a>
```

Wenn dieses Mitglied gelöscht wird, lautet das resultierende Tag `<a>Link Text</a>` und wird in Dreamweaver nicht mehr als Hyperlink angezeigt. Hier kann es sinnvoll sein, nur den Attributwert zu entfernen. Zu diesem Zweck können Sie der EDML-Mitgliederdatei das folgende Tag hinzufügen:

```
<delete deleteType="innerOnly"/>
```

Eine andere Möglichkeit besteht darin, das gesamte Tag zu entfernen, wenn das Attribut gelöscht wird, indem Sie `<delete deleteType="tagOnly"/>` eingeben. Das Ergebnis ist der Text *Link Text*.

Konflikte mit SHARE-IN-MEMORY-JavaScript-Dateien vermeiden

Enthalten mehrere HTML-Dateien Referenzen auf eine bestimmte JavaScript-Datei, lädt Dreamweaver das JavaScript an einen zentralen Ort, auf dem die HTML-Dateien auf dieselbe JavaScript-Quelle zugreifen können. Diese Dateien enthalten folgende Zeile:

```
//SHARE-IN-MEMORY=true
```

Wenn Ihre Datei mit der Direktive `SHARE-IN-MEMORY` versehen ist und eine HTML-Datei darauf Bezug nimmt (durch das `SCRIPT`-Tag mit dem Attribut `SRC`), lädt Dreamweaver die Datei an einen zentralen Ort, an dem sie anschließend implizit in alle HTML-Dateien einbezogen wird.

Hinweis: Da an diesen zentralen Speicherort geladene JavaScript-Dateien Arbeitsspeicher gemeinsam nutzen, können die Dateien keinerlei Deklarationen duplizieren. Wenn eine `SHARE-IN-MEMORY`-Datei eine Variable oder eine Funktion definiert, die auch von einer anderen JavaScript-Datei definiert wird, entsteht ein Namenskonflikt. Beachten Sie daher beim Schreiben neuer JavaScript-Dateien diese Dateien und ihre Benennungskonventionen.

Kapitel 18: Datenquellen

Über die Funktionen der Datenquellen-API von Adobe Dreamweaver CS3 können Sie im Bedienfeld **Bindungen** Datenquellen in das Menü mit dem Pluszeichen (+) einfügen (weitere Informationen finden Sie unter der Funktion `dreamweaver.dbi.getDataSources()` im *Dreamweaver API-Referenzhandbuch*).

Datenquellendateien werden im Ordner **Configuration\DataSources\Servermodell-Name** gespeichert. Dreamweaver unterstützt zurzeit folgende Servermodelle: ASP.NET/C#, ASP.NET/VisualBasic, ASP/JavaScript, ASP/VBScript, ColdFusion, JSP und PHP/MySQL. In jedem Unterordner für die Servermodelle befinden sich HTML- und EDML-Dateien, die mit dem jeweiligen Servermodell in Zusammenhang stehen.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Datenquellen:

Pfad	Datei	Beschreibung
Configuration\DataSources\ServerModelName	Datenquellename.htm	Gibt den Namen der Datenquelle und den Speicherort der unterstützten JavaScript-Dateien an.
Configuration\DataSources\ServerModelName	Datenquellename.edml	Definiert den Code, den Dreamweaver zur Darstellung des Datenquellenwerts in das Dokument einfügt.
Configuration\DataSources\ServerModelName	Datenquellename.js	Enthält die JavaScript-Funktionen zum Hinzufügen, Einfügen und Löschen des erforderlichen Codes in einem Dokument.

Funktionsweise von Datenquellen

Benutzer von Dreamweaver können über das Bedienfeld **Bindungen** dynamische Daten hinzufügen. Die dynamischen Datenobjekte, die in dem Menü mit dem Pluszeichen (+) angezeigt werden, basieren auf dem Servermodell der jeweiligen Seite. So können Sie beispielsweise Datensätze und Befehle einfügen oder Variablen wie Sitzungsvariablen oder Anwendungsvariablen für ASP-Anwendungen anfordern.

Im Folgenden werden die einzelnen Schritte beim Hinzufügen dynamischer Daten beschrieben:

1 Wenn der Benutzer im Bedienfeld **Bindungen** auf die Schaltfläche mit dem Pluszeichen (+) klickt, wird das Pop-upmenü eingeblendet.

Um den Inhalt des Menüs bestimmen zu können, sucht Dreamweaver zunächst nach der Datei **DataSources.xml**, die sich im selben Ordner wie die Datenquellen befindet (zum Beispiel **Configuration\DataSources\ASP_js\DataSources.xml**). Die Datei **DataSources.xml** beschreibt den Inhalt des Kontextmenüs. Sie enthält Verweise auf HTML-Dateien, die in das Kontextmenü eingefügt werden sollten.

Dreamweaver untersucht jede aufgeführte HTML-Datei auf title-Tags. Enthält die Datei ein title-Tag, wird der Inhalt des title-Tags im Menü angezeigt. Enthält die Datei kein title-Tag, wird der Dateiname im Menü angezeigt.

Nachdem Dreamweaver die Datei DataSources.xml ausgewertet hat bzw. wenn die Datei nicht vorhanden ist, durchsucht das Programm den Rest des Ordners nach anderen Elementen, die im Menü angezeigt werden sollen. Wenn Dreamweaver Dateien im Hauptordner vorfindet, die sich nicht bereits im Menü befinden, werden diese dem Menü hinzugefügt. Wenn Unterordner gefunden werden, in denen sich Dateien befinden, die sich nicht bereits im Menü befinden, erstellt Dreamweaver ein Untermenü und fügt diese Dateien dem Untermenü hinzu.

2 Wenn ein Benutzer im Menü mit dem Pluszeichen (+) einen Menüpunkt auswählt, ruft Dreamweaver die Funktion `addDynamicSource()` auf, sodass dem Dokument des Benutzers Code für die Datenquelle hinzugefügt werden kann.

3 Dreamweaver durchläuft sämtliche Dateien im Ordner des entsprechenden Servermodells und ruft die Funktion `findDynamicSources()` in jeder Datei auf. Für jeden Wert im zurückgegebenen Array ruft Dreamweaver die Funktion `generateDynamicSourceBindings()` in derselben Datei auf, um eine aktuelle Liste aller Datenquellen für das Dokument des Benutzers zu erhalten. Diese Felder werden dem Benutzer in Form eines Struktursteuerelements innerhalb der Dialogfelder **Dynamische Daten** oder **Dynamischer Text** bzw. im Bedienfeld **Bindungen** angezeigt. Die Strukturansicht der Datenquelle für ein ASP-Dokument könnte wie nachfolgend aussehen:

```
Recordset (Recordset1)
    ColumnOneInRecordset
    ColumnTwoInRecordset
Recordset (Recordset2)
    ColumnOfRecordset
Anforderung
    NameOfRequestVariable
    NameOfAnotherRequestVariable
Session
    NameOfSessionVariable
```

4 Wenn der Benutzer im Bedienfeld **Bindungen** auf den Namen einer Datenquelle doppelklickt, um diese zu ändern, ruft Dreamweaver die Funktion `editDynamicSource()` auf, um damit die Änderungen innerhalb der Struktur zu verarbeiten.

5 Wenn der Benutzer auf die Schaltfläche mit dem Minuszeichen (–) klickt, erhält Dreamweaver die aktuelle Knotenauswahl aus der Datenquellenstruktur und leitet diese an die Funktion `deleteDynamicSource()` weiter, die den zuvor mit der Funktion `addDynamicSource()` hinzugefügten Code löscht. Wenn die aktuelle Auswahl nicht gelöscht werden kann, gibt die Funktion eine Fehlermeldung zurück. Nach Rückgabe der Funktion `deleteDynamicSource()` aktualisiert Dreamweaver die Datenquellenstruktur, indem es die Funktionen `findDynamicSources()` und `generateDynamicSourceBindings()` aufruft.

6 Wenn ein Benutzer eine Datenquelle auswählt und im Dialogfeld **Dynamische Daten** oder **Dynamischer Text** auf **OK** klickt bzw. im Bedienfeld **Bindungen** auf **Einfügen** oder **Binden** klickt, ruft Dreamweaver die Funktion `generateDynamicDataRef()` auf. Der zurückgegebene Wert wird im Dokument an der Stelle eingefügt, an der sich die Einfügemarke gerade befindet.

7 Wenn der Benutzer das Dialogfeld **Dynamische Daten** oder **Dynamischer Text** aufruft, um ein bereits vorhandenes dynamisches Datenobjekt zu bearbeiten, muss die Auswahl in der Datenquellenstruktur auf das dynamische Datenobjekt initialisiert werden. Zur Initialisierung des Struktursteuerelements durchläuft Dreamweaver sämtliche Dateien im entsprechenden Servermodellordner (beispielsweise im Ordner **Configuration\DataSources\ASP_js**) und ruft die Implementierung der Funktion `inspectDynamicDataRef()` in jeder Datei auf.

Dreamweaver ruft die Funktion `inspectDynamicDataRef()` auf, um das dynamische Datenobjekt vom Code im Dokument des Benutzers wieder in ein Element in der Strukturansicht zu konvertieren. (Dieser Prozess hat die umgekehrte Wirkung wie der Aufruf der Funktion `generateDynamicDataRef()`.) Wenn die Funktion `inspectDynamicDataRef()` ein Array mit zwei Elementen zurückgibt, zeigt Dreamweaver visuell an, welches Strukturelement mit der aktuellen Auswahl verbunden ist.

8 Jedes Mal, wenn der Benutzer die Auswahl ändert, ruft Dreamweaver die Funktion `inspectDynamicDataRef()` auf, um zu ermitteln, ob es sich bei der neuen Auswahl um dynamischen Text oder ein Tag mit einem dynamischen Attribut handelt. Wenn es sich um dynamischen Text handelt, zeigt Dreamweaver die Bindungen für die aktuelle Auswahl im Bedienfeld **Bindungen** an.

9 Mithilfe der Dialogfelder **Dynamische Daten** oder **Dynamischer Text** bzw. des Bedienfelds **Bindungen** ist es möglich, das Datenformat für ein dynamisches Textobjekt oder ein dynamisches Attribut zu ändern, das der Benutzer bereits in die Seite eingefügt hat. Wenn sich das Format ändert, ruft Dreamweaver die Funktion `generateDynamicDataRef()` auf, um den String abzurufen, der in das Dokument des Benutzers eingefügt werden soll. Anschließend wird der String an die Funktion `formatDynamicDataRef()` übergeben (siehe „`formatDynamicDataRef()`“ auf Seite 293). Der von der Funktion `formatDynamicDataRef()` zurückgegebene String wird in das Dokument des Benutzers eingefügt.

Ein einfaches Datenquellenbeispiel

Diese Erweiterung hängt eine benutzerdefinierte Datenquelle an das Bedienfeld **Bindungen** für ColdFusion-Dokumente an. Benutzer können die gewünschte Variable der neuen Datenquelle angeben.

In diesem Beispiel wird eine Datenquelle namens **MyDataSource** erstellt, die die JavaScript-Datei **MyDataSource.js**, die Datei **MyDataSource_DataRef.edml** sowie die „MyDataSource_Variable“-Befehlsdateien zum Implementieren eines Dialogfelds umfasst, in dem Benutzer den Namen einer bestimmten Variablen eingeben können. Das Beispiel **MyDataSource** basiert auf der Implementierung der Datenquelle **Cookie-Variable** und der Datenquelle **URL-Variable**. Die Dateien für diese Datenquellen befinden sich im Ordner **Configuration\DataSources\ColdFusion**.

Führen Sie zum Erstellen dieser Datenquelle die folgenden Schritte aus:

- „Datenquellen-Definitionsdateien erstellen“ auf Seite 277
- „EDML-Dateien erstellen“ auf Seite 278
- „JavaScript-Dateien erstellen, mit denen die API-Funktionen von Datenquellen implementiert werden“ auf Seite 278
- „Unterstützende Befehlsdateien für Benutzereingaben erstellen“ auf Seite 280
- „Neue Datenquelle testen“ auf Seite 282

Datenquellen-Definitionsdateien erstellen

Die Datenquellen-Definitionsdatei teilt Dreamweaver den Namen der Datenquelle mit, die im Menü mit dem Pluszeichen (+) des Bedienfelds **Bindungen** angezeigt wird. Darüber hinaus enthält sie Angaben darüber, wo sich die unterstützenden JavaScript-Dateien für die Implementierung der Datenquelle befinden.

Wenn ein Benutzer auf das Menü mit dem Pluszeichen (+) des Bedienfelds **Bindungen** klickt, durchsucht Dreamweaver den Ordner **DataSources** für das aktuelle Servermodell, um alle verfügbaren Datenquellen, die in den HTML-Dateien (.htm) definiert sind, zusammenzutragen. Um Benutzern eine neue Datenquelle zur Verfügung zu stellen, müssen Sie daher eine Datenquellen-Definitionsdatei erstellen, die den Namen der Datenquelle mithilfe des Tags `title` und den Speicherort aller unterstützender JavaScript-Dateien mithilfe des Tags `script` angibt.

Außerdem werden zur Implementierung dieser Datenquelle mehrere unterstützende Dateien benötigt. Im Allgemeinen sind die Funktionen in diesen unterstützenden Dateien nicht unbedingt erforderlich, jedoch oft hilfreich (und in diesem Beispiel notwendig). Beispielsweise enthält die Datei **dwscriptsServer.js** die Funktion `dwscripts.stripCFOutputTags()`, die zur Implementierung dieser Datenquelle verwendet wird. Zudem erstellen Sie mithilfe der Datei **DataSourceClass.js** eine Instanz der Klasse **DataSource**, die als Rückgabewert der Funktion `findDynamicSources()` verwendet wird.

So erstellen Sie die Datenquellen-Definitionsdatei:

1 Erstellen Sie eine neue, leere Datei.

2 Geben Sie Folgendes ein:

```
<HTML>
<HEAD>
<TITLE>MyDataSource</TITLE>
<SCRIPT SRC="../../Shared/Common/Scripts/dwscripts.js"></SCRIPT>
<SCRIPT SRC="../../Shared/Common/Scripts/dwscriptsServer.js"></SCRIPT>
<SCRIPT SRC="../../Shared/Common/Scripts/DataSourceClass.js"></SCRIPT>
<SCRIPT SRC="MyDataSource.js"></SCRIPT>
</HEAD>
<body></body>
</HTML>
```

3 Speichern Sie die Datei unter dem Namen **MyDataSource.htm** im Ordner **Configuration\DataSources\ColdFusion**.

EDML-Dateien erstellen

Die EDML-Datei definiert den Code, den Dreamweaver zur Darstellung des Datenquellenwerts in das Dokument einfügt. (Weitere Informationen zu EDML-Dateien finden Sie in „Serververhalten“ auf Seite 233). Wenn ein Benutzer einem Dokument einen bestimmten Wert aus einer Datenquelle hinzufügt, fügt Dreamweaver den Code ein, mit dem der eigentliche Wert zur Laufzeit berechnet wird. Die Mitglieder-EDML-Datei definiert den Code für das Dokument (weitere Informationen finden Sie unter „Mitglieder-EDML-Dateien“ auf Seite 251).

Für die MyDatasource-Variable soll Dreamweaver den ColdFusion-Code `<cfoutput>#MyXML.variable#</cfoutput>` einfügen, wobei *variable* der Wert ist, den der Benutzer aus der Datenquelle benötigt.

So erstellen Sie die EDML-Datei:

- 1 Erstellen Sie eine neue, leere Datei.
- 2 Geben Sie Folgendes ein:

```
<participant>
  <quickSearch><![CDATA[#]]></quickSearch>
  <insertText location="replaceSelection"><![CDATA[<cfoutput>#MyDatasource.
    @@bindingName@@#</cfoutput>]]></insertText>
  <searchPatterns whereToSearch="tag+cfoutput">
    <searchPattern paramNames="sourceName, bindingName"><![CDATA[/#(?:\s*\w+\s*\()?(
      (MyDatasource)\.(\w+)\b(?:\s*\s*/i)]></searchPattern>
  </searchPatterns>
</participant>
```

- 3 Speichern Sie die Datei unter dem Namen **MyDatasource_DataRef.edml** im Ordner **Configuration\DataSources\ColdFusion**.

JavaScript-Dateien erstellen, mit denen die API-Funktionen von Datenquellen implementiert werden

Nachdem Sie den Namen der Datenquelle, den Namen der unterstützenden JavaScript-Datei und den Code für das Arbeitsdokument von Dreamweaver definiert haben, müssen Sie die JavaScript-Funktionen für Dreamweaver angeben, damit der Benutzer den notwendigen Code einem Dokument hinzufügen, darin einfügen oder daraus löschen kann.

Ausgehend vom Aufbau der Datenquelle **Cookie-Variable** können Sie, wie im folgenden Beispiel gezeigt, die Datenquelle **MyXML** implementieren. (Der Befehl `MyDatasource_Variable`, der in der `addDynamicSource()`-Funktion verwendet wird, ist unter „Unterstützende Befehlsdateien für Benutzereingaben erstellen“ auf Seite 280 definiert.)

So erstellen Sie die JavaScript-Datei:

- 1 Erstellen Sie eine neue leere Datei.
- 2 Geben Sie Folgendes ein:

```
//***** GLOBALS VARS *****
var MyDatasource_FILENAME = "REQ_D.gif";
var DATASOURCELEAF_FILENAME = "DSL_D.gif";

//***** API *****
function addDynamicSource()
{
  MM.retVal = "";
  MM.MyDatasourceContents = "";
  dw.popupCommand("MyDatasource_Variable");
  if (MM.retVal == "OK")
  {
    var theResponse = MM.MyDatasourceContents;
    if (theResponse.length)
    {
      var siteURL = dw.getSiteRoot();
      if (siteURL.length)
      {
        dwscripts.addListValueToNote(siteURL, "MyDatasource", theResponse);
      }
    }
  }
}
```

```

        }
        else
        {
            alert(MM.MSG_DefineSite);
        }
    }
    else
    {
        alert(MM.MSG_DefineMyDatasource);
    }
}

function findDynamicSources()
{
    var retList = new Array();

    var siteURL = dw.getSiteRoot();

    if (siteURL.length)
    {
        var bindingsArray = dwscripts.getListValuesFromNote(siteURL, "MyDatasource");
        if (bindingsArray.length > 0)
        {
            // Hier wird zum Speichern der Rückgabewerte eine Instanz
            // der DataSource-Klasse laut Definition in der Datei DataSourceClass.js erstellt.

            retList.push(new DataSource("MyDatasource",
                                        MyDatasource_FILENAME,
                                        false,
                                        "MyDatasource.htm"))
        }
    }

    return retList;
}

function generateDynamicSourceBindings(sourceName)
{
    var retVal = new Array();

    var siteURL = dw.getSiteRoot();

    // Für lokalisierten Objektnamen
    if (sourceName != "MyDatasource")
    {
        sourceName = "MyDatasource";
    }

    if (siteURL.length)
    {
        var bindingsArray = dwscripts.getListValuesFromNote(siteURL, sourceName);
        retVal = getDataSourceBindingList(bindingsArray,
                                           DATASOURCELEAF_FILENAME,
                                           true,
                                           "MyDatasource.htm");
    }

    return retVal;
}

function generateDynamicDataRef(sourceName, bindingName, dropObject)
{
    var paramObj = new Object();
    paramObj.bindingName = bindingName;

```

```

var retStr = extPart.getInsertString("", "MyDatasource_DataRef", paramObj);

// Beim Einfügen in ein CFOUTPUT-Tag oder Binden an die Attribute eines
// ColdFusion-Tags müssen die cfoutput-Tags entfernt werden. Deshalb wird
// die Funktion dwscripts.canStripCfOutputTags() aus dwscriptsServer.js verwendet.

if (dwscripts.canStripCfOutputTags(dropObject, true))
{
    retStr = dwscripts.stripCfOutputTags(retStr, true);
}
return retStr;
}

function inspectDynamicDataRef(expression)
{
    var retArray = new Array();

    if(expression.length)
    {
        var params = extPart.findInString("MyDatasource_DataRef", expression);
        if (params)
        {
            retArray[0] = params.sourceName;
            retArray[1] = params.bindingName;
        }
    }

    return retArray;
}

function deleteDynamicSource(sourceName, bindingName)
{
    var siteURL = dw.getSiteRoot();

    if (siteURL.length)
    {
        //For localized object name
        if (sourceName != "MyDatasource")
        {
            sourceName = "MyDatasource";
        }

        dwscripts.deleteListValueFromNote(siteURL, sourceName, bindingName);
    }
}

```

3 Speichern Sie die Datei unter dem Namen **MyDatasource.htm** im Ordner **Configuration\DataSources\ColdFusion**.

Unterstützende Befehlsdateien für Benutzereingaben erstellen

Die Funktion `addDynamicSource()` enthält den Befehl `dw.popupCommand("MyDatasource_Variable")`, mit dem ein Dialogfeld geöffnet wird, in das ein Benutzer einen bestimmten Variablennamen eingeben kann. Sie müssen aber dennoch das Dialogfeld **MyDatasource Variable** erstellen.

Um ein Dialogfeld für Benutzereingaben bereitzustellen, müssen Sie einen neuen Satz Befehlsdateien erstellen: eine Befehls-Definitionsdatei in HTML und eine Befehls-Implementierungsdatei in JavaScript (weitere Informationen über Befehlsdateien finden Sie unter „Funktionsweise von Befehlen“ auf Seite 123).

Die Befehls-Definitionsdatei teilt Dreamweaver den Speicherort der unterstützenden JavaScript-Dateien für die Implementierung sowie die Art des Dialogfelds mit, das dem Benutzer angezeigt wird. Mit der unterstützenden JavaScript-Datei werden die Schaltflächen für das Dialogfeld und die Zuweisung der Benutzereingaben aus dem Dialogfeld definiert.

So erstellen Sie die Befehlsdefinitionsdatei:

1 Erstellen Sie eine neue, leere Datei.

2 Geben Sie Folgendes ein:

```

<!DOCTYPE HTML SYSTEM "-//Macromedia//DWEExtension layout-engine 5.0//dialog">
<html>
<head>
<title>MyDatasource Variable</title>
<script src="MyDatasource_Variable.js"></script>
<SCRIPT SRC="../../Shared/MM/Scripts/CMN/displayHelp.js"></SCRIPT>
<SCRIPT SRC="../../Shared/MM/Scripts/CMN/string.js"></SCRIPT>
<link href="../../fields.css" rel="stylesheet" type="text/css">
</head>
<body>
<form>
  <div ALIGN="center">
    <table border="0" cellpadding="2" cellspacing="4">
      <tr>
        <td align="right" valign="baseline" nowrap>Name:</td>
        <td align="baseline" nowrap>
          <input name="theName" type="text" class="medTField">
        </td>
      </tr>
    </table>
  </div>
</form>
</body>
</html>

```

3 Speichern Sie die Datei unter dem Namen **MyDatasource_Variable.htm** im Ordner **Configuration\Commands**.

Hinweis: Die Datei **MyDatasource_Variable.js** ist die Implementierungsdatei, die im nächsten Schritt erstellt wird.

So erstellen Sie die unterstützende JavaScript-Datei:

1 Erstellen Sie eine neue, leere Datei.

2 Geben Sie Folgendes ein:

```

//***** API *****

function commandButtons() {
  return new Array(MM.BTN_OK,"okClicked()",MM.BTN_Cancel,"window.close()");
}

//***** LOKALE FUNKTIONEN *****

function okClicked() {
  var nameObj = document.forms[0].theName;

  if (nameObj.value) {
    if (IsValidVarName(nameObj.value)) {
      MM.MyDatasourceContents = nameObj.value;
      MM.retVal = "OK";
      window.close();
    } else {
      alert(nameObj.value + " " + MM.MSG_InvalidParamName);
    }
  } else {
    alert(MM.MSG_NoName);
  }
}

```

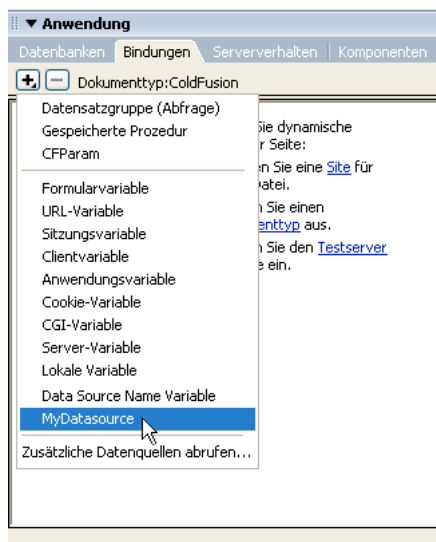
3 Speichern Sie die Datei unter dem Namen **MyDatasource_Variable.js** im Ordner **Configuration\Commands**.

Neue Datenquelle testen

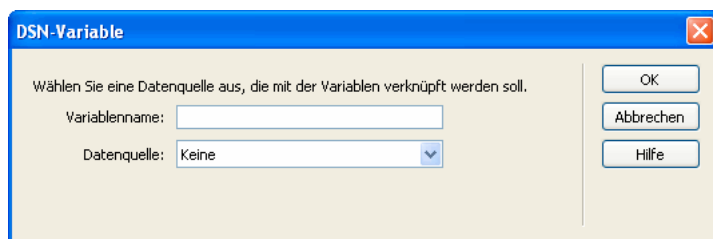
Sie können Dreamweaver nun öffnen (bzw. neu starten, wenn es bereits geöffnet ist) und eine ColdFusion-Datei öffnen oder eine neue erstellen.

So testen Sie Ihre neue Datenquelle:

1 Klicken Sie auf das Menü mit dem Pluszeichen (+) des Bedienfelds **Bindungen**, um alle verfügbaren Datenquellen anzuzeigen. **MyDatasource** sollte ganz unten in der Liste angezeigt werden:

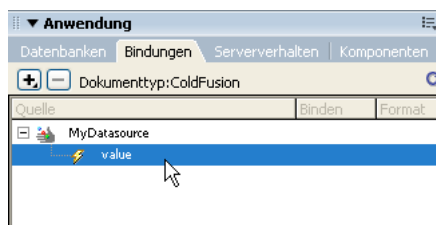


2 Klicken Sie auf die Datenquellenoption **MyDatasource**. Daraufhin wird das von Ihnen erstellte Dialogfeld **MyData-source Variable** geöffnet:

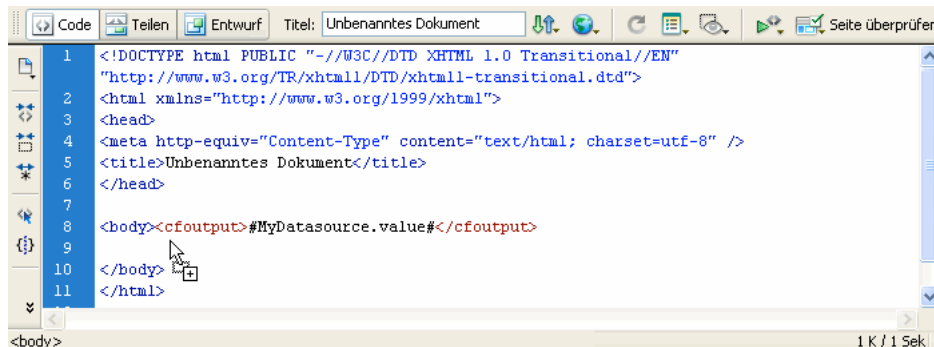


3 Geben Sie in das Dialogfeld einen Wert ein, und klicken Sie auf **OK**.

Die Datenquelle wird in einer Strukturansicht des Bedienfelds **Bindungen** angezeigt, wobei sich die Variable aus dem Dialogfeld unterhalb des Datenquellennamens befindet:



4 Ziehen Sie die Variable in Ihr Dokument. Dreamweaver fügt dann den entsprechenden Code aus der EDML-Datei ein:



Die API-Funktionen für Datenquellen

Über die Funktionen in der Datenquellen-API können Sie Datenquellen suchen, hinzufügen, bearbeiten und löschen sowie dynamische Datenobjekte erzeugen und überprüfen.

addDynamicSource()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Diese Funktion fügt eine dynamische Datenquelle hinzu. Da sich in jeder Datenquellendatei eine Implementierung dieser Funktion befindet, ruft Dreamweaver die entsprechende Implementierung der Funktion `addDynamicSource()` auf, sobald im Menü mit dem Pluszeichen (+) eine Datenquelle ausgewählt wurde.

Dreamweaver ruft für Datensätze oder Befehle beispielsweise die Funktion `dw.serverBehaviorInspector.popupServerBehavior()` auf, die in das Dokument ein neues Serververhalten einfügt. Für Anfragevariablen, Sitzungsvariablen und Anwendungsvariablen zeigt Dreamweaver ein HTML/JavaScript-Dialogfeld an, um die Eingabe des Variablennamens zu ermöglichen. Das Serververhalten speichert dann den Namen der Variablen zum späteren Gebrauch.

Nachdem die Funktion `addDynamicSource()` einen Wert zurückgibt, löscht Dreamweaver den Inhalt der Datenquellenstruktur und ruft die Funktionen `findDynamicSources()` und `generateDynamicSourceBindings()` auf, um die Datenquellenstruktur wieder mit Daten zu füllen.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

deleteDynamicSource()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Dreamweaver ruft diese Funktion auf, wenn ein Benutzer eine Datenquelle aus einer Strukturansicht auswählt und auf die Schaltfläche mit dem Minuszeichen (–) klickt.

Wenn zum Beispiel eine Datensatzgruppe oder ein Befehl ausgewählt wurde, wird in Dreamweaver über die Funktion `deleteDynamicSource()` die Funktion `dw.serverBehaviorInspector.deleteServerBehavior()` aufgerufen. Wurde eine Anfragevariable, Sitzungsvariable oder Anwendungsvariable ausgewählt, speichert die Funktion den Löschvorgang der Variablen und zeigt diese nicht mehr an. Nachdem die Funktion `deleteDynamicSource()` einen Wert zurückgibt, löscht Dreamweaver den Inhalt der Datenquellenstruktur und ruft die Funktionen `findDynamicSources()` und `generateDynamicSourceBindings()` auf, um so eine aktuelle Liste sämtlicher Datenquellen für das Dokument des Benutzers abzurufen.

Argumente

sourceName, bindingName

- Das Argument *sourceName* ist der Name des übergeordneten Knotens, dem der untergeordnete Knoten zugeordnet ist.
- Das Argument *bindingName* ist der Name des untergeordneten Knotens.

Rückgabewerte

Dreamweaver erwartet keinen Wert.

displayHelp()

Beschreibung

Wenn diese Funktion definiert ist, wird die Schaltfläche **Hilfe** unterhalb der Schaltflächen **OK** und **Abbrechen** im Dialogfeld angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche **Hilfe** klickt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet keinen Wert.

Beispiel

```
// Die folgende Instanz von displayHelp() öffnet
// (in einem Browser) eine Datei, die die Verwendung
// der Erweiterung erläutert.
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/superDuperHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

editDynamicSource()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion wird aufgerufen, wenn ein Benutzer im Bedienfeld **Bindungen** auf den Namen einer Datenquelle doppelklickt, um diese zu bearbeiten. Sie können diese Funktion implementieren, um damit vom Benutzer vorgenommene Änderungen in der Struktur verarbeiten zu lassen. Andernfalls wird automatisch das Serververhalten aufgerufen, das der Datenquelle zugeordnet ist. Der Erweiterungsentwickler kann mithilfe dieser Funktion die Standardimplementierung von Serververhalten überschreiben und eine benutzerdefinierte Verarbeitungsprozedur integrieren.

Argumente

sourceName, bindingName

- Das Argument *sourceName* ist der Name des übergeordneten Knotens, dem der untergeordnete Knoten zugeordnet ist.
- Das Argument *bindingName* ist der Name des untergeordneten Knotens.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert. *true*, wenn die Funktion die Änderung verarbeitet hat, andernfalls *false*.

findDynamicSources()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Diese Funktion gibt die Knoten der obersten Ebene der Datenquellenstruktur zurück, die im Dialogfeld **Dynamische Daten** oder **Dynamischer Text** bzw. im Bedienfeld **Bindungen** angezeigt wird. Jede Datenquellendatei verfügt über eine Implementierung der Funktion `findDynamicSources()`. Wenn Dreamweaver eine Struktur aktualisiert, werden alle Dateien im Ordner **DataSources** gelesen, und in jeder Datei wird die Funktion `findDynamicSources()` aufgerufen.

Rückgabewerte

Dreamweaver erwartet ein Array aus JavaScript-Objekten, von denen jedes bis zu fünf Eigenschaften haben kann. Diese werden in der folgenden Liste beschrieben:

- Die Eigenschaft *title* ist der Beschriftungsstring, der rechts neben dem Symbol der übergeordneten Knoten angezeigt wird. Die Eigenschaft *title* muss immer definiert werden.
- Die Eigenschaft *imageFile* ist der Pfad zur GIF-Datei, die das Symbol enthält, mit der der übergeordnete Knoten im Struktursteuerelement der Dialogfelder **Dynamische Daten** oder **Dynamischer Text** bzw. des Bedienfelds **Bindungen** dargestellt wird. Diese Eigenschaft ist erforderlich.
- Die Eigenschaft *allowDelete* ist optional. Wenn diese Eigenschaft auf *false* gesetzt ist und der Benutzer im Bedienfeld **Bindungen** auf diesen Knoten klickt, wird die Schaltfläche mit dem Minuszeichen (–) deaktiviert. Ist diese Eigenschaft auf *true* gesetzt, wird die Schaltfläche mit dem Minuszeichen (–) aktiviert. Wenn die Eigenschaft nicht definiert wurde, lautet die Standardeinstellung *true*.
- Die Eigenschaft *dataSource* ist der Name der Datei, in der die Funktion `findDynamicSources()` definiert ist. Beispielsweise setzt die Funktion `findDynamicSources()` in der Datei **Session.htm**, die sich im Ordner **Configuration\DataSources\ASP_Js** befindet, die Eigenschaft *dataSource* auf *session.htm*. Diese Eigenschaft ist erforderlich.
- Die Eigenschaft *name* ist der Name des Serververhaltens, das mit der Datenquelle verknüpft ist (falls vorhanden). Einige Datenquellen, wie Datensatzgruppen, sind mit Serververhalten verknüpft. Wenn Sie eine Datensatzgruppe erstellen und ihr den Namen *rsAuthors* zuweisen, sollte das Attribut *name* mit dem Wert von *rsAuthors* übereinstimmen. Die Eigenschaft *name* wird immer definiert, kann aber ein leerer String ("") sein, wenn mit der Datenquelle (z. B. einer Sitzungsvariablen) kein Serververhalten verknüpft ist.

Hinweis: Eine JavaScript-Klasse, die diese Eigenschaften definiert, ist in der Datei **DataSourceClass.js** im Ordner **Configuration\Shared\Common\Scripts** gespeichert.

generateDynamicDataRef()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Diese Funktion generiert das dynamische Datenobjekt für einen untergeordneten Knoten.

Argumente

sourceName, bindingName

- Das Argument *sourceName* ist der Name des übergeordneten Knotens, mit dem der untergeordnete Knoten in Verbindung steht.
- Das Argument *bindingName* ist der Name des untergeordneten Knotens, aus dem das dynamische Datenobjekt erzeugt werden soll.

Rückgabewerte

Dreamweaver erwartet einen String, der zur Formatierung an die Funktion `formatDynamicDataRef()` weitergegeben wird, bevor er in das Dokument eines Benutzers eingefügt wird.

generateDynamicSourceBindings()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Diese Funktion gibt die untergeordneten Knoten eines übergeordneten Knotens zurück.

Argumente

sourceName

- Das Argument *sourceName* ist der Name des übergeordneten Knotens, dessen untergeordnete Knoten zurückgegeben werden sollen.

Rückgabewerte

Dreamweaver erwartet ein Array aus JavaScript-Objekten, von denen jedes bis zu vier Eigenschaften haben kann. Diese werden in der folgenden Liste beschrieben:

- Die Eigenschaft `title` ist der Bezeichnungsstring, der rechts neben dem Symbol des übergeordneten Knotens angezeigt wird. Diese Eigenschaft ist erforderlich.
- Die Eigenschaft `allowDelete` ist optional. Wenn diese Eigenschaft auf den Wert `false` eingestellt wird und der Benutzer im Bedienfeld **Bindungen** auf diesen Knoten klickt, wird die Schaltfläche mit dem Minuszeichen (–) deaktiviert. Ist diese Eigenschaft auf den Wert `true` gesetzt, wird die Schaltfläche mit dem Minuszeichen (–) aktiviert. Wenn die Eigenschaft nicht definiert wurde, lautet der Standardwert `true`.
- Die Eigenschaft `dataSource` ist der Name der Datei, in der die Funktion `findDynamicSources()` definiert ist. Beispielsweise setzt die Funktion `findDynamicSources()` in der Datei **Session.htm**, die sich im Ordner **Configuration\DataSources\ASP_Js** befindet, die Eigenschaft `dataSource` auf `session.htm`. Diese Eigenschaft ist erforderlich.
- Die Eigenschaft `name` ist der Name des Serververhaltens, das mit der Datenquelle verknüpft ist (falls vorhanden). Diese Eigenschaft ist erforderlich. Einige Datenquellen, wie Datensatzgruppen, sind mit Serververhalten verknüpft. Wenn Sie eine Datensatzgruppe erstellen und ihr den Namen `rsAuthors` zuweisen, sollte die Eigenschaft `name` mit dem Wert von `rsAuthors` übereinstimmen. Andere Datenquellen, wie beispielsweise Sitzungsvariablen, verfügen nicht über zugehörige Serververhalten. Ihre Eigenschaft „name“ sollte daher ein leerer String ("") sein.

Hinweis: Eine JavaScript-Klasse, die diese Eigenschaften definiert, ist in der Datei **DataSourceClass.js** im Ordner **Configuration\Shared\Common\Scripts** gespeichert.

inspectDynamicDataRef()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Diese Funktion bestimmt den entsprechenden Knoten in der Datenquellenstruktur eines dynamischen Datenobjekts. Die Funktion `inspectDynamicDataRef()` vergleicht den von Dreamweaver übergebenen String mit dem String, der von `generateDynamicDataRef()` für jeden Knoten in der Struktur zurückgegeben wurde. Wenn eine Übereinstimmung gefunden wird, zeigt die Funktion `inspectDynamicDataRef()` an, welcher Knoten der Struktur mit dem übergebenen String übereinstimmt. Die Knoten werden mithilfe eines Arrays identifiziert, das zwei Elemente enthält. Das erste Element ist der Name des übergeordneten Knotens, das zweite ist der Name des untergeordneten Knotens. Wenn keine Übereinstimmung gefunden wird, gibt die Funktion `inspectDynamicDataRef()` ein leeres Array zurück.

Jede Implementierung der Funktion `inspectDynamicDataRef()` sucht nur nach Übereinstimmungen der eigenen Objektart. Die Datensatzgruppen-Implementierung der Funktion `inspectDynamicDataRef()` findet beispielsweise nur dann Übereinstimmungen, wenn der übergebene String mit einem Datensatzgruppenknoten in der Struktur übereinstimmt.

Argumente

string

- Das Argument *string* ist das dynamische Datenobjekt.

Rückgabewerte

Dreamweaver erwartet ein Array aus zwei Elementen (übergeordneter Name und untergeordneter Name) für jeden übereinstimmenden Knoten. Wenn keine Übereinstimmungen gefunden werden, wird der Wert `null` zurückgegeben.

Kapitel 19: Serverformate

Unter „Datenquellen“ auf Seite 275 wird erläutert, wie in Dreamweaver CS3 dynamische Daten in das Dokument des Benutzers eingefügt werden, indem an der entsprechenden Stelle ein Serverausdruck hinzugefügt wird. Wird das Dokument des Benutzers anschließend von einem Webserver abgerufen, wird dieser Serverausdruck in einen Wert aus einer Datenbank, den Inhalt einer Abfragevariable oder einen anderen dynamischen Wert konvertiert. Mit dem Serverformat von Dreamweaver kann der Benutzer festlegen, in welchem Format ihm dieser dynamische Wert der Website angezeigt wird.

In diesem Abschnitt wird die API behandelt, mit der die dynamischen Daten formatiert werden, die von den in „Datenquellen“ auf Seite 275 beschriebenen Funktionen zurückgegeben werden. Die Funktionen, die in beiden Abschnitten beschrieben werden, arbeiten bei der Formatierung dynamischer Daten zusammen. Wenn der Benutzer ein Format für die dynamischen Daten wählt, ruft Dreamweaver die Datenquellenfunktion `generateDynamicDataRef()` auf, um den String abzurufen, der in das Dokument des Benutzers eingefügt werden soll (siehe „`generateDynamicDataRef()`“ auf Seite 285). Vor dem Einfügen des Strings in das Dokument des Benutzers übergibt Dreamweaver den String an die Funktion `formatDynamicDataRef()`, die in diesem Abschnitt beschrieben wird. Der von der Funktion `formatDynamicDataRef()` zurückgegebene String sind die formatierten dynamischen Daten, die nun in das Dokument des Benutzers eingefügt werden.

In Dreamweaver können Sie Daten mithilfe der integrierten Formattypen formatieren. Weiterhin können Sie auf Grundlage der integrierten Formattypen bzw. mithilfe benutzerdefinierter Formattypen neue Formate erstellen.

Die dynamischen Daten können auf mehrere Arten formatiert werden. Mit den Befehlen im Menü **Format**, in den Dialogfeldern **Dynamische Daten** oder **Dynamischer Text** bzw. dem Bedienfeld **Bindungen** kann der Benutzer die Daten formatieren, bevor sie in ein HTML-Dokument eingefügt werden. Der Benutzer kann ein Format erstellen, indem er im Menü **Format** auf die Option **Formatliste bearbeiten** und dann im Plusmenü (+) auf den gewünschten Formattyp klickt. Das Plusmenü (+) enthält eine Liste der Formattypen. Bei Formattypen handelt es sich um grundlegende Formatkategorien, wie beispielsweise „Währung“, „Datum/Uhrzeit“ oder „Groß-/Kleinschreibung“. Formattypen sammeln die gemeinsamen Parameter für eine Formatkategorie, wodurch das Erstellen eines neuen Formats vereinheitlicht wird.

Ein neues Format könnte beispielsweise ein anderes Währungsformat sein. Die Währungsformatierung besteht im Wesentlichen aus den folgenden Schritten: Konvertieren einer Zahl in einen String und Einfügen von Trennzeichen und Dezimalkommata sowie eines Währungssymbols (z. B. des Dollar-Zeichens \$). Im Datentyp des Währungsformats werden alle gemeinsamen Parameter zusammengefasst, und Sie werden aufgefordert, die erforderlichen Werte für diese Parameter einzugeben.

Funktionsweise der Datenformatierung

Alle Formatdateien befinden sich im Ordner **Configuration\ServerFormats\AktuellesServermodell**. Die einzelnen Unterordner enthalten je eine XML-Datei und mehrere HTML-Dateien.

In der Datei **Formats.xml** werden alle Wahlmöglichkeiten des Menüs **Format** beschrieben. Dreamweaver fügt die Optionen **Formatliste bearbeiten** und **Keine** hinzu.

In dem Ordner befindet sich außerdem eine HTML-Datei für jeden aktuell installierten Formattyp. Hierzu gehören **Groß-/Kleinschreibung**, **Währung**, **Datum/Uhrzeit**, **Mathematik**, **Nummer**, **Prozent**, **Einfach** und **Feineinstellung**.

Die Datei Formats.xml

Die Datei Formats.xml enthält ein Tag `format` für jede Option im Menü **Format**. Jedes Tag `format` beinhaltet die folgenden erforderlichen Attribute:

- Das Attribut `file=fileName` ist die HTML-Datei für diesen Formattyp (z. B. "Währung").
- Das Attribut `title=string` ist der String, der im Menü **Format** angezeigt wird (z. B. "Währung - Standard").
- Das Attribut `expression=regexp` ist ein regulärer Ausdruck, der dynamische Datenobjekte zuordnet, die dieses Format verwenden. Der Ausdruck legt fest, welches Format aktuell auf ein dynamisches Datenobjekt angewendet wird. Der Ausdruck für das Format "Währung - Standard" lautet beispielsweise "`<%\s*=\s*FormatCurrency\(.*, -1, -2, -2, -2\)\s*%>|<%\s*=\s*DoCurrency\(.*, -1, -2, -2, -2\)\s*%>`". Der Wert des Attributs `expression` muss unter allen `format`-Tags in der Datei eindeutig sein. Er muss spezifisch genug sein, um zu gewährleisten, dass nur Instanzen dieses Formats mit dem Ausdruck übereinstimmen.
- Das Attribut `visibility=[hidden | visible]` zeigt an, ob der Wert im Menü **Format** angezeigt wird. Wenn das Attribut `visibility` den Wert `hidden` aufweist, wird das Format nicht im Menü **Format** angezeigt.

Das Tag `format` kann weitere, willkürlich benannte Attribute enthalten.

Einige Funktionen für die Datenformatierung erfordern das Argument `format`. Hierbei handelt es sich um ein JavaScript-Objekt. Dieses Objekt ist der Knoten, der dem Tag `format` in der Datei Formats.xml entspricht. Das Objekt sollte eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden Tags `format` enthalten.

Das folgende Beispiel zeigt das Tag `format` für den String "Währung - Standard":

```
<format file="Currency" title="Currency - default" ~
expression="<%\s*=\s*FormatCurrency\(.*, -1, -2, -2, -2\)\s*%>|~
  <%\s*=\s*DoCurrency\(.*, -1, -2, -2, -2\)\s*%>"
NumDigitsAfterDecimal=-1 IncludeLeadingDigit=-2 ~
  UseParensForNegativeNumbers=-2 GroupDigits=-2/>
```

Dieses Format hat den Typ `Währung`. Der String "Währung - Standard" wird im Menü **Format** angezeigt. Der Ausdruck `<%\s*=\s*FormatCurrency\(.*, -1, -2, -2, -2\)\s*%>|<%\s*=\s*DoCurrency\(.*, -1, -2, -2, -2\)\s*%>` findet Instanzen dieses Formats im Dokument des Benutzers.

Die Parameter `NumDigitsAfterDecimal`, `IncludeLeadingDigit`, `UseParensForNegativeNumbers` und `GroupDigits` sind nicht unbedingt erforderliche Parameter für den Formattyp `Währung`. Diese Parameter werden im Dialogfeld **Parameter** für den Formattyp `Währung` angezeigt. Das Dialogfeld **Parameter** wird angezeigt, wenn der Benutzer den Formattyp `Währung` im Plusmenü (+) des Dialogfelds **Formatliste bearbeiten** wählt. Mit den für diese Parameter angegebenen Werten wird das neue Format definiert.

Das Plusmenü (+) des Dialogfelds „Formatliste bearbeiten“

Soll eine Datei im Ordner **ServerFormats** nicht im Plusmenü (+) des Dialogfelds **Formatliste bearbeiten** angezeigt werden, fügen Sie den folgenden String als erste Zeile in die HTML-Datei ein:

```
<!-- MENU-LOCATION=NONE -->
```

Um den Inhalt des Menüs bestimmen zu können, sucht Dreamweaver zunächst nach der Datei **DataSources.xml**, die sich in demselben Ordner wie die Datenquellen (zum Beispiel in **Configuration\ServerFormats\ASP**) befindet. Die Datei **ServerFormats.xml** beschreibt den Inhalt des Plusmenüs (+) des Dialogfelds **Formatliste bearbeiten**. Sie enthält Verweise auf HTML-Dateien, die im Menü aufgeführt werden sollen.

Dreamweaver untersucht jede aufgeführte HTML-Datei auf title-Tags. Enthält die Datei ein title-Tag, wird der Inhalt des title-Tags im Menü angezeigt. Enthält die Datei kein title-Tag, wird der Dateiname im Menü angezeigt.

Nachdem Dreamweaver die Suche nach der Datei **DataSources.xml** ausgewertet hat bzw. wenn die Datei nicht vorhanden ist, durchsucht das Programm den Rest des Ordners nach anderen Elementen, die im Menü angezeigt werden sollen. Wenn Dreamweaver Dateien im Hauptordner vorfindet, die sich nicht bereits im Menü befinden, werden diese dem Menü hinzugefügt. Wenn Unterordner mit Dateien gefunden werden, die sich nicht bereits im Menü befinden, erstellt Dreamweaver ein Untermenü und fügt diese Dateien in das Untermenü ein.

Aufruf der Funktionen zur Datenformatierung

Die Funktionen zur Datenformatierung werden in den folgenden Fällen aufgerufen:

- Der Benutzer wählt im Dialogfeld **Dynamische Daten** oder **Dynamischer Text** einen Knoten aus der Datenquellenstruktur und im Popupmenü **Format** ein Format aus. Wenn der Benutzer das Format auswählt, ruft Dreamweaver die Funktion `generateDynamicDataRef()` auf und übergibt den Rückgabewert der Funktion `generateDynamicDataRef()` an die Funktion `formatDynamicDataRef()`. Der Rückgabewert der Funktion `formatDynamicDataRef()` wird im Dialogfeld unter der Einstellung **Code** angezeigt. Wenn der Benutzer auf **OK** klickt, wird der Codestring in das Dokument des Benutzers eingefügt. Nun ruft Dreamweaver die Funktion `applyFormat()` auf, um eine Funktionsdeklaration einzufügen. Weitere Informationen finden Sie unter „generateDynamicDataRef()“ auf Seite 285. Ein ähnliches Verfahren wird durchgeführt, wenn der Benutzer mit dem Bedienfeld **Bindungen** arbeitet.
- Wenn der Benutzer das Format ändert oder das dynamische Datenobjekt löscht, wird die Funktion `deleteFormat()` aufgerufen. Mit der Funktion `deleteFormat()` werden die Unterstützungsskripts aus dem Dokument entfernt.
- Wenn der Benutzer im Dialogfeld **Formatliste bearbeiten** auf die Schaltfläche mit dem Pluszeichen (+) klickt, zeigt Dreamweaver ein Menü mit allen Formattypen für das jeweilige Servermodell an. Jeder Formattyp entspricht einer Datei im Ordner **Configuration\ServerFormats\AktuellesServermodell**.

Wählt der Benutzer im Plusmenü (+) ein Format aus, für das ein benutzerdefinierter Parameter erforderlich ist, führt Dreamweaver die `onload`-Prozedur für das `body`-Tag aus und zeigt dann das Dialogfeld **Parameter** an. Dieses Dialogfeld enthält die Parameter für den Formattyp. Wenn der Benutzer in diesem Dialogfeld Parameter für das Format auswählt und anschließend auf **OK** klickt, ruft Dreamweaver die Funktion `applyFormatDefinition()` auf.

Wenn für das ausgewählte Format kein Parameter-Dialogfeld angezeigt werden muss, ruft Dreamweaver die Funktion `applyFormatDefinition()` auf, wenn der Benutzer den Formattyp im Plusmenü (+) auswählt.

- Wenn der Benutzer später das Format bearbeitet (durch Auswahl des Formats im Dialogfeld **Formatliste bearbeiten** und Wahl der Schaltfläche **Bearbeiten**), ruft Dreamweaver vor dem Einblenden des Dialogfelds **Parameter** die Funktion `inspectFormatDefinition()` auf, sodass die Formular-Steuerelemente auf die korrekten Werte initialisiert werden können.

Die API-Funktionen für Serverformate

Die Serverformate-API umfasst die folgenden Funktionen zur Datenformatierung.

applyFormat()

Verfügbarkeit

Dreamweaver UltraDev 1

Beschreibung

Mit dieser Funktion wird ein Dokument eines Benutzers durch Hinzufügen einer Format-Funktionsdeklaration bearbeitet. Wenn ein Benutzer ein Format im Textfeld **Format** des Dialogfelds **Dynamische Daten** bzw. **Dynamischer Text** oder des Bedienfelds **Bindungen** auswählt, nimmt Dreamweaver zwei Änderungen am Dokument des Benutzers vor: Dreamweaver fügt die geeignete Formatfunktion vor dem HTML-Tag hinzu (sofern diese nicht bereits vorhanden ist) und ändert das dynamische Datenobjekt, sodass dieses die geeignete Formatfunktion aufruft.

Dreamweaver fügt die Funktionsdeklaration hinzu, indem die JavaScript-Funktion `applyFormat()` in der Datenformatdatei aufgerufen wird. Das dynamische Datenobjekt wird durch Aufrufen der Funktion `formatDynamicDataRef()` geändert.

Die Funktion `applyFormat()` sollte das DOM verwenden, um Funktionsdeklarationen ganz oben im Dokument hinzuzufügen. Wenn der Benutzer beispielsweise **Währung - Standard** auswählt, fügt die Funktion die Funktionsdeklaration `währung` hinzu.

Argumente

format

- Das Argument *format* ist ein JavaScript-Objekt, das das anzuwendende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem Tag *format* in der Datei **Formats.xml** entspricht. Das Objekt sollte eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden Tags *format* enthalten.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

applyFormatDefinition()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Übernimmt die Änderungen eines Formats, das mit dem Menü **Format bearbeiten** erstellt wurde.

Mit dem Dialogfeld **Formatliste bearbeiten** können Formate erstellt, bearbeitet und gelöscht werden. Diese Funktion wird aufgerufen, damit die Änderungen, die an einem Format vorgenommen wurden, wirksam werden. Weiterhin können mit dieser Funktion zusätzliche, willkürlich benannte Eigenschaften für das Objekt festgelegt werden. Jede Eigenschaft wird als Attribut des Tags *format* in der Datei **Formats.xml** gespeichert.

Argumente

format

- Das Argument *format* entspricht dem JavaScript-Objekt *format*. Diese Funktion muss die Eigenschaft *expression* des JavaScript-Objekts auf den regulären Ausdruck für das Format einstellen. Weiterhin können mit dieser Funktion zusätzliche, willkürlich benannte Eigenschaften für das Objekt festgelegt werden. Die einzelnen Eigenschaften werden als Attribut des Tags *format* gespeichert.

Rückgabewerte

Dreamweaver erwartet das Formatobjekt, wenn die Funktion erfolgreich ausgeführt wird. Wenn ein Fehler auftritt, gibt die Funktion einen Fehlerstring zurück. Sollte ein leerer String zurückgegeben werden, wird das Formular geschlossen, doch das neue Format wird nicht erstellt. Dies entspricht dem Befehl **Abbrechen**.

deleteFormat()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Entfernt die Format-Funktionsdeklaration aus dem oberen Teil des Dokuments.

Wenn der Benutzer das Format eines dynamischen Datenobjekts ändert (im Dialogfeld **Dynamische Daten** bzw. **Dynamischer Text** oder im Bedienfeld **Bindungen**) oder ein formatiertes dynamisches Datenobjekt löscht, ruft Dreamweaver die Funktion `deleteFormat()` auf, um die Funktionsdeklaration aus dem oberen Teil des Dokuments zu entfernen und den Funktionsaufruf aus dem dynamischen Datenobjekt zu löschen.

Verwenden Sie das DOM mit der Funktion `deleteFormat()`, um die Funktionsdeklaration oben aus dem aktuellen Dokument zu entfernen.

Argumente

format

Das Argument *format* ist ein JavaScript-Objekt, das das zu entfernende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem Tag *format* in der Datei **Formats.xml** entspricht.

Rückgabewerte

Dreamweaver erwartet keinen Wert.

formatDynamicDataRef()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Fügt dem dynamischen Datenobjekt den Format-Funktionsaufruf hinzu. Wenn ein Benutzer ein Format im Textfeld **Format** des Dialogfelds **Dynamische Daten** bzw. **Dynamischer Text** oder des Bedienfelds **Bindungen** auswählt, nimmt Dreamweaver zwei Änderungen am Dokument des Benutzers vor: Dreamweaver fügt die geeignete Formatfunktion vor dem HTML-Tag hinzu (sofern diese nicht bereits vorhanden ist) und ändert das dynamische Datenobjekt, sodass dieses die geeignete Formatfunktion aufruft.

Dreamweaver fügt die Funktionsdeklaration hinzu, indem die JavaScript-Funktion `applyFormat()` in der Datenformatdatei aufgerufen wird. Das dynamische Datenobjekt wird durch Aufrufen der Funktion `formatDynamicDataRef()` geändert.

Die Funktion `formatDynamicDataRef()` wird aufgerufen, wenn der Benutzer ein Format im Feld **Format** des Bedienfelds **Bindungen** oder des Dialogfeld **Dynamische Daten** bzw. **Dynamischer Text** auswählt. Durch diese Funktion wird das Dokument nicht bearbeitet.

Argumente

dynamicDataObject, format

- Das Argument *dynamicDataObject* ist ein String, der das dynamische Datenobjekt enthält.
- Das Argument *format* ist ein JavaScript-Objekt, das das anzuwendende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem Tag *format* in der Datei **Formats.xml** entspricht. Das Objekt sollte eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden Tags *format* enthalten.

Rückgabewerte

Dreamweaver erwartet den neuen Wert für das dynamische Datenobjekt.

Wenn ein Fehler eintritt, zeigt die Funktion unter bestimmten Umständen eine Warnmeldung an. Wenn die Funktion einen leeren String zurückgibt, wurde im Feld **Format** die Option **Keine** ausgewählt.

inspectFormatDefinition()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Initialisiert Formular-Steuerelemente, wenn der Benutzer ein vorhandenes Format im Dialogfeld **Formatliste bearbeiten** ändert.

Argumente

format

Das Argument *format* ist ein JavaScript-Objekt, das das anzuwendende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem Tag *format* in der Datei **Formats.xml** entspricht. Das Objekt sollte eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden Tags *format* enthalten.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Kapitel 20: Komponenten

Mit Adobe Dreamweaver CS3 können Sie viele der gängigsten Komponententypen erstellen. Darüber hinaus können Sie mit Dreamweaver die im Bedienfeld **Komponenten** angezeigten Komponententypen erweitern.

Grundlagen zu Komponenten

Programmierer greifen zum Zusammenfassen („Kapseln“) ihrer Arbeit auf unterschiedliche Verfahren zurück. Stellen Sie sich die *Zusammenfassung* wie das Erzeugen einer in einer virtuellen Blackbox vorhandenen Größe vor. Um sie zu verwenden, müssen Sie nicht verstehen, wie sie funktioniert. Sie müssen nur wissen, welche Informationen für den Vorgang benötigt werden und welche Informationen er ausgibt. Nehmen wir an, ein Programmierer schreibt ein Programm, das Informationen aus einer Mitarbeiterdatenbank abrufen. Alle Personen sowie andere Programme können dann mithilfe dieses Programms die Datenbank abfragen. Dieses Programm ist also wiederverwendbar.

Die Erfahrung hat gezeigt, dass gut organisierte Programme mit Kapselung einfacher zu warten, zu verbessern und wieder zu verwenden sind. Verschiedene Technologien bieten unterschiedliche Möglichkeiten zur Zusammenfassung, und diese Strategien werden unterschiedlich bezeichnet: *Funktionen*, *Module* und andere. In Adobe Dreamweaver werden mit dem Begriff *Komponente* einige der bekanntesten und modernsten Zusammenfassungsstrategien wie Webdienste, JavaBeans und ColdFusion-Komponenten (CFCs) bezeichnet. Wenn Benutzer Webanwendungen in Dreamweaver erstellen, werden sie durch die im Bedienfeld **Komponenten** verfügbaren Webdienste, JavaBeans und CFCs unterstützt.

Komponenten aus neuen Technologien (wie Webdienste, JavaBeans oder CFCs) sind selbsterklärend. In der Regel liegen in den die Komponente bildenden Dateien eingebettete Informationen über die Komponente vor. Die Fähigkeit einer Komponente, diese Informationen nach außen mitzuteilen oder freizugeben, nennt man *Introspektion*. Anders gesagt, ein Programm wie Dreamweaver kann eine Komponente nach der Liste der bereitgestellten Funktionen fragen (d. h. nach Funktionen, die von einem anderen Programm aufgerufen werden können). Je nach verwendeter Technologie können Komponenten auch andere Informationen über sich selbst liefern. Ein Webdienst kann beispielsweise neue Datentypen beschreiben.

Das Bedienfeld „Komponenten“ erweitern

Wenn Sie eine Komponentenstrategie erfunden haben (oder verwenden möchten), die im Bedienfeld **Komponenten** von Dreamweaver derzeit nicht vorhanden ist, können Sie den Deklarationscode des Bedienfelds erweitern, sodass neue Komponenten aufgenommen werden können.

Um eine neue Art von Komponente zum Bedienfeld **Komponenten** in Dreamweaver hinzuzufügen, müssen Sie die verfügbaren Komponenten suchen (in der Benutzerumgebung) und von jeder Komponente eine Beschreibung anfordern (oder diese analysieren, wenn sie unter Verwendung von ASCII-Dateien geschrieben wurden).

Die genaue Methode zum Suchen von Komponenten und Abrufen von Komponentendetails unterscheidet sich je nach verwendeter Technologie. Außerdem gibt es Unterschiede je nach verwendetem Servermodell (ASP.NET, JSP/J2EE, ColdFusion usw.). Der von Ihnen geschriebene JavaScript-Code zur Erweiterung des Bedienfelds **Komponenten** muss also die hinzuzufügende Komponententechnologie berücksichtigen. Die hier beschriebenen Funktionen sollen Sie beim Abrufen von Informationen zur Anzeige im Bedienfeld **Komponenten** unterstützen. Den Skriptcode zum Suchen und Untersuchen von Komponenten (Abfrage der internen Struktur und Darstellung der Felder, Methoden und Eigenschaften in Dreamweaver) müssen Sie jedoch größtenteils selbst schreiben.

Dazu kommt, dass Servermodelle wie ASP.NET, JSP/J2EE und ColdFusion einige, aber nicht alle Komponententypen unterstützen. Beispielsweise unterstützt ASP.NET Webdienste, aber nicht JavaBeans. ColdFusion unterstützt Webdienste und CFCs. Wenn Sie einen neuen Komponententyp zum Bedienfeld **Komponenten** hinzufügen, muss dieser für das Servermodell geeignet sein. Wenn beispielsweise ein Dreamweaver-Benutzer an einer ColdFusion-Site arbeitet, sollten in der Pop-up-Menü im Bedienfeld Komponenten nur Webdienste und CF-Komponenten angezeigt werden.

Die zu ändernden Dateien werden in diesem Abschnitt näher erläutert. In einigen Fällen müssen Sie JavaScript-Code schreiben, der bestimmte komponentenbezogene Funktionen aufruft.

Anpassen des Bedienfelds „Komponenten“

Im Bedienfeld **Komponenten** von Dreamweaver können Benutzer Komponenten laden und damit arbeiten. Darin werden die Komponententypen aufgeführt, die für die jeweiligen, aktivierten Servermodelle verfügbar sind. Da beispielsweise JavaBeans nur für JavaServer-Seiten (JSP f. engl. „JavaServer Pages“) geeignet ist, können JavaBeans-Komponenten nur im Bedienfeld **Komponenten** eines JSP-Servers angezeigt werden. Genauso verhält es sich mit CFCs: Sie sind nur für ColdFusion-Seiten geeignet, werden also auch nur im Bedienfeld **Komponenten** eines ColdFusion-Servers angezeigt.

Im Rahmen der Erweiterbarkeit können Sie neue Komponententypen in das Bedienfeld aufnehmen. Wenn Sie neue Komponententypen in das Bedienfeld **Komponenten** einfügen, müssen Sie einige allgemeine Schritte durchführen:

- 1 Die Komponente muss in die Liste der verfügbaren Komponenten für die entsprechenden Servermodelle eingefügt werden.
- 2 Es ist eine Anleitung für die Einrichtung der Komponente hinzuzufügen, die in Form von interaktiven, nummerierten Schritten im Bedienfeld **Komponenten** oder in einem Dialogfeld angezeigt wird (abhängig von der Erweiterung, für welche die Schritte implementiert werden). Neben allen vom Benutzer abgeschlossenen Schritten muss ein Häkchen angezeigt werden.
- 3 Die Komponenten, deren Typ entweder nur auf dem Computer des Benutzers oder nur auf der aktuellen Site vorkommen, sind aufzulisten.
- 4 Wenn der Benutzer auf die Plus-Schaltfläche (+) im Bedienfeld **Komponenten** klickt, muss eine neue Komponente erstellt werden.

Darüber hinaus soll dem Benutzer wahrscheinlich auch die Möglichkeit eingeräumt werden, vorhandene Komponenten zu bearbeiten oder zu löschen.

Dateien für das Bedienfeld „Komponenten“

Im Ordner **Configuration\Components** gibt es einen Unterordner für jedes implementierte Servermodell. Komponentendateien werden im Ordner **Configuration\Components\Servermodell\Komponententyp** abgelegt. Sie können andere Servermodelle und unterstützende Servererweiterungen hinzufügen (weitere Informationen finden Sie in „Servermodelle“ auf Seite 307 und „Serververhalten“ auf Seite 233).

So erstellen Sie eine benutzerdefinierte Komponente für das Bedienfeld Komponenten:

- 1 Erstellen Sie eine HTML-Datei, die die Speicherorte der unterstützenden JavaScript- und Bilddateien bestimmt.
- 2 Schreiben Sie ein JavaScript zur Aktivierung der Komponente.
- 3 Bestimmen Sie die (neu erstellten oder bereits vorhandenen) GIF-Bilddateien, die die Komponente im Bedienfeld **Komponenten** darstellen.

Soll eine Komponente auch in der Strukturansicht angezeigt werden, müssen Sie die damit verknüpften optionalen Dateien erstellen und Daten in die Strukturansicht übernehmen.

Sie können eine Komponente nur für eine oder mehrere Webseiten oder für eine ganze Website einrichten. Ihr JavaScript-Code muss eine Ablauflogik für die Komponentenbeständigkeit enthalten, d. h. für die Speichervorgänge zwischen den Sitzungen und das Neuladen zu Beginn einer neuen Sitzung.

Das folgende Beispiel zeigt einen Dateneintrag in der Datei **JavaBeansList.xml** (gespeichert im Konfigurationsordner für mehrere Benutzer), der die Komponentenklasse und deren Speicherort angibt:

```
<javabeans>
<javabean classname="TestCollection.MusicCollection"
classlocation="d:\music\music.jar"></javabean>
</javabeans>
```

JavaBeans sollte den Code zum Speichern der Komponente in einem Verzeichnis enthalten, das für eine Mehrbenutzerumgebung konfiguriert ist, damit sich die Komponente beim nächsten Anwendungsstart selbst aus der gespeicherten Datendatei lädt.

Eine Dienstkomponente hinzufügen

So können Sie mit Dreamweaver einen neuen Zugriffsprotokoll-Dienst (LDAP, Lightweight Directory Access Protocol) hinzufügen:

1 Erstellen Sie anhand bestehender Komponentendateien (z. B. der Dateien im Anwendungsordner **Configuration\Components\Common\WebServices**) alle erforderlichen Dateien und die von Ihnen gewünschten optionalen Dateien, um den neuen Komponententyp anzuzeigen. Siehe dazu nachfolgende Tabelle:

Dateiname	Beschreibung	Erforderlich/Optional
*.htm	Die Erweiterungsdatei, die andere unterstützende JavaScript- und GIF-Dateien kennzeichnet.	Erforderlich
*.js	Die Erweiterungsdatei, die den Rückruf an die Komponenten-API implementiert.	Erforderlich
*.gif	Das Bild, das im Popupmenü Komponenten angezeigt wird.	Erforderlich
*Menus.xml	Der Speicherort für Metadaten zur Strukturierung des Bedienfelds Komponenten . Obwohl die allgemeine Webdienst-Komponente diese Datei nicht verwendet, können Sie die Datei WebServicesMenus.xml im Anwendungsordner Components\ColdFusion\WebServices als Beispiel verwenden.	Optional
*.gif	Bilder der Symbolleiste können ein- oder ausgeblendet werden, wie in folgendem Beispiel: ToolBarImageUp.gif ToolBarImageDown.gif ToolBarImageDisabled.gif Oder Strukturknotenbilder.	Optional

Hinweis: Sie sollten dasselbe Präfix für die Dateien einer Komponente verwenden, sodass diese Dateien problemlos ihren jeweiligen Komponenten zugeordnet werden können.

2 Schreiben Sie den JavaScript-Code zum Implementieren der neuen Serverkomponente.

Die Erweiterungsdatei (HTM) bestimmt den Speicherort von JavaScript-Code im Tag `SCRIPT`. Diese JavaScript-Dateien befinden sich entweder im Ordner **Shared**, im selben Ordner wie die Erweiterungsdatei oder im Ordner **Common** für Code, der für mehrere Servermodelle gilt.

Beispielsweise enthält die Datei WebServices.htm im Ordner **Configuration\Components\Common\WebServices** die folgende Zeile:

```
<SCRIPT SRC="../../Common/WebServices/WebServicesCommon.js"></SCRIPT>
```

Weitere Informationen zu den verfügbaren Funktionen für Komponenten-APIs finden Sie unter „API-Funktionen für das Bedienfeld „Komponenten““ auf Seite 298.



Beim Hinzufügen eines neuen Dienstes sollten Sie mit dem Bedienfeld **Komponenten** die Metadaten durchsuchen, sodass die Daten beim Erstellen der Erweiterung sofort verfügbar sind. Dreamweaver kann aufgenommene Komponenten durchsuchen und Knoten in der Komponentenstruktur anzeigen. Die Funktion **Ziehen und Ablegen** sowie die Tastatursteuerung werden in der Codeansicht des Bedienfelds **Komponenten** unterstützt.

Daten in die Strukturansicht übernehmen

Mithilfe der Eigenschaft `ComponentRec` können Sie eine Strukturansicht des Bedienfelds **Komponenten** auffüllen, sodass sie in diesem Bedienfeld an der richtigen Position angezeigt wird. Jeder Knoten in einer Strukturansicht muss folgende Eigenschaften aufweisen:

Name der Eigenschaft	Beschreibung	Erforderlich/Optional
<code>name</code>	Name des Strukturknotenelements	Erforderlich
<code>image</code>	Symbol für das Strukturknotenelement. Wenn nichts angegeben ist, wird ein Standardsymbol verwendet.	Optional
<code>hasChildren</code>	Wenn der Benutzer in der Strukturansicht auf die Schaltfläche mit dem Pluszeichen (+) oder Minuszeichen (-) klickt, werden untergeordnete Elemente geladen. Sie können so auch mit einer Struktur arbeiten, in die noch keine Daten übernommen wurden.	Erforderlich
<code>toolTipText</code>	QuickInfo-Text für das Strukturknotenelement	Optional
<code>isCodeViewDraggable</code>	Bestimmt, ob das Element durch Ziehen in die Codeansicht verschoben werden kann.	Optional
<code>isDesignViewDraggable</code>	Bestimmt, ob das Element durch Ziehen in die Entwurfsansicht verschoben werden kann.	Optional

Der folgende `WebServicesClass`-Knoten hat beispielsweise untergeordnete Webmethoden:

```
this.name = "TrafficLocatorWebService";
this.image = "Components/Common/WebServices/WebServices.gif";
this.hasChildren = true;
this.toolTipText = "TrafficLocatorWebService";
this.isCodeViewDraggable = true;
// Folgendes ermöglicht die Aktivierung/Deaktivierung der angezeigten Schaltfläche
// oberhalb der Komponentenstruktur
this.allowDelete = true;
this.isDesignViewDraggable = false;
```

API-Funktionen für das Bedienfeld „Komponenten“

In diesem Abschnitt werden die API-Funktionen für das Übertragen von Daten in das Bedienfeld **Komponenten** erläutert.

getComponentChildren()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Diese Funktion gibt eine Liste untergeordneter `ComponentRec`-Objekte für das aktive übergeordnete Objekt `ComponentRec` zurück. Damit die Strukturelemente der untersten Ebene geladen werden können, muss die Funktion Zugriff auf die Metadaten im Speicher haben.

Argumente

`{parentComponentRec}`

Das Argument `parentComponentRec` ist das Objekt `componentRec` des übergeordneten Elements. Fehlt es, erwartet Dreamweaver eine Liste der `ComponentRec`-Objekte für den Stammknoten.

Rückgabewerte

Ein Array von `ComponentRec`-Objekten.

Beispiel

Siehe function `GetComponentChildren(componentRec)` in der Datei **WebServices.js** im Ordner **Configuration\Components\Common\WebServices**.

getContextMenuId()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Ruft die ID für das Kontextmenü des Komponententyps ab. Jeder Komponententyp kann mit einem Kontextmenü verknüpft sein. Die Popupmenüs für das Kontextmenü werden in der Datei *ComponentNameMenus.xml* definiert, ihre Funktionsweise entspricht der Datei *menu.xml*. Der Menüstring kann statisch oder dynamisch sein. Tastaturbefehle werden unterstützt.

Argumente

Keine

Rückgabewerte

Ein String, der die ID für das Kontextmenü definiert.

Beispiel

Im folgenden Beispiel werden die Menüoptionen des Bedienfelds **Komponenten** für die zum Servermodell ASP.NET/C# gehörigen Webdienste festgelegt sowie Tastenkombinationen für das Menü definiert:

```
function getContextMenuId()
{
    return "DWWebServicesContext";
}
```

Dabei ist `DWWebServicesContext` in der Datei **Configuration\Components\ASP.NET_CSharp\WebServices\WebServicesMenus.xml** wie folgt definiert:

```
<shortcutlist id="DWWebServicesContext">
    <shortcut key="Del" domRequired="false"
        enabled="(dw.serverComponentsPalette.getSelectedNode() != null &&
            (dw.serverComponentsPalette.getSelectedNode().objectType=='Root'))"
        command="clickedDelete();" id="DWShortcuts_ServerComponent_Delete" />
</shortcutlist>

<menubar name="" id="DWWebServicesContext">
    <menu name="Server Component Popup" id="DWContext_WebServices">
        <menuitem name="Edit Web Service" domRequired="false"
            enabled="dw.serverComponentsPalette.getSelectedNode() != null &&
            (dw.serverComponentsPalette.getSelectedNode().objectType=='Root') &&
            dw.serverComponentsPalette.getSelectedNode().wsRec != null &&
            dw.serverComponentsPalette.getSelectedNode().wsRec.ProxyGeneratorName != null"
            command="editWebService()" id="DWContext_WebServices_EditWebService" />
        ...
    </menu>
</menubar>
```

getCodeViewDropCode()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion ruft den Code ab, der in der Codeansicht mit **Ziehen und Ablegen** bzw. über den Befehl **Ausschneiden**, **Kopieren** oder **Einfügen** aus dem Bedienfeld **Komponenten** verschoben wird.

Argumente

componentRec

- Das Argument *componentRec* ist ein Objekt.

Rückgabewerte

Der String, der den Code für die Komponente enthält.

Beispiel

Das folgende Beispiel enthält den Code für einen allgemeinen Webdienst:

```
function getCodeViewDropCode(componentRec)
{
    var codeToDrop="";
    if (componentRec)
    {
        if (componentRec.objectType=="Class")
        {
            codeToDrop = dw.getExtDataValue("webservices_constructor","insertText");
            codeToDrop = codeToDrop.replace(RegExp("@@Id@", "g"), componentRec.name);
            codeToDrop = codeToDrop.replace(RegExp("@@Class@", "g"), componentRec.name);
        }
        else if (componentRec.objectType=="Method")
        {
            codeToDrop = componentRec.dropCode;
        }
        if (componentRec.dropCode)
        {
            codeToDrop = componentRec.dropCode;
        }
        else
        {
            codeToDrop = componentRec.name;
        }
    }
    return codeToDrop;
}
```

getSetupSteps()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Dreamweaver ruft diese Funktion auf, wenn `setupStepsCompleted()` den Wert Null oder eine positive Ganzzahl zurückgibt. Die Funktion steuert die serverbasierten Setup-Anweisungen. Diese können mit den Erweiterungen implementiert werden, die ein modales Dialogfeld und Serverkomponenten verwenden.

Sie gibt ein Array der Strings zurück, das dann in Dreamweaver, je nach Art der Erweiterung, im Dialogfeld mit den Setup-Anweisungen oder im Bedienfeld **Komponenten** angezeigt wird.

Argumente

Keine

Rückgabewerte

Ein Array von $n+1$ Strings, wobei n für die Anzahl der auszuführenden Schritte steht:

- Der Titel, der zu Anfang der Setup-Schritte angezeigt wird
- Die Textanleitungen für jeden Schritt, die beliebigen HTML-Markup enthalten können, der für ein `<li>`-Tag zulässig ist.

Sie können Hyperlinks (`<a>`-Tags) im folgenden Format in die Schrittanweisungen aufnehmen:

```
<a href="#" onMouseDown="handler">Blue Underlined Text</a>
```

Der Wert `"handler"` kann durch einen der folgenden Strings für Ereignisprozeduren oder durch einen JavaScript-Ausdruck wie `„dw.browseDocument('http://www.adobe.com')“` ersetzt werden:

- Die Ereignisprozedur `"Event:SetCurSite"` öffnet ein Dialogfeld, in dem die aktuelle Site festgelegt werden kann.
- Die Ereignisprozedur `"Event:CreateSite"` öffnet ein Dialogfeld, in dem eine neue Site erstellt werden kann.
- Die Ereignisprozedur `"Event:SetDocType"` öffnet ein Dialogfeld, in dem der Dokumenttyp des Benutzerdokuments geändert werden kann.
- Die Ereignisprozedur `"Event:CreateConnection"` öffnet ein Dialogfeld, in dem eine neue Datenbankverbindung erstellt werden kann.
- Die Ereignisprozedur `"Event:SetRDSPassword"` öffnet ein Dialogfeld, in dem der Benutzername und das Kennwort für den Remote Development Service (RDS) festgelegt werden können (nur für ColdFusion).
- Die Ereignisprozedur `"Event:CreateCFDataSource"` öffnet den ColdFusion-Administrator in einem Browser.

Beispiel

Im folgenden Beispiel werden vier Schritte für ColdFusion-Komponenten angegeben. Der vierte Schritt enthält einen Hyperlink, sodass der Benutzer den RDS-Benutzernamen und das Kennwort eingeben kann:

```
function getSetupSteps()
{
    var doSDK = false;
    dom = dw.getDocumentDOM();
    if (dom && dom.serverModel)
    {
        var aServerModelName = dom.serverModel.getDisplayName();
    }
    else
    {
        var aServerModelName = site.getServerDisplayNameForSite();
    }
    if (aServerModelName.length)
    {
        if (aServerModelName != "ColdFusion")
        {
            if (needsSDKInstalled != null)
            {
                doSDK = needsSDKInstalled();
            }
        }
    }

    var someSteps = new Array();
    someSteps.push(MM.MSG_WebService_InstructionsTitle);
    someSteps.push(MM.MSG_Dynamic_InstructionsStep1);
    someSteps.push(MM.MSG_Dynamic_InstructionsStep2);
    if (doSDK == true)
    {
        someSteps.push(MM.MSG_WebService_InstructionsStep3);
    }
}
```

```

    }
    someSteps.push(MM.MSG_WebService_InstructionsStep4);

    return someSteps;
}

```

setupStepsCompleted()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Dreamweaver ruft diese Funktion auf, bevor die Registerkarte **Komponenten** angezeigt wird. Anschließend ruft Dreamweaver die Funktion `getSetupSteps()` auf, wenn die Funktion `setupStepsCompleted()` den Wert Null oder eine positive Ganzzahl zurückgibt.

Argumente

Keine

Rückgabewerte

Eine Ganzzahl, die die Anzahl der Schritte angibt, die der Benutzer bereits ausgeführt hat, wie in der folgenden Liste beschrieben:

- Der Wert Null oder eine positive Ganzzahl zeigt die Anzahl der bereits ausgeführten Schritte an.
- Der Wert -1 besagt, dass alle erforderlichen Schritte ausgeführt wurden. In diesem Fall wird die Anweisungsliste nicht mehr angezeigt.

handleDesignViewDrop()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Bestimmt den Ablegevorgang, wenn der Benutzer eine Tabelle oder Ansicht aus dem Bedienfeld **Datenbanken** oder eine Komponente aus dem Bedienfeld **Komponenten** in die Entwurfsansicht zieht.

Argumente

componentRec

- Das Argument *componentRec* ist ein Objekt, das folgende Eigenschaften enthält:
 - Die Eigenschaft *name* ist der Name des Strukturknotenelements.
 - Die Eigenschaft *image* ist ein optionales Symbol für das Strukturknotenelement. Ist es nicht vorhanden, verwendet Dreamweaver MX ein Standardsymbol.
 - Die Eigenschaft *hasChildren* ist ein Boolescher Wert, der anzeigt, ob untergeordnete Knoten zu einem Knoten vorhanden sind: Wenn *true*, zeigt Dreamweaver MX sowohl die Schaltfläche mit dem Pluszeichen (+) als auch die mit dem Minuszeichen (-) für das Strukturknotenelement an. Wenn *false*, ist das Element nicht erweiterbar.
 - Die Eigenschaft *toolTipText* ist eine optionale QuickInfo für das Strukturknotenelement.
 - Die Eigenschaft *isCodeViewDraggable* ist ein Boolescher Wert, der anzeigt, ob ein Knoten durch Ziehen und Ablegen in die Codeansicht verschoben werden kann.
 - Die Eigenschaft *isDesignViewDraggable* ist ein Boolescher Wert, der anzeigt, ob ein Knoten durch Ziehen und Ablegen in die Entwurfsansicht verschoben werden kann.

Rückgabewerte

Ein Boolescher Wert, der angibt, ob der Ablegevorgang erfolgreich war: `true`, wenn erfolgreich, andernfalls `false`.

Beispiel

Im folgenden Beispiel wird festgestellt, ob die Komponente eine Tabelle oder eine Ansicht ist, und dann der entsprechende Wert für `bHandled` zurückgegeben:

```
function handleDesignViewDrop(componentRec)
{
    var bHandled = false;
    if (componentRec)
    {
        if ((componentRec.objectType == "Table") ||
            (componentRec.objectType == "View"))
        {
            alert("popup Recordset Server Behavior");
            bHandled = true;
        }
    }
    return bHandled;
}
```

handleDoubleClick()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Wenn der Benutzer auf einen Knoten in der Struktur doppelklickt, wird die Ereignisprozedur aufgerufen, um die Bearbeitung freizugeben. Diese Funktion ist optional. Die Funktion kann `false` zurückgeben. Dies bedeutet, dass die Ereignisprozedur nicht definiert ist. In diesem Fall wird durch Doppelklicken das Standardverhalten ausgelöst, d. h. das Ein- oder Ausblenden der Knoten in der Struktur.

Argumente

componentRec

- Das Argument *componentRec* ist ein Objekt, das folgende Eigenschaften enthält:
 - Die Eigenschaft `name` ist der Name des Strukturknotenelements.
 - Die Eigenschaft `image` ist ein optionales Symbol für das Strukturknotenelement. Ist das Symbol nicht vorhanden, verwendet Dreamweaver ein Standardsymbol.
 - Die Eigenschaft `hasChildren` ist ein Boolescher Wert, der anzeigt, ob untergeordnete Knoten zu einem Knoten vorhanden sind: Wenn `true`, zeigt Dreamweaver sowohl die Schaltfläche mit dem Pluszeichen (+) als auch die mit dem Minuszeichen (–) für das Strukturknotenelement an. Wenn `false`, ist das Element nicht erweiterbar.
 - Die Eigenschaft `toolTipText` ist eine optionale QuickInfo für das Strukturknotenelement.
 - Die Eigenschaft `isCodeViewDraggable` ist ein Boolescher Wert, der anzeigt, ob ein Knoten durch Ziehen und Ablegen in die Codeansicht verschoben werden kann.
 - Die Eigenschaft `isDesignViewDraggable` ist ein Boolescher Wert, der anzeigt, ob ein Knoten durch Ziehen und Ablegen in die Entwurfsansicht verschoben werden kann.

Rückgabewerte

Keine

Beispiel

Im folgenden Beispiel wird angegeben, wie die Erweiterung auf einen Doppelklick auf ein Strukturknotenelement reagiert. Wird der Wert `false` zurückgegeben, werden die Knoten je nach Standardeinstellung ein- oder ausgeblendet.

```
function handleDoubleClick(componentRec)
{
    var selectedObj = dw.serverComponentsPalette.getSelectedNode();
    if(dwscripts.IS_WIN)
    {
        if (selectedObj && selectedObj.wsRec && selectedObj.wsRec[ProxyGeneratorNamePropName])
        {
            if (selectedObj.objectType == "Root")
            {
                editWebService();
                return true;
            }
            else if (selectedObj.objectType == "MissingProxyGen")
            {
                displayMissingProxyGenMessage(componentRec);
                editWebService();
                return true;
            }
        }
    }
    return false;
}
```

toolbarControls()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Jeder Komponententyp gibt eine Liste mit `toolbarButtonRec`-Objekten zurück. Diese stehen für Symbole der Symbolleiste in der Reihenfolge von links nach rechts. Jedes `toolbarButtonRec`-Objekt enthält die folgenden Eigenschaften:

Name der Eigenschaft	Beschreibung
<code>image</code>	Pfad zur Bilddatei
<code>disabledImage</code>	(Optional) Pfad zum Bild für die deaktivierte Schaltfläche – sucht nach der Symbolleisten-Schaltfläche
<code>pressedImage</code>	(Optional) Pfad zum Bild für die „gedrückte“ Schaltfläche – sucht nach der Symbolleisten-Schaltfläche
<code>toolTipText</code>	QuickInfo für Symbolleisten-Schaltfläche
<code>toolStyle</code>	Links/rechts
<code>enabled</code>	JavaScript-Code, der einen Booleschen Wert zurückgibt (<code>true</code> oder <code>false</code>). Diese Eigenschaften werden unter folgenden Umständen aufgerufen: • Wenn <code>dreamweaver.serverComponents.refresh()</code> aufgerufen wird • Wenn sich die Auswahl in der Struktur ändert • Wenn sich das Servermodell ändert
<code>command</code>	Der auszuführende JavaScript-Code. Die Befehlsprozedur kann eine Aktualisierung mit der Funktion <code>dreamweaver.serverComponents.refresh()</code> erzwingen.
<code>menuId</code>	Die eindeutige Menü-ID für die Popupmenü-Schaltfläche, wenn auf eine Schaltfläche geklickt wird. Ist diese ID vorhanden, setzt sie die Befehlsprozedur außer Kraft. Anders ausgedrückt, die Schaltfläche kann entweder mit einem Befehl oder mit einem Popupmenü verknüpft sein, nicht jedoch beides gleichzeitig.

Argumente

Keine

Rückgabewerte

Ein Array von Symbolleisten-Schaltflächen in der Reihenfolge von links nach rechts.

Beispiel

Im folgenden Beispiel werden den Symbolleisten-Schaltflächen Eigenschaften zugewiesen:

```

function toolbarControls()
{
    var toolBarBtnArray = new Array();
    dom = dw.getDocumentDOM();
    var plusButton = new ToolbarControlRec();
    var aServerModelName = null;
    if (dom && dom.serverModel)
    {
        aServerModelName = dom.serverModel.getDisplayName();
    }
    else
    {
        //auf der Site nach einem potentiellen Servermodell suchen
        aServerModelName = site.getServerDisplayNameForSite();
    }
    if (aServerModelName.length)
    {
        if (aServerModelName == "ColdFusion")
        {
            plusButton.image           = PLUS_BUTTON_UP;
            plusButton.pressedImage    = PLUS_BUTTON_DOWN;
            plusButton.disabledImage   = PLUS_BUTTON_UP;
            plusButton.toolStyle       = "left";
            plusButton.toolTipText     = MM.MSG_WebServicesAddToolTipText;
            plusButton.enabled         = "dwscripits.IS_WIN";
            plusButton.command         = "invokeWebService()";
        }
        else
        {
            plusButton.image           = PLUSDROPBUTTONUP;
            plusButton.pressedImage    = PLUSDROPBUTTONDOWN;
            plusButton.disabledImage   = PLUSDROPBUTTONUP;
            plusButton.toolStyle       = "left";
            plusButton.toolTipText     = MM.MSG_WebServicesAddToolTipText;
            plusButton.enabled         = "dwscripits.IS_WIN";
            plusButton.menuId          = "DWWebServicesChoosersContext";
        }
        toolBarBtnArray.push(plusButton);

        var minusButton              = new ToolbarControlRec();
        minusButton.image             = MINUSBUTTONUP;
        minusButton.pressedImage      = MINUSBUTTONDOWN;
        minusButton.disabledImage     = MINUSBUTTONDISABLED;
        minusButton.toolStyle         = "left";
        minusButton.toolTipText       = MM.MSG_WebServicesDeleteToolTipText;
        minusButton.command           = "clickedDelete()";
        minusButton.enabled           = "(dw.serverComponentsPalette.getSelectedNode() != null &&
            dw.serverComponentsPalette.getSelectedNode() &&
            ((dw.serverComponentsPalette.getSelectedNode().objectType=='Root') ||
            (dw.serverComponentsPalette.getSelectedNode().objectType == 'Error') ||
            (dw.serverComponentsPalette.getSelectedNode().objectType ==
            'MissingProxyGen'))))";
        toolBarBtnArray.push(minusButton);

        if (aServerModelName != null && aServerModelName.indexOf(".NET") >= 0)
        {
            var deployWServiceButton    = new ToolbarControlRec();
            deployWServiceButton.image   = DEPLOYSUPPORTBUTTONUP;
            deployWServiceButton.pressedImage = DEPLOYSUPPORTBUTTONDOWN;

```

```
        deployWServiceButton.disabledImage = DEPLOYSUPPORTBUTTONUP;
        deployWServiceButton.toolStyle      = "right";
        deployWServiceButton.toolTipText    = MM.MSG_WebServicesDeployToolTipText;
        deployWServiceButton.command        = "site.showTestingServerBinDeployDialog()";
        deployWServiceButton.enabled        = true;
        toolBarBtnArray.push(deployWServiceButton);
    }
    //Nur für Windows: die neu erstellte Proxy-Schaltfläche hinzufügen.
    //bug 45552:
    if(navigator.platform.charAt(0) != "M")
    {
        var proxyButton                = new ToolbarControlRec();
        proxyButton.image               = PROXYBUTTONUP;
        proxyButton.pressedImage        = PROXYBUTTONDOWN;
        proxyButton.disabledImage       = PROXYBUTTONDISABLED;
        proxyButton.toolStyle           = "right";
        proxyButton.toolTipText         = MM.MSG_WebServicesRegenToolTipText;
        proxyButton.command              = "reGenerateProxy()";
        proxyButton.enabled              = "enableRegenerateProxyButton()";

        toolBarBtnArray.push(proxyButton);
    }
}
return toolBarBtnArray;
}
```


Kapitel 21: Servermodelle

Servermodelle führen Skripts auf einem Server aus. Beim Definieren einer neuen Site können Benutzer das Servermodell festlegen, das sie auf Site-Ebene und auf Dokumentebene verwenden möchten. Dieses Servermodell verarbeitet alle dynamischen Elemente, die der Benutzer zum Dokument hinzufügt.

Konfigurationsdateien für Servermodelle sind im Ordner **Configuration\ServerModels** abgelegt. In diesem Ordner gibt es eine HTML-Datei für jedes Servermodell, die die dafür erforderlichen Funktionen implementiert.

Anpassung von Servermodellen

Mithilfe der API für Servermodelle können Sie die Funktionen für ein Servermodell an Ihre Anforderungen anpassen.

Beim ersten Aufrufen von Adobe Dreamweaver CS3 müssen Benutzer die zu verwendenden Servermodelle angeben. Für den Fall, dass ein Benutzer diese Angabe nicht macht, können Sie ein dynamisches Dialogfeld anlegen, mit dem der Benutzer aufgefordert wird, die erforderlichen Schritte auszuführen. Das Dialogfeld wird angezeigt, wenn der Benutzer versucht, ein Serverobjekt einzufügen. Weitere Informationen zum Erstellen dieses Dialogfelds finden Sie unter den Funktionen „getSetupSteps()“ auf Seite 300 und „setupStepsCompleted()“ auf Seite 302.

Sie können auch ein spezialisiertes Servermodell erstellen. Adobe empfiehlt, die im Lieferumfang von Dreamweaver enthaltenen Servermodelle nicht zu bearbeiten, sondern gleich neue zu erstellen. (Weitere Informationen zum Erstellen neuer Dokumenttypen für das von Ihnen verwendete Servermodell finden Sie unter „Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 15.)

Wenn Sie ein neues Servermodell erstellen, müssen Sie eine Implementierung der Funktion `canRecognizeDocument()` in die Servermodell-Datei einschließen. Diese Funktion gibt die Präferenzenebene Ihres Servermodells beim Bearbeiten eines Dateityps an Dreamweaver weiter, wenn eine bestimmte Dateinamenerweiterung mehreren Servermodellen zugewiesen wurde.

API-Funktionen für Servermodelle

In diesem Abschnitt werden die Funktionen beschrieben, die die Servermodelle für Dreamweaver konfigurieren.

canRecognizeDocument()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Beim Öffnen einer Datei, deren Dateityp mehreren Servermodellen zugewiesen ist, ruft Dreamweaver diese Funktion für jedes betreffende Servermodell auf, um zu prüfen, ob die Funktion das Dokument als eine ihr zugehörige Datei identifizieren kann. Ist eine Dateinamenerweiterung mehreren Servermodellen zugewiesen, erhält dasjenige Servermodell Priorität, für das die Funktion die höchste Ganzzahl zurückgibt.

Hinweis: Da alle für Dreamweaver definierten Servermodelle den Wert 1 zurückgeben, können Servermodelle von Drittherstellern die Zuweisung der Dateinamenerweiterungen überschreiben.

Argumente

dom

Das Argument *dom* ist das Adobe-Dokumentobjekt, das von der Funktion `dreamweaver.getDocumentDOM()` zurückgegeben wird.

Rückgabewerte

Dreamweaver erwartet eine Ganzzahl für die Priorität, die Sie dem Servermodell für die Dateinamenerweiterung zugewiesen haben. Diese Funktion sollte den Wert -1 zurückgeben, wenn einem Servermodell die Erweiterung nicht zugewiesen wurde. Andernfalls sollte ein Wert größer Null zurückgegeben werden.

Beispiel

Im nachfolgenden Beispiel gibt die Funktion den Wert 2 zurück, wenn der Benutzer ein JavaScript-Dokument für das aktuelle Servermodell öffnet. Durch diesen Wert erhält das aktuelle Servermodell Priorität vor dem Standard-Servermodell von Dreamweaver.

```
var retVal = -1;
var langRE = /@s*language\s*=\s*(\"|\'|)?javascript(\"|\'|)?/i;
// Suche nach dem String language="javascript"
var oHTML = dom.documentElement.outerHTML;
if (oHTML.search(langRE) > -1)
    retVal = 2;
return retVal;
```

getFileExtensions()

Verfügbarkeit

Dreamweaver UltraDev 1, nicht empfohlen für Dreamweaver MX

Beschreibung

Gibt die Dateinamenerweiterungen der Dokumentdateien zurück, mit denen ein Servermodell arbeiten kann. So unterstützt beispielsweise das Servermodell ASP die Erweiterungen .asp und .htm. Mit dieser Funktion wird ein Array von Strings zurückgegeben. Dreamweaver füllt mit diesen Strings die Liste **Standard-Seitenerweiterung** auf, die in der Kategorie **Anwendungsserver-Infos** des Dialogfelds **Site-Definition** zu finden ist.

Hinweis: Die Liste **Standard-Seitenerweiterung** ist nur in Dreamweaver 4 und früheren Versionen verfügbar. In Dreamweaver MX und späteren Versionen werden im Dialogfeld **Site-Definition** keine Einstellungen für die Dateinamenerweiterung angezeigt. Dreamweaver liest die Datei *Extensions.txt* und analysiert das Element *documenttype* in der Datei *mmDocumentTypes.xml*. (Weitere Informationen über diese beiden Dateien und das Element *documenttype* finden Sie unter „Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 15.)

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array an Strings für die gültigen Dateinamenerweiterungen.

getLanguageSignatures()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt ein Objekt zurück, das die Methode und die Array-Signaturen der Skriptsprache beschreibt. Mithilfe der Funktion `getLanguageSignatures()` kann die Zuordnung von generischen und sprachspezifischen Signaturen für folgende Elemente abgeglichen werden:

- Die Funktion
- Konstruierte Werte
- Ablagecode (Rückgabewerte)
- Arrays
- Ausnahmen
- Datentypzuordnungen für einfache Datentypen

Die Funktion `getLanguageSignatures()` gibt eine Zuordnung dieser Signaturdeklarationen zurück. Entwickler von Erweiterungen können mithilfe dieser Zuordnung sprachspezifische Codeblöcke generieren, die Dreamweaver dann je nach Servermodell auf der Seite einfügt, wenn der Benutzer Elemente (wie z. B. eine Methode für Webdienste) durch Ziehen und Ablegen verschiebt.

Beispiele zum Schreiben dieser Funktion finden Sie in den HTML-Implementierungsdateien für die Servermodelle JSP und ASP.Net. Implementierungsdateien für Servermodelle sind im Ordner **Configuration\ServerModels** abgelegt.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Objekt, das die Skriptsprachen-Signaturen definiert. Dieses Objekt ordnet die generischen Signaturen sprachspezifischen Signaturen zu.

getServerExtension()

Verfügbarkeit

Dreamweaver UltraDev 4; von der Verwendung in Dreamweaver MX wird abgeraten

Beschreibung

Diese Funktion gibt die Standarddateierweiterung von Dateien zurück, die das aktuelle Servermodell verwenden. Das Objekt `serverModel` wird auf das Servermodell der aktuell ausgewählten Site gesetzt, wenn derzeit kein Benutzerdokument ausgewählt ist.

Argumente

Keiner

Rückgabewerte

Dreamweaver erwartet einen String, der die unterstützten Dateinamenerweiterungen zurückgibt.

getServerInfo()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt ein JavaScript-Objekt zurück, das über den JavaScript-Code aufgerufen werden kann. Sie können dieses Objekt über die JavaScript-Funktion `dom.serverModel.getServerInfo()` aufrufen. `serverName`, `serverLanguage` und `serverVersion` sind besondere Eigenschaften, die Sie über folgende JavaScript-Funktionen aufrufen können:

```
dom.serverModel.getServerName()
dom.serverModel.getServerLanguage()
dom.serverModel.getServerVersion()
```

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Objekt, das die Eigenschaften Ihres Servermodells enthält.

Beispiel

```
var obj = new Object();
obj.serverName = "ASP";
obj.serverLanguage = "JavaScript";
obj.serverVersion = "2.0";
...
return obj;
```

getServerLanguages()

Verfügbarkeit

Dreamweaver UltraDev 1, nicht empfohlen für Dreamweaver MX

Beschreibung

Diese Funktion gibt die unterstützten Skriptsprachen eines Servermodells mit einem String-Array zurück. Dreamweaver füllt mit diesen Strings die Liste **Standard-Skriptsprache** auf, die in der Kategorie **Anwendungsserver-Infos** des Dialogfelds **Site-Definition** zu finden ist.

***Hinweis:** Die Liste **Standard-Skriptsprache** steht nur in Dreamweaver 4 und früheren Versionen zur Verfügung. In Dreamweaver MX und späteren Versionen werden im Dialogfeld **Site-Definition** keine unterstützten Skriptsprachen angezeigt. Zudem ist die Funktion `getServerLanguages()` nicht mehr implementiert. Der Grund dafür ist, dass in Dreamweaver jedes Servermodell nur eine Serversprache unterstützt.*

In früheren Versionen von Dreamweaver konnte ein Servermodell mehrere Skriptsprachen unterstützen. So unterstützt beispielsweise das Servermodell ASP die Sprachen JavaScript und VBScript.

Wenn eine Datei im Ordner **ServerFormats** nur auf eine bestimmte Skriptsprache angewandt werden soll, fügen Sie die folgende Anweisung als erste Zeile in die HTML-Datei ein:

```
<!-- SCRIPTING-LANGUAGE=XXX -->
```

In diesem Beispiel steht `xxx` für die Skriptsprache. Diese Anweisung bewirkt, dass das Serververhalten nur dann im Plusmenü (+) des Bedienfelds **Serververhalten** angezeigt wird, wenn die aktuell ausgewählte Skriptsprache `xxx` ist.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array von Strings, die die unterstützten Skriptsprachen darstellen.

getServerModelExtDataNameUD4()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt die Servermodell-Implementierung zurück, mit der Dreamweaver die UltraDev4-Erweiterungsdateien im Ordner **Configuration\ExtensionData** aufrufen soll.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String, wie z. B. "ASP/JavaScript".

getServerModelDelimiters()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt die Skripttrennzeichen zurück, die vom Anwendungsserver verwendet werden. Zudem zeigt sie an, ob die einzelnen Zeichen zum Zusammenführen von Codeblocks verwendet werden können. Sie können auf diesen von JavaScript zurückgegebenen Wert zugreifen, indem Sie die Funktion `dom.serverModel.getDelimiters()` aufrufen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array von Objekten, von denen jedes die folgenden drei Eigenschaften enthält:

- Die Eigenschaft *startPattern* ist ein regulärer Ausdruck, der dem Anfangstrennzeichen des Skripts entspricht (z. B. `<*`).
- Die Eigenschaft *endPattern* ist ein regulärer Ausdruck, der dem Abschlusstrennzeichen des Skripts entspricht (z. B. `%>`).
- Die Eigenschaft *participateInMerge* ist ein Boolescher Wert, der festlegt, ob der zwischen Trennzeichen gesetzte Code für das Zusammenführen von Codeblöcken geeignet ist (`true`) oder nicht (`false`).

getServerModelDisplayName()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt den Namen zurück, der in der Benutzeroberfläche für dieses Servermodell angezeigt werden soll. Sie können diesen Wert aus JavaScript über die Funktion `dom.serverModel.getDisplayName()` aufrufen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String, wie z. B. "ASP JavaScript".

getServerModelFolderName()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt den Namen des Unterordners für dieses Servermodell im Ordner **Configuration** zurück. Sie können diesen von JavaScript zurückgegebenen Wert über die Funktion `dom.serverModel.getFolderName()` aufrufen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet einen String, wie z. B. "ASP_JS".

getServerSupportsCharset()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion gibt den Wert `true` zurück, wenn der aktuelle Server den angegebenen Zeichensatz unterstützt. Sie können über JavaScript ermitteln, ob das Servermodell einen bestimmten Zeichensatz unterstützt, indem Sie die Funktion `dom.serverModel.getServerSupportsCharset()` aufrufen.

Argumente

metaCharSetString

Das Argument *metaCharSetString* ist ein String, der den Wert für das Dokumentattribut "charset=" speichert.

Rückgabewerte

Dreamweaver erwartet einen Booleschen Wert.

getVersionArray()

Verfügbarkeit

Dreamweaver UltraDev 1, nicht empfohlen für Dreamweaver MX

Beschreibung

Diese Funktion ruft die Zuordnung von Servertechnologien zu bestimmten Versionsnummern ab. Diese Funktion wird durch die Funktion `dom.serverModel.getServerVersion()` aufgerufen.

Argumente

Keine

Rückgabewerte

Dreamweaver erwartet ein Array von Versionsobjekten, jedes davon mit Versionsname und -nummer, wie in folgenden Beispielen aufgeführt:

- ASP Version 2.0
- ADODB Version 2.1

Kapitel 22: Datenübersetzer

Datenübersetzer konvertieren spezielle Markup-Codes, z.B. serverbasierte Include-Dateien, konditionelle JavaScript-Anweisungen oder andere Codes (wie PHP3, JSP, CFML und ASP), in Code, der in Adobe Dreamweaver CS3 gelesen und angezeigt werden kann. In Dreamweaver können Sie Attribute innerhalb von Tag-Sätzen sowie ganze Tags und Codeblöcke übersetzen lassen. Alle Datenübersetzer, ob Block/Tag- oder Attribut-Übersetzer, sind HTML-Dateien.

Übersetzte Tags oder Codeblöcke müssen sich innerhalb gesperrter Bereiche befinden, damit das ursprüngliche Markup nicht geändert wird. Für übersetzte Attribute sind keine Sperren nötig. Dadurch wird das Untersuchen von Tags mit solchen Attributen zu einer einfachen Angelegenheit.

Bei der Übersetzung von Daten, vor allem bei ganzen Tags oder Codeblöcken, können komplexe Operationen anfallen, die sich entweder mit JavaScript nicht durchführen lassen oder effizienter in C durchgeführt werden könnten. Wenn Sie mit C oder C++ vertraut sind, sollten Sie auch den Abschnitt „C-Level-Erweiterbarkeit“ auf Seite 331 lesen.

In der folgenden Tabelle sehen Sie die Dateien zum Erstellen von Datenübersetzern:

Pfad	Datei	Beschreibung
Configuration/ThirdPartyTags/	<i>language.xml</i>	Enthält Informationen über Tags in der Markup-Sprache.
Configuration\ThirdPartyTags\	<i>language.gif</i>	Symbol für Tags in der Sprache.
Configuration/Translators/	<i>language.htm</i>	Enthält JavaScript-Funktionen für den Datenübersetzer.

Funktionsweise von Datenübersetzern

Dreamweaver behandelt alle Dateien auf die gleiche Weise. Dabei spielt es keine Rolle, ob die Übersetzer ganze Tags oder nur Attribute übersetzen. Beim Start liest Dreamweaver alle Dateien im Ordner **Configuration\Translators** und ruft die Funktion `getTranslatorInfo()` auf, um Informationen über den Übersetzer zu erhalten. Dateien, in denen die Funktion `getTranslatorInfo()` nicht vorhanden oder aufgrund eines Fehlers nicht definiert ist, werden von Dreamweaver ignoriert.

Hinweis: Damit der Startvorgang nicht durch JavaScript-Fehler gestört wird, werden Fehler in den Übersetzerdateien erst gemeldet, nachdem alle Übersetzer geladen wurden. Weitere Informationen zum Debuggen von Übersetzern finden Sie unter „Fehlersuche im Übersetzer“ auf Seite 319.

Darüber hinaus ruft Dreamweaver in allen relevanten Übersetzerdateien (die in den Übersetzungsvoreinstellungen angegeben sind) die Funktion `translateMarkup()` auf, sobald Anzeichen vorliegen, dass der Benutzer neuen Inhalt hinzugefügt oder vorhandenen Inhalt geändert hat, der nun übersetzt werden muss. Dreamweaver ruft die Funktion `translateMarkup()` auf, wenn der Benutzer:

- eine Datei in Dreamweaver öffnet,
- wieder in die Entwurfsansicht wechselt, nachdem er im Bedienfeld **HTML** oder in der Codeansicht Änderungen vorgenommen hat,
- die Eigenschaften eines Objekts im aktuellen Dokument ändert,
- ein Objekt einfügt (mithilfe des Bedienfelds **Objekte** oder über das Menü **Einfügen**),
- das aktuelle Dokument aktualisiert, nachdem er daran in einer anderen Anwendung Änderungen vorgenommen hat,
- dem Dokument eine Vorlage zuweist,
- ein Objekt mit Ziehen und Ablegen in das Dokumentfenster einfügt,
- Änderungen in einer abhängigen Datei speichert,
- eine Befehls-, Verhaltens-, Eigenschafteninspektor- oder sonstige Erweiterungsdatei aufruft, die die Eigenschaft `innerHTML` oder `outerHTML` eines Tag-Objekts oder die `data`-Eigenschaft eines Kommentarobjekts definiert,
- **Datei > Konvertieren > 3.0 Browser-kompatibel** wählt,

- **Modifizieren** > **Konvertieren** > **Tabellen in AP-Divs** wählt,
- **Modifizieren** > **Konvertieren** > **AP-Divs in Tabellen** wählt,
- ein Tag oder Attribut im Quick Tag Editor ändert und die Tabulator- bzw. Eingabetaste betätigt.

Welche Übersetzer sollen verwendet werden?

Alle Übersetzer müssen die Funktionen `getTranslatorInfo()` und `translateMarkup()` umfassen, und sie müssen sich im Ordner **Configuration/Translators** befinden. Sie unterscheiden sich jedoch hinsichtlich des Codes, den sie in das Benutzerdokument einfügen, und in der Art und Weise, in der der Code untersucht werden muss. Dies wird in der folgenden Liste beschrieben:

- Für kleinere Server-Markups, die Attributwerte festlegen oder einem HTML-Standardtag konditionelle Attribute hinzufügen, schreiben Sie einen Attributübersetzer. HTML-Standardtags mit übersetzten Attributen können mit den Eigenschafteninspektoren untersucht werden, die ein fester Bestandteil von Dreamweaver sind. Es ist nicht nötig, einen kundenspezifischen Eigenschafteninspektor zu schreiben (siehe „Übersetztes Attribut in ein Tag aufnehmen“ auf Seite 314).
- Zum Übersetzen von ganzen Tags (beispielsweise eines Server-Side Includes (SSI)) oder eines Codeblocks (beispielsweise JavaScript, ColdFusion, PHP oder andere Skripts) schreiben Sie einen Block/Tag-Übersetzer. Der von einem Block/Tag-Übersetzer generierte Code kann nicht mit den integrierten Eigenschafteninspektoren von Dreamweaver untersucht werden. Sie müssen für den übersetzten Inhalt einen benutzerspezifischen Eigenschafteninspektor schreiben, damit die Benutzer die Eigenschaften des Originalcodes ändern können (siehe „Übersetzte Tags und Codeblöcke sperren“ auf Seite 316.)

Übersetztes Attribut in ein Tag aufnehmen

Die Übersetzung von Attributen basiert auf der Fähigkeit des Dreamweaver-Parsers, Server-Markups zu ignorieren. Dreamweaver ignoriert standardmäßig bereits die gängigsten Server-Markups (einschließlich ASP, CFML und PHP). Wenn Sie ein Server-Markup mit anderen öffnenden und schließenden Marken verwenden, müssen Sie die Tag-Datenbank des Fremdanbieters so anpassen, dass der Übersetzer korrekt funktioniert. Weitere Informationen zum Anpassen der Tag-Datenbank von Fremdanbietern finden Sie unter „Dreamweaver anpassen“ im Handbuch *Dreamweaver verwenden*.

Beim Speichern des ursprünglichen Server-Markup durch Dreamweaver generiert der Übersetzer einen gültigen Attributwert, der im Dokumentfenster angezeigt werden kann. (Wenn Sie das Server-Markup nur für Attribute verwenden, die keinen für den Benutzer sichtbaren Effekt haben, benötigen Sie also keinen Übersetzer.)

Der Übersetzer erstellt ein Attribut, das einen sichtbaren Effekt im Dokumentfenster hat. Dazu wird dem Tag mit dem Server-Markup das spezielle Attribut `mmTranslatedValue` hinzugefügt. Das Attribut `mmTranslatedValue` und dessen Wert sind im HTML-Bedienfeld nicht sichtbar, sie werden auch nicht zusammen mit dem Dokument gespeichert.

Das Attribut `mmTranslatedValue` darf nur einmal innerhalb eines Tags verwendet werden. Wenn die Wahrscheinlichkeit besteht, dass der Übersetzer pro Tag mehr als ein Attribut übersetzen muss, müssen Sie eine Routine in den Übersetzer aufnehmen, die Nummern an `mmTranslatedValue` anhängt (beispielsweise `mmTranslatedValue1`, `mmTranslatedValue2` usw.).

Der Wert des Attributs `mmTranslatedValue` muss ein URL-kodierter String mit mindestens einem gültigen Attribut-Wert-Paar sein. Dies bedeutet, dass `mmTranslatedValue="src=%22open.jpg%22"` eine gültige Übersetzung sowohl für `src="<? if (dayType == weekday) then open.jpg else closed.jpg" ?>` als auch `<? if (dayType == weekday) then src="open.jpg" else src="closed.jpg" ?>` ist. `mmTranslatedValue="%22open.jpg%22"` ist für beide Beispiele nicht zulässig, da es nur den Wert, nicht aber das Attribut enthält.

Mehrere Attribute gleichzeitig übersetzen

`mmTranslatedValue` kann mehr als ein gültiges Attribut-Wert-Paar enthalten. Das lässt sich an dem folgenden, nicht übersetzten Code veranschaulichen:

```
 alt="We're open 24 hours a day from  
12:01am Monday until 11:59pm Friday">
```

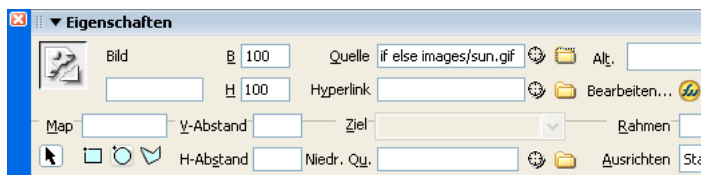
Nachfolgend ein Beispiel für einen übersetzten Markup:

```
  
mmTranslatedValue="src=%22open.jpg%22 width=%22320%22 height=%22100%22"  
alt="We're open 24 hours a day from 12:01am Monday until 11:59pm Friday">
```

Die Leerzeichen zwischen den Attribut-Wert-Paaren in `mmTranslatedValue` sind nicht kodiert. Da Dreamweaver nach diesen Leerzeichen sucht, wenn der übersetzte Wert dargestellt werden soll, muss jedes Attribut-Wert-Paar in `mmTranslatedValue` separat kodiert und anschließend zurückgegeben werden, um die Vollständigkeit von `mmTranslatedValue` zu gewährleisten. Ein Beispiel für diesen Vorgang finden Sie unter „Einfaches Übersetzungsbeispiel“ auf Seite 320.

Übersetzte Attribute untersuchen

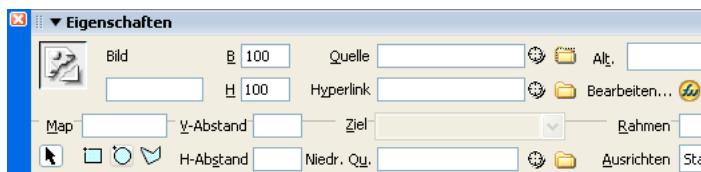
Wenn das Server-Markup ein einzelnes Attribut festlegt, das in einem Eigenschafteninspektor dargestellt wird, zeigt Dreamweaver das Markup im Eigenschafteninspektor an, wie in der folgenden Abbildung dargestellt:



Das Markup wird in jedem Fall angezeigt, auch wenn ihm kein Übersetzer zugeordnet ist. Der Übersetzer wird ausgeführt, sobald der Benutzer das im Eigenschafteninspektor angezeigte Markup bearbeitet.

Steuert ein Server-Markup mehrere Attribute in einem Tag, wird es nicht im Eigenschafteninspektor angezeigt. Dennoch informiert das Blitzsymbol den Benutzer, dass für das ausgewählte Element ein übersetztes Markup existiert.

Hinweis: Das Blitzsymbol wird nicht angezeigt, wenn Text, Tabellenzellen, Zeilen oder Spalten ausgewählt sind. Die Übersetzung wird dennoch durchgeführt, sofern der Benutzer das Server-Markup im Bedienfeld bearbeitet und ein für den entsprechenden Markup-Typ geeigneter Übersetzer vorhanden ist.



Die Felder im Eigenschafteninspektor sind bearbeitbar. Benutzer können Werte für Attribute eingeben, die vom Server-Markup gesteuert werden, sodass es zu Duplikaten von Attributen kommen kann. Wenn für ein bestimmtes Attribut sowohl ein übersetzter Wert als auch ein regulärer Wert eingestellt sind, zeigt Dreamweaver den übersetzten Wert im Dokumentfenster an. Sie müssen festlegen, ob der Übersetzer nach Duplikaten von Attributen suchen und sie entfernen soll.

Übersetzte Tags und Codeblöcke sperren

In den meisten Fällen soll ein Übersetzer das Markup ändern, sodass Dreamweaver dieses anzeigen kann. Dennoch sollte das Original-Markup – und nicht dessen Änderung – gespeichert werden. Zu diesem Zweck stellt Dreamweaver spezielle XML-Tags zur Verfügung, in die der übersetzte Inhalt eingeschlossen werden kann und die auf den Originalcode verweisen.

Wenn Sie diese XML-Tags verwenden, werden die Inhalte der ursprünglichen Attribute in der Codeansicht dupliziert. Beim Speichern der Datei wird das ursprüngliche, unübersetzte Markup in die Datei geschrieben. Dreamweaver zeigt in der Codeansicht den unübersetzten Inhalt an.

Nachfolgend ein Beispiel für die Syntax der XML-Tags:

```
<MM:BeginLock translatorClass="translatorClass" ↵
type="tagNameOrType" depFiles="dependentFilesList" ↵
orig="encodedOriginalMarkup">
Übersetzter Inhalt
<MM:EndLock>
```

Die Werte in diesem Beispiel haben folgende Bedeutung:

- Der Wert *translatorClass* ist die eindeutige ID für den Übersetzer. Es handelt sich dabei um den ersten String in dem Array, das von `getTranslatorInfo()` zurückgegeben wird.
- Der Wert *tagNameOrType* ist ein String, der den Markup-Typ innerhalb des gesperrten Bereichs angibt (oder den Tag-Namen, der zu dem Markup gehört). Der String darf nur alphanumerische Zeichen, Bindestriche (-) oder Unterstriche (_) enthalten. Sie können den Wert in der Funktion `canInspectSelection()` eines benutzerspezifischen Eigenschafteninspektors untersuchen und feststellen, ob sich der Übersetzer für den Inhalt eignet. Weitere Informationen finden Sie unter „Eigenschafteninspektoren für gesperrte Inhalte erstellen“ auf Seite 317. Gesperrter Inhalt kann nicht von den in Dreamweaver integrierten Eigenschafteninspektoren untersucht werden. So ist es beispielsweise nicht möglich, durch die Festlegung `type="IMG"` das Bedienfeld **Bild** aufzurufen.
- *dependentFilesList* ist ein String mit einer Liste durch Kommas getrennter Dateien, von denen das gesperrte Markup abhängt. Dateien werden als URLs referenziert, relativ zum Dokument des Benutzers. Wenn der Benutzer eine der Dateien im String *dependentFilesList* aktualisiert, übersetzt Dreamweaver erneut automatisch den Inhalt in dem Dokument, das die Liste enthält.
- *encodedOriginalMarkup* ist ein String mit dem ursprünglichen, nicht übersetzten Markup, das mit einem kleinen Teilsatz der URL-Kodierung codiert wurde (verwenden Sie %22 für ", %3C für <, %3E für > und %25 für %). Die schnellste Möglichkeit zur URL-Kodierung eines Strings bietet die Methode `escape()`. Wenn beispielsweise `myString` dem Verweis `` entspricht, gibt `escape(myString)` den Verweis `%3Cimg%20src=%22foo.gif%22%3E` zurück.

Das folgende Beispiel zeigt den gesperrten Codebereich, der bei der Übersetzung des Server-Side Include `<!--#include virtual="/footer.html" -->` generiert werden kann:

```
<MM:BeginLock translatorClass="MM_SSI" type="ssi" ↵
depFiles="C:\sites\webdev\footer.html" orig="%3C!--#include ↵
virtual=%22/footer.html%22%20--%3E">
<!-- Beginn Fußzeile -->
<CENTER>
<HR SIZE=1 NOSHADE WIDTH=100%>

<BR>

[<A TARGET="_top" HREF="/">home</A>]
[<A TARGET="_top" HREF="/products/">products</A>]
[<A TARGET="_top" HREF="/services/">services</A>]
[<A TARGET="_top" HREF="/support/">support</A>]
[<A TARGET="_top" HREF="/company/">about us</A>]
[<A TARGET="_top" HREF="/help/">help</A>]
</CENTER>
<!-- Ende Fußzeile -->
<MM:EndLock>
```

Das Erstellen eines Übersetzers, der Code innerhalb eines script-Tags sperrt, kann dazu führen, dass die Übersetzung fehlschlägt. Angenommen, Sie haben eine Site mit folgender Struktur:

```
<script language="javascript">
<!--
function foo() {
alert('<bean:message key="show.message"/>');
}
// -->
</script>
```

Anschließend erstellen Sie einen Übersetzer für das `bean:message-struts`-Tag. Das Ausführen des Übersetzungsprogramms wird zu einem Fehler führen, da Sie einen `MM:BeginLock`-Abschnitt innerhalb eines `MM:BeginLock`-Abschnitts erstellen. Sie können dies umgehen, indem Sie das Tag `bean:message`, das reguläre JSP-Tags wie `<%=My_lookup.lookup("show.message") %>` verwendet, mit einem JSP-Wrapper umschließen. Dadurch wird Ihr Übersetzer veranlasst, diesen Code zu überspringen, und die Übersetzung kann erfolgreich abgeschlossen werden.

Eigenschafteninspektoren für gesperrte Inhalte erstellen

Wenn Sie einen Übersetzer erstellt haben, müssen Sie auch einen Eigenschafteninspektor für den Inhalt anlegen, damit der Benutzer dessen Eigenschaften ändern kann (zum Beispiel die einzuschließende Datei oder eine der Bedingungen in einer bedingten Anweisung). Das Untersuchen von übersetztem Inhalt stellt ein sehr spezielles Problem dar, und zwar aus einer Reihe von Gründen:

- Der Benutzer möchte möglicherweise die Eigenschaften des übersetzten Inhalts ändern, und diese Änderungen müssen in den unübersetzten Inhalt umgesetzt werden.
- Das Dokumentobjektmodell (DOM) enthält den übersetzten Inhalt (d. h. die Sperr-Tags und die Tags, die sie umgeben, sind Knoten im DOM), aber die Eigenschaft `outerHTML` des Objekts `documentElement` und die Funktionen `dreamweaver.getSelection()` und `dreamweaver.nodeToOffsets()` wirken auf die nicht übersetzte Quelle.
- Die untersuchten Tags sind vor und nach dem Übersetzen verschieden.

Ein Eigenschafteninspektor für das Tag `HAPPY` könnte einen Kommentar ähnlich dem folgenden Code haben:

```
<!-- tag:HAPPY,priority:5,selection:exact,hline,vline, attrName:xxx,~ attrValue:yyy -->
```

Der Eigenschafteninspektor für den übersetzten `HAPPY`-Tag könnte jedoch einen Kommentar ähnlich dem folgenden Code haben:

```
<!-- tag:*LOCKED*,priority:5,selection:within,hline,vline -->
```

Die Funktion `canInspectSelection()` für den nicht übersetzten `HAPPY`-Eigenschafteninspektor ist einfach. Da der `selection`-Typ `exact` ist, kann er ohne weitere Analyse den Wert `true` zurückgeben. Für den Eigenschafteninspektor des übersetzten `HAPPY`-Tags ist die Funktion komplizierter: Das Schlüsselwort `*LOCKED*` gibt an, dass der Eigenschafteninspektor geeignet ist, wenn sich die Auswahl innerhalb eines gesperrten Bereichs befindet. Da ein Dokument aber mehrere gesperrte Bereiche enthalten kann, sind weitere Überprüfungen nötig, um festzustellen, ob der Eigenschafteninspektor diesem bestimmten gesperrten Bereich entspricht.

Es gibt noch ein weiteres Problem beim Untersuchen von übersetzten Inhalten. Wenn Sie die Funktion `dom.getSelection()` aufrufen, sind die standardmäßig zurückgegebenen Werte Offsets in die unübersetzte Quelle. Um die Auswahl korrekt zu erweitern, sodass der gesperrte Bereich (und nur dieser) ausgewählt wird, verwenden Sie die folgende Methode:

```
var currentDOM = dw.getDocumentDOM();
var offsets = currentDOM.getSelection();
var theSelection = currentDOM.offsetsToNode(offsets[0],offsets[0]+1);
```

Falls Sie als zweites Argument `offsets[0]+1` verwenden, ist sichergestellt, dass Sie innerhalb des Anfangs-Sperrtags bleiben, wenn Sie die Offsets in einen Knoten konvertieren. Wenn Sie als zweites Argument `offsets[1]` verwenden, besteht die Gefahr, dass Sie den Knoten oberhalb der Sperre auswählen.

Nachdem Sie die Auswahl vorgenommen und sich vergewissert haben, dass für `nodeType` der Wert `node.ELEMENT_NODE` gilt, können Sie das `type`-Attribut überprüfen, um festzustellen, ob der gesperrte Bereich dem Eigenschafteninspektor entspricht:

```
if (theSelection.nodeType == node.ELEMENT_NODE && ~
theSelection.getAttribute('type') == 'happy'){
    return true;
}else{
    return false
}
```

Um die Textfelder des Eigenschafteninspektors für das übersetzte Tag auszufüllen, müssen Sie den Wert des Attributs `orig` analysieren. Wenn der unübersetzte Code beispielsweise `<HAPPY TIME="22">` lautet und der Eigenschafteninspektor über das Textfeld „Time“ verfügt, müssen sie den Wert des Attributs `TIME` aus dem String `orig` extrahieren.

```
function inspectSelection() {
    var currentDOM = dw.getDocumentDOM();
    var currSelection = currentDOM.getSelection();
    var theObj = currentDOM.offsetsToNode(currSelection[0], currSelection[0]+1);

    if (theObj.nodeType != Node.ELEMENT_NODE) {
        return;
    }

    // Um die kodierten Zeichen wieder in ihre ursprünglichen
    // Werte zurückzukodieren, verwenden Sie die Methode unescape().
    var origAtt = unescape(theObj.getAttribute("ORIG"));

    // Den String für die Verarbeitung in Kleinbuchstaben konvertieren
    var origAttLC = origAtt.toLowerCase();

    var timeStart = origAttLC.indexOf('time="');
    var timeEnd = origAttLC.indexOf('"', timeStart+6);
    var timeValue = origAtt.substring(timeStart+6, timeEnd);

    document.layers['timelayer'].document.timeForm.timefield.value = timeValue;
}
```

Sobald Sie das Attribut `orig` analysiert haben, um die Felder des Eigenschafteninspektors für das übersetzte Tag auszufüllen, sollten Sie den Wert für das Attribut `orig` setzen, falls der Benutzer den Wert in einem der Textfelder ändert. Möglicherweise gelten aber Einschränkungen für das Vornehmen von Änderungen in einem gesperrten Bereich. Dieses Problem lässt sich umgehen, indem Sie das Original-Markup ändern und neu übersetzen.

Der Eigenschafteninspektor für übersetzte Server-Side Includes (die Datei `ssi_translated.js` im Ordner **Configuration/Inspectors**) demonstriert diese Vorgehensweise mit der Funktion `setComment()`. Statt das Attribut `orig` umzuschreiben, assembliert der Inspektor einen neuen SSI-Kommentar. Anschließend wird der Kommentar anstelle des alten Kommentars in das Dokument eingefügt. Dabei wird der gesamte Dokumentinhalt umgeschrieben, sodass das Attribut `orig` neu generiert wird. Der folgende Code verdeutlicht diese Methode:

```
// Den neuen include-Kommentar assemblieren. radioStr und URL sind
// weiter vorne im Code definierte Variablen.
newInc = "<!--#include " + radioStr + "=" + "'" + URL + "'" + " -->";

// Den Inhalt des Dokuments abrufen.
var entireDocObj = dreamweaver.getDocumentDOM();
var docSrc = entireDocObj.documentElement.outerHTML;

// Alle vor und nach dem SSI-Kommentar speichern
// der SSI-Kommentar in den Variablen beforeSelStr und afterSelStr variables.
var beforeSelStr = docSrc.substring(0, currSelection[0] );
var afterSelStr = docSrc.substring(currSelection[1]);

// Den neuen Inhalt des Dokuments assemblieren
docSrc = beforeSelStr + newInc + afterSelStr;
```

```
// Setzen Sie das outerHTML des HTML-Tags (durch das
// Objekt documentElement repräsentiert) auf den neuen Inhalt
// und setzen Sie dann die Auswahl wieder auf die gesperrte
// Region, die den SSI-Kommentar umschließt.
entireDocObj.documentElement.outerHTML = docSrc;
entireDocObj.setSelection(curSelection[0], curSelection[0]+1);
```

Fehlersuche im Übersetzer

Wenn die Funktion `translateMarkup()` bestimmte Fehlertypen enthält, wird der Übersetzer zwar ordnungsgemäß geladen, aber nicht ausgeführt. Dennoch werden keine Fehlermeldungen ausgegeben. Durch diese Methode wird zwar die Stabilität von Dreamweaver erhöht, aber möglicherweise die Fehlersuche erschwert, beispielsweise wenn plötzlich ein kleiner Syntaxfehler in 20 oder 30 Codezeilen gefunden werden muss.

Kann Ihr Übersetzer nicht ausgeführt werden, können Sie ihn in einen Befehl umwandeln, um den Fehler zu beheben. Führen Sie dazu die folgenden Schritte aus:

1 Kopieren Sie den gesamten Inhalt der Übersetzerdatei in ein neues Dokument, und speichern Sie dieses im Unterordner **Configuration/Commands** des Dreamweaver-Anwendungsordners.

2 Fügen Sie oben im Dokument zwischen den `SCRIPT`-Tags die folgende Funktion ein:

```
function commandButtons(){
    return new Array( "OK","translateMarkup(dreamweaver.~
    getDocumentPath('document'), dreamweaver.getSiteRoot(), ~
    dreamweaver.getDocumentDOM().documentElement.outerHTML); ~
    window.close()", "Cancel", "window.close()");
}
```

3 Deaktivieren Sie am Ende der Funktion `translateMarkup()` die Zeile `return whateverTheReturnValueIs`, und ersetzen Sie sie durch `dreamweaver.getDocumentDOM().documentElement.outerHTML = whateverTheReturnValueIs`, wie im folgenden Beispiel dargestellt:

```
// Zurückgeben von theCode;
dreamweaver.getDocumentDOM().documentElement.outerHTML = theCode;
}
/* Ende von translateMarkup() */
```

4 Fügen Sie im Hauptteil des Dokuments das folgende Formular ohne Eingabefelder hinzu.

```
<body>
<form>
Hello.
</form>
</body>
```

5 Starten Sie Dreamweaver neu, und wählen Sie im Menü **Befehle** den gewünschten Übersetzerbefehl aus. Wenn Sie auf **OK** klicken, wird die Funktion `translateMarkup()` aufgerufen, die eine Übersetzung simuliert.

Sollte keine Fehlermeldung angezeigt werden und die Übersetzung weiterhin fehlschlagen, liegt vermutlich ein logischer Fehler im Code vor.

6 Tragen Sie die Anweisung `alert()` an strategischen Punkten der Funktion `translateMarkup()` ein (sodass die relevanten Verzweigungen erfasst werden), um die Variablen- und Eigenschaftenwerte an verschiedenen Punkten zu prüfen.

```
for (var i=0; i< foo.length; i++){
    alert("we're at the top of foo.length array, and the value of i is " + i);
    /* restliche Schleife */
}
```

7 Nach dem Hinzufügen der `alert()`-Anweisungen wählen Sie im Menü **Befehle** den gewünschten Befehl aus. Klicken Sie dann auf **Abbrechen**, und wählen Sie den Befehl erneut aus. Dadurch wird die Befehlsdatei neu geladen, und Ihre Änderungen werden wirksam.

Einfaches Übersetzungsbeispiel

Mit einem Beispiel können Sie die Attributübersetzung besser verstehen. Der nachfolgende Übersetzer ist das Markup „Pound Conditional“ (Poco). Dabei handelt es sich um eine mit ASP und PHP verwandte Syntax.

Führen Sie zum Erstellen des Attributübersetzers die folgenden Schritte aus:

- „Das tagspec-Tag erstellen“ auf Seite 320
- „Symbol erstellen“ auf Seite 320
- „Attributübersetzer erstellen“ auf Seite 320

Das tagspec-Tag erstellen

Beim Erstellen des Übersetzers legen Sie zunächst einen `tagspec`-Tag für das Markup „Poco“ an. Dadurch wird die Analyse unübersetzter Poco-Anweisungen durch Dreamweaver verhindert.

So erstellen Sie das tagspec-Tag:

- 1 Erstellen Sie eine neue, leere Datei.
- 2 Geben Sie Folgendes ein:

```
<tagsec tag_name="poco" start_string="<#" end_string="#>" ~  
detect_in_attribute="true" icon="poco.gif" icon_width="17" ~  
icon_height="15"></tagsec>
```

- 3 Speichern Sie die Datei unter dem Namen **poco.xml** im Ordner **Configuration/ThirdPartyTags**.

Ein Beispiel für das Erstellen des Tags `tagsec` finden Sie in der Datei **Tags.xml** im Ordner **Configuration\ThirdPartyTags**.

Symbol erstellen

Als Nächstes erstellen Sie das Symbol für Poco-Tags.

So erstellen Sie das Symbol:

- 1 Erstellen Sie eine Grafik der Größe 18 x 18 Pixel als Symbol für Poco-Tags.
- 2 Speichern Sie die Datei unter dem Namen **poco.gif** im Ordner **Configuration/ThirdPartyTags**.

Attributübersetzer erstellen

Sie erstellen eine HTML-Datei mit den für den Attributübersetzer erforderlichen Funktionen.

So erstellen Sie die HTML-Datei:

- 1 Erstellen Sie eine neue leere Datei.
- 2 Geben Sie Folgendes ein:

```

<html>
<head>
<title>Conditional Translator</title>
<meta http-equiv="Content-Type" content="text/html; charset=">
<script language="JavaScript">

/*****
 * Dieser Übersetzer verarbeitet die folgenden Anweisungssyntaxen: *
 * <# if (condition) then foo else bar #>                               *
 * <# if (condition) then att="foo" else att="bar" #>                   *
 * <# if (condition) then att1="foo" att2="jinkies"                     *
 * att3="jeepers" else att1="bar" att2="zoinks" #>                       *
 *                                                                       *
 * Er verarbeitet keine Anweisungen ohne else-Klausel.                 *
 *****/

var count = 1;

function translateMarkup(docNameStr, siteRootStr, inStr){
var count = 1;
// Zähler stellt eindeutige mmTranslatedValues sicher
var outStr = inStr;
// zu manipulierender String
var spacer = "";
// String für die Verarbeitung von Leerzeichen zwischen kodierten Attributen
var start = inStr.indexOf('<# if'); // Erste Instanz von Pound-Conditional-Code
// Deklariert, aber nicht initialisiert. //
var attAndValue;
// Boolescher Wert, der angibt, ob das Attribut Teil
// der bedingten Anweisung ist
var trueStart;
// Der Anfang des true-Falls
var falseStart;
// Der Anfang des false-Falls
var trueValue;
// Der auszugebende HTML-Code im true-Fall
var attName;
// Der Name des Attributs, das bedingt
// gesetzt worden ist.
var equalSign;
// Die Position des Gleichheitszeichens direkt
// links von <#, falls vorhanden
var transAtt;
// Das gesamte übersetzte Attribut
var transValue;
// Der Wert, der URL-kodiert werden muss
var back3FromStart;
// Drei Zeichen zurück von der Ausgangsposition
// (wird verwendet, um ein Gleichheitszeichen links von <# zu finden
var tokens;
// Ein Array mit allen im true-Fall gesetzten Attributen
var end;
// Das Ende der aktuellen bedingten Anweisung
// Solange es noch ein konditionelles <# gibt, das nicht
// übersetzt worden ist.

```

```

while (start != -1){
    back3FromStart = start-3;
    end = outStr.indexOf(' #>',start);
    equalSign = outStr.indexOf('= "<# if',back3FromStart);
    attAndValue = (equalSign != -1)?false:true;
    trueStart = outStr.indexOf('then', start);
    falseStart = outStr.indexOf(' else', start);
    trueValue = outStr.substring(trueStart+5, falseStart);
    tokens = dreamweaver.getTokens(trueValue, ' ');

    // Wenn attAndValue den Wert false annimmt, sollten Sie herausfinden,
    // welches Attribut Sie übersetzen, indem Sie vom Gleichheitszeichen
    // zum ersten Leerzeichen zurücksetzen. Der Teilstring zwischen dem Leerzeichen
    // und dem Gleichheitszeichen ist das Attribut
    if (!attAndValue){
        for (var i=equalSign; i > 0; i--){
            if (outStr.charAt(i) == " "){
                attName = outStr.substring(i+1,equalSign);
                break;
            }
        }
        transValue = attName + '=' + trueValue + '';
        transAtt = ' mmTranslatedValue' + count + '=' + ~
        escape(transValue) + '';
        outStr = outStr.substring(0,end+4) + transAtt + ~
        outStr.substring(end+4);
    }
    // Hat attAndValue den Wert true und ist tokens größer als
    // 1, so entspricht trueValue einer Reihe von Attribut-Wert-
    // Paaren und nicht einem einzelnen Wert. In diesem Fall muss jedes Attribut-Wert-
    // Paar separat kodiert und dann wieder zurückaddiert werden,
    // um den übersetzten Wert zu erhalten.
    }else if (tokens.length > 1){
        transAtt = ' mmTranslatedValue' + count + '='
        for (var j=0; j < tokens.length; j++){
            tokens[j] = escape(tokens[j]);
            if (j>0){
                spacer=" ";
            }
            transAtt += spacer + tokens[j];
        }
        transAtt += '';
        outStr = outStr.substring(0,end+3) + transAtt + ~
        outStr.substring(end+3);
    }

    // Hat attAndValue den Wert true und tokens ist nicht größer
    // als 1, so ist trueValue ein einzelnes Attribut-Wert-Paar.
    // Dies ist der einfachste Fall, bei dem nur noch
    // trueValue kodiert werden muss.
    }else{
        transValue = trueValue;
        transAtt = ' mmTranslatedValue' + count + '=' + ~
        escape(transValue) + '';
        outStr = outStr.substring(0,end+3) + transAtt + ~
        outStr.substring(end+3);
    }
}

```



```

        // Setzt den Zähler hoch, so dass die nächste Instanz
        // von mmTranslatedValue einen eindeutigen Namen haben wird. Dann
        // wird das nächste konditionelle <# im Code gesucht.
        count++;
        start = outStr.indexOf('<# if',end);
    }
    // Gibt den übersetzten String zurück.
    return outStr
}
function getTranslatorInfo(){
    returnArray = new Array(7);

    returnArray[0] = "Pound_Conditional";           // Die Übersetzerklasse
    returnArray[1] = "Pound Conditional Translator"; // Der Titel
    returnArray[2] = "2";                           // Die Anzahl der Erweiterungen
    returnArray[3] = "html";                         // Die erste Erweiterung
    returnArray[4] = "htm";                          // Die zweite Erweiterung
    returnArray[5] = "1";                           // Die Anzahl der Ausdrücke
    returnArray[6] = "<#";                          // Der erste Ausdruck
    returnArray[7] = "byString";                     //
    returnArray[8] = "50";                           //

    return returnArray
}
</script>
</head>

<body>
</body>
</html>

```

3 Speichern Sie die Datei unter dem Namen **Poco.htm** im Ordner **Configuration/Translators**.

Einfaches Beispiel für einen Block/Tag-Übersetzer

Das Konzept der Übersetzung kann mit einem Übersetzer deutlicher gemacht werden, der ausschließlich in JavaScript geschrieben ist und bei dem keine C-Bibliothek für Funktionen irgendwelcher Art herangezogen wird. Der folgende Übersetzer wäre in C effizienter, aber die JavaScript-Version ist einfacher und eignet sich daher besser für Demonstrationszwecke.

Wie die meisten Übersetzer dient auch dieser dazu, Serververhalten zu imitieren. Das Beispiel geht davon aus, dass der Webserver so konfiguriert ist, dass er das Tag `KENT` durch das Bild eines jeweils anderen Ingenieurs ersetzt – je nach Wochentag, Uhrzeit und Plattform des Benutzers. Der Übersetzer tut das gleiche, allerdings lokal.

So erstellen Sie den Block/Tag-Übersetzer:

- 1** Erstellen Sie eine neue leere Datei.
- 2** Geben Sie den folgenden Code ein:

```

<html>
<head>
<title>Kent Tag Translator</title>
<meta http-equiv="Content-Type" content="text/html; charset=">
<script language="JavaScript">

```

```

/*****
 * Die Funktion getTranslatorInfo() liefert Informationen *
 * über den Übersetzer (wie Klasse und Name), über die *
 * Dokumenttypen, die vermutlich das zu übersetzende *
 * Markup enthalten und über die regulären Ausdrücke, die *
 * dem Dokument mit dem zu übersetzenden Markup *
 * entsprechen (ob der Übersetzer auf alle Dateien, keine *
 * Dateien, Dateien mit den festgelegten Erweiterungen, *
 * oder Dateien, die mit den festgelegten Ausdrücken *
 * übereinstimmen, ausgeführt werden soll). *
 *****/
function getTranslatorInfo(){
    //Erstellen eines Arrays mit 6 Werten
    returnArray = new Array(6);

    returnArray[0] = "DREAMWEAVER_TEAM";           // Die Übersetzerklasse
    returnArray[1] = "Kent Tags";                  // Der Titel
    returnArray[2] = "0";                          // Die Anzahl der Erweiterungen
    returnArray[3] = "1";                          // Die Anzahl der Ausdrücke
    returnArray[4] = "<kent";                       // Ausdruck
    returnArray[5] = "byExpression";               // ausführen, wenn die Datei "<kent" enthält
    return returnArray;
}

/*****
 * Die Funktion translateMarkup() führt die eigentliche Übersetzung durch. *
 * In diesem Übersetzer wird die Funktion vollständig in *
 * JavaScript geschrieben (d.h. er verlässt sich auf keine C-Bibliothek) -- *
 * und ist extrem ineffizient. Es ist jedoch ein einfaches Beispiel, *
 * aus dem man gut lernen kann. *
 *****/
function translateMarkup(docNameStr, siteRootStr, inStr){
    var outStr = "";                               // Der nach der Übersetzung auszugebende String
    var start = inStr.indexOf('<kent>');           // Die erste Position des KENT-Tags
                                                    // im Dokument.

    var replCode = replaceKentTag();               // Ruft die Funktion replaceKentTag() auf,
                                                    // die den Code abzuruft, der KENT ersetzen wird.

    var outStr = "";                               // Der nach der Übersetzung auszugebende String

    //Besitzt das Dokument keinen Inhalt, wird die Übersetzung beendet.
    if ( inStr.length <= 0 ){
        return "";
    }

    // Wenn start, was der Position des KENT-Tags in inStr entspricht,
    // ungleich -1 ist (also wenn es ein weiteres
    // KENT-Tag im Dokument gibt)
    while (start != -1){
        // Kopiere alles bis zum Start des KENT-Tags.
        // Das ist sehr wichtig, da Übersetzer niemals etwas anderes übersetzen sollten
        // als das zu übersetzende Markup.
        outStr = inStr.substring(0, start);
        // Ersetze das KENT-Tag durch den übersetzten HTML-Code, der in spezielle
        // Sperr-Tags eingeschlossen ist. Weitere Informationen über die Ersetzungsoperation erhalten Sie
        // in den Kommentaren zur Funktion replaceKentTag()

```

```

outStr = outStr + replCode;

// Kopiere alles nach dem KENT-Tag.
outStr = outStr + inStr.substring(start+6);

// Verwende den soeben erstellten String für den nächsten Durchgang
// durch das Dokument. Das ist der ineffizienteste Teil des Ganzen.
inStr = outStr;
start = inStr.indexOf('<kent>');

}
// Sind im Dokument keine weiteren KENT-Tags vorhanden, wird outStr zurückgegeben.
return outStr;
}

/*****
* Die Funktion replaceKentTag() assembliert den HTML-Code, der *
* das KENT-Tag ersetzen wird, und die speziellen Sperr-Tags, die den *
* HTML-Code umschließen werden. Es wird die Funktion getImage() aufgerufen, um *
* das SRC des IMG-Tags zu ermitteln. *
*****/
function replaceKentTag(){
    // Das anzuzeigende Bild.
    var image = getImage();
    // Der Speicherort des Bilds auf der lokalen Festplatte.
    var depFiles = dreamweaver.getSiteRoot() + image;
    // Das IMG-Tag, das zwischen die Sperr-Tags eingefügt werden wird.
    var imgTag = '<IMG SRC='/' + image + ' " WIDTH="320" HEIGHT="240" ALT="Kent">\n';
    // Erster Teil des öffnenden Sperr-Tags. Der Rest des Tags wird weiter unten
    // assembliert.
    var start = '<MM:BeginLock translatorClass="DREAMWEAVER_TEAM" type="kent"';
    // Das schließende Sperr-Tag.
    var end = '<MM:EndLock>';

    //Assemblieren der Sperr-Tags und des ersetzenden HTML-Codes.
    var replCode = start + ' depFiles="' + depFiles + '"';
    replCode = replCode + ' orig="%3Ckent%3E">\n';
    replCode = replCode + imgTag;
    replCode = replCode + end;

    return replCode;
}

/*****
* Die Funktion getImage() bestimmt, welches Bild angezeigt werden soll *
* auf der Basis des Wochentags, der Uhrzeit und der *
* Benutzerplattform. Datum und Zeit basieren auf der Weltzeit UTC *
* (Greenwich Mean Time) minus 8 Stunden, also *
* pazifische Standardzeit (PST). Die Zeitumstellung *
* wird in dieser Routine nicht berücksichtigt. *
*****/
function getImage(){
    var today = new Date(); // Das aktuelle Datum und die Uhrzeit.
    var day = today.getUTCDay(); // Der Wochentag in der Zeitzone GMT.
    // 0=Sonntag, 1=Montag, usw.

```

```

        var hour = today.getUTCHours();           // Die aktuelle Stunde in GMT im
                                                    // 24-Stunden-Format.
var SFhour = hour - 8;                           // Die Uhrzeit in San Francisco im
                                                    // 24-Stunden-Format.
var platform = navigator.platform;               // Benutzerplattform. Alle Windows-Computer
                                                    // werden von Dreamweaver als "Win32" identifiziert,
                                                    // alle Macs als "MacPPC".
var imageRef;                                   // Die zurückzugebende Bildreferenz.
    // Ist SFhour ein negativer Wert, müssen Sie zwei Anpassungen vornehmen.
    // Zunächst subtrahieren Sie 1 vom Tageszähler, da in GMT bereits die frühen
    // Morgenstunden des nächsten Tages angebrochen sind. Dann addieren Sie SFhour zu 24, um
    // eine gültige Zeitangabe im 24-Stunden-Format zu erhalten.
if (SFhour < 0){
    day = day - 1;
    // Der Tag um 1 zurückgezählt würde einen negativen Wert ergeben, aber es ist Samstag,
    // setzen Sie daher den Wert auf 6.
    if (day < 0){
        day = 6;
    }
    SFhour = SFhour + 24;
}

// Nun legen Sie fest, welches Foto angezeigt werden soll, abhängig davon, ob es sich um einen
Arbeitsstag oder das
// Wochenende handelt, welche Zeit es ist, und ob es sich um eine Uhrzeit und einen Tag handelt, an
dem Kent
// arbeitet und auf welcher Plattform der Benutzer arbeitet.

//Wenn es nicht Sonntag ist
if (day != 0){
    ///Und wenn es zwischen 10 Uhr morgens und Mittag einschl. ist
    if (SFhour >= 10 && SFhour <= 12){
        imageRef = "images/kent_tiredAndIrritated.jpg";
        //Wenn es zwischen 13 und 15 Uhr einschl. ist
    }else if (SFhour >= 13 && SFhour <= 15){
        imageRef = "images/kent_hungry.jpg";
        //Oder wenn es zwischen 16 und 17 Uhr einschl. ist
    }else if (SFhour >= 16 && SFhour <= 17){
        //Wenn der Benutzer am Mac arbeitet, wird ein Bild von Kent am Mac angezeigt
        if (platform == "MacPPC"){
            imageRef = "images/kent_gettingStartedOnMac.jpg";
            //Wenn der Benutzer mit Windows arbeitet, wird ein Bild von Kent am PC angezeigt
        }else{
            imageRef = "images/kent_gettingStartedOnWin.jpg";
        }
    }
    //Oder wenn es nach 18 Uhr, aber vor Mitternacht ist
    }else if (SFhour >= 18){
        //Wenn es Samstag ist
        if (day == 6){
            imageRef = "images/kent_dancing.jpg";
            //Wenn es nicht Samstag ist, wird die Benutzerplattform geprüft
        }else if (platform == "MacPPC"){
            imageRef = "images/kent_hardAtWorkOnMac.jpg";
        }
    }
}

```

```

        }else{
            imageRef = "images/kent_hardAtWorkOnWin.jpg";
        }
    }else{
        imageRef = "images/kent_sleeping.jpg";
    }
    //Wenn es nach Mitternacht und vor 10 Uhr ist oder zu irgendeinem sonstigen Zeitpunkt am Sonntag
    }else{
        imageRef = "images/kent_sleeping.jpg";
    }
    return imageRef;
}

</script>
</head>

<body>
</body>
</html>

```

3 Speichern Sie die Datei unter dem Namen **kent.htm** im Ordner **Configuration/Translators**.

Die API-Funktionen für Datenübersetzer

In diesem Abschnitt werden die zur Definition von Übersetzern für Dreamweaver verwendeten Funktionen beschrieben.

getTranslatorInfo()

Beschreibung

Diese Funktion liefert Informationen über den Übersetzer und über die Dateien, auf die der Übersetzer angewendet werden kann.

Argumente

Keine

Rückgabewerte

Ein String-Array. Die Elemente des Arrays müssen in der folgenden Reihenfolge vorliegen:

- 1 Der String *translatorClass* kennzeichnet den Übersetzer eindeutig. Dieser String muss mit einem Buchstaben beginnen und darf nur alphanumerische Zeichen, Bindestriche (-) und Unterstriche (_) enthalten.
- 2 Der String *title* beschreibt den Übersetzer mit max. 40 Zeichen.
- 3 *_ nExtensions* gibt die Anzahl der nachfolgenden Dateinamenserweiterungen an. Ist *nExtensions* gleich 0, kann der Übersetzer auf jede Datei angewendet werden. Ist *nExtensions* gleich 0, stellt *nRegExps* das nächste Element des Arrays dar.
- 4 *extension* gibt die Dateinamenserweiterung der Dateien an (zum Beispiel "htm" oder "SHTML"), auf die der betreffende Übersetzer angewendet werden kann. Bei diesem String muss nicht zwischen Groß- und Kleinschreibung unterschieden werden und er darf nicht mit einem Punkt beginnen. Das Array muss die gleiche Anzahl von *extension*-Elementen enthalten, wie in *nExtensions* angegeben sind.
- 5 *nRegExps* gibt die Anzahl der nachfolgenden regulären Ausdrücke an. Ist *nRegExps* gleich 0, dann ist *runDefault* das nächste Element im Array.
- 6 The *regExps* legt einen regulären Ausdruck fest, den Sie überprüfen können. Das Array muss die gleiche Anzahl an *regExps*-Elementen enthalten, wie in *nRegExps* angegeben, und mindestens ein *regExps*-Element muss einem Codebereich in der HTML-Quelle des Dokuments entsprechen, damit der Übersetzer eine Datei bearbeiten kann.

7 `runDefault` legt den Ausführungszeitpunkt des Übersetzers fest. Nachfolgend sind die möglichen Werte aufgeführt:

String	Definition
"allFiles"	Der Übersetzer wird immer ausgeführt.
"noFiles"	Der Übersetzer wird nie ausgeführt.
"byExtension"	Der Übersetzer wird für die Dateien mit den in der Erweiterung festgelegten Dateinamenserweiterungen ausgeführt.
"byExpression"	Der Übersetzer wird ausgeführt, wenn der Ausdruck im Dokument mit einem der festgelegten regulären Ausdrücke übereinstimmt.
"bystring"	Der Übersetzer wird ausgeführt, wenn das Dokument einen der festgelegten Strings enthält.

Hinweis: Wenn Sie `runDefault` auf "byExtension" setzen, aber keine Erweiterungen angeben (siehe Schritt 4), ist das Ergebnis dasselbe wie bei der Einstellung "allFiles". Wenn Sie `runDefault` auf "byExpression" setzen, aber keine Ausdrücke angeben (siehe Schritt 6), ist das Ergebnis dasselbe wie bei der Einstellung "noFiles".

8 `priority` gibt die Standardpriorität zum Ausführen dieses Übersetzers an. Die Priorität ist ein Wert zwischen 0 und 100. Wird keine Priorität angegeben, gilt der Standardwert 100. Die höchste Priorität ist 0 und die niedrigste 100. Werden auf ein Dokument mehrere Übersetzer angewendet, legt diese Einstellung die Reihenfolge der Anwendung fest. Der Übersetzer mit der höchsten Priorität wird zuerst angewendet. Haben mehrere Übersetzer dieselbe Priorität, werden sie von `translatorClass` in alphabetischer Reihenfolge angewendet.

Beispiel

Das folgende Beispiel für `getTranslatorInfo()` stellt Informationen über einen Übersetzer für Server-Side-Includes zur Verfügung:

```
function getTranslatorInfo(){
    var transArray = new Array(11);
    transArray[0] = "SSI";
    transArray[1] = "Server-Side Includes";
    transArray[2] = "4";
    transArray[3] = "htm";
    transArray[4] = "stm";
    transArray[5] = "html";
    transArray[6] = "shtml";
    transArray[7] = "2";
    transArray[8] = "<!--#include file";
    transArray[9] = "<!--#include virtual";
    transArray[10] = "byExtension";
    transArray[11] = "50";

    return transArray;
}
```

translateDOM()

Verfügbarkeit

Dreamweaver CS3.

Beschreibung

Dreamweaver führt zwei Übersetzungsvorgänge durch. Der erste durchschreitet alle Übersetzer, und ruft die `translateMarkup()`-Funktionen auf. Nachdem diese Funktionen aufgerufen sind, ruft der zweite Übersetzungsvorgang die `translateDOM()`-Funktionen auf. Das übergebene `dom` ist das zu übersetzende `dom`. Die einzigen Bearbeitungen, die während den zweiten Durchgang zugelassen sind, sind diejenigen, die sich mit übersetzten Attributen beschäftigen.

Argumente

dom, sourceStr

- Das Argument *dom* .
- Das Argument *sourceStr* ist derselbe String, der an `translateMarkup` übergeben wird. Er wird zu Referenzzwecken bereitgestellt, aber jegliche Übersetzung sollte mit dem Argument *dom* durchgeführt werden, und nicht mit dem Argument *sourceStr*.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
translateDOM( dom, sourceStr );           //kein Rückgabewert
```

Die folgende Instanz der Funktion `translateDOM()` verbirgt das Tag mit der ID `div1` im Dokument.

```
function translateDOM(dom, sourceStr){
    var div1 = dom.getAttributeById("div1");
    if (div1){
        div1.style.display = "none";
    }
}
```

translateMarkup()

Beschreibung

Diese Funktion führt die Übersetzung durch.

Argumente

docName, siteRoot, docContent

- *docName* ist ein String mit dem URL für das zu übersetzende Dokument im Format „file://“.
- *siteRoot* ist ein String mit dem URL im Format „file://“ für den Stamm der Site, die das zu übersetzende Dokument enthält. Befindet sich das Dokument außerhalb einer Site, ist dieser String unter Umständen leer.
- Das Argument *docContent* ist ein String, der den Inhalt des Dokuments enthält.

Rückgabewerte

Ein String, der das übersetzte Dokument enthält. Wurde nichts übersetzt, ist der String leer.

Beispiel

Das folgende Beispiel für die Funktion `translateMarkup()` ruft die C-Funktion `translateASP()` auf, die in einer dynamischen Bibliothek (DLL) (Windows) bzw. einer Codebibliothek (Macintosh) mit dem Namen **ASPTrans** enthalten ist:

```
function translateMarkup(docName, siteRoot, docContent){
    var translatedString = "";
    if (docContent.length > 0){
        translatedString = ASPTrans.translateASP(docName, siteRoot, ~
        docContent);
    }
    return translatedString;
}
```

Vollständig aus JavaScript bestehende Beispiele finden Sie unter „Einfaches Übersetzungsbeispiel“ auf Seite 320 sowie unter „Einfaches Beispiel für einen Block/Tag-Übersetzer“ auf Seite 323.

liveDataTranslateMarkup()

Verfügbarkeit

Dreamweaver UltraDev 1.

Beschreibung

Diese Funktion übersetzt Dokumente, wenn der Benutzer sich im Dialogfeld **Live Data** befindet. Wenn der Benutzer den Befehl **Ansicht > Live Data** wählt oder auf die Schaltfläche **Aktualisieren** klickt, ruft Dreamweaver anstelle von `translateMarkup()` die Funktion `liveDataTranslateMarkup()` auf.

Argumente

docName, siteRoot, docContent

- *docName* ist ein String mit der URL für das zu übersetzende Dokument im Format „file://“.
- *siteRoot* ist ein String mit dem URL im Format „file://“ für den Stamm der Site, die das zu übersetzende Dokument enthält. Befindet sich das Dokument außerhalb einer Site, ist dieser String unter Umständen leer.
- Das Argument *docContent* ist ein String, der den Inhalt des Dokuments enthält.

Rückgabewerte

Ein String, der das übersetzte Dokument enthält. Wurde nichts übersetzt, ist der String leer.

Beispiel

Das folgende Beispiel für die Funktion `liveDataTranslateMarkup()` ruft die C-Funktion `translateASP()` auf, die in einer DLL-Datei (Windows) bzw. in einer Codebibliothek (Macintosh) namens **ASPTrans** enthalten ist:

```
function liveDataTranslateMarkup(docName, siteRoot, docContent){
    var translatedString = "";
    if (docContent.length > 0){
        translatedString = ASPTrans.translateASP(docName, siteRoot, docContent);
    }
    return translatedString;
}
```


Kapitel 23: C-Level-Erweiterbarkeit

Aufgrund der C-Level-Erweiterbarkeit können Erweiterungsdateien für Adobe Dreamweaver CS3 mit einer Kombination aus JavaScript und benutzerdefiniertem C-Code implementiert werden. Sie definieren die Funktionen in C, fassen sie in einer DLL (Dynamic Link Library) oder einer gemeinsam genutzten Bibliothek zusammen, speichern diese im Unterordner **Configuration\JSExtensions** des Dreamweaver-Anwendungsordners und rufen die Java-Script-Funktionen über den integrierten JavaScript-Interpreter von Dreamweaver auf.

So könnten Sie beispielsweise ein Dreamweaver-Objekt definieren, das den Inhalt einer benutzerspezifischen Datei in das aktuelle Dokument einfügt. Da das Client-JavaScript keine Unterstützung für die Datei-Eingabe/Ausgabe (E/A) enthält, müssen Sie eine entsprechende Funktion in C schreiben.

Integration von C-Funktionen

Sie könnten mit dem folgenden HTML- und JavaScript-Code ein einfaches Objekt **Insert Text from File** erstellen. Die Funktion `objectTag()` ruft die C-Funktion `readContentsOfFile()` auf, die in der Bibliothek `myLibrary` gespeichert ist.

```
<HTML>
<HEAD>
<SCRIPT>
function objectTag() {
    fileName = document.forms[0].myFile.value;
    return myLibrary.readContentsOfFile(fileName);
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Den Namen der zu importierenden Datei eingeben:
<INPUT TYPE="file" NAME="myFile">
</FORM>
</BODY>
</HTML>
```

Die Funktion `readContentsOfFile()` übernimmt vom Benutzer eine Liste mit Argumenten, ruft das Argument mit dem Dateinamen ab, liest den Inhalt der Datei und gibt den Inhalt der Datei zurück. Weitere Informationen zu den JavaScript-Datenstrukturen und -funktionen von `readContentsOfFile()` finden Sie unter „C-Level-Erweiterbarkeit und der JavaScript-Interpreter“ auf Seite 333.

```

JSBool
readContentsOfFile(JSContext *cx, JSObject *obj, unsigned int argc, jsval *argv, jsval *rval)
{
    char *fileName, *fileContents;
    JSBool success;
    unsigned int length;

    /* Sicherstellen, dass die aufrufende Funktion genau ein Argument übergibt. Wenn nicht,
    * den Interpreter anweisen, die Skriptausführung zu unterbrechen.*/
    if (argc != 1){
        JS_ReportError(cx, "Wrong number of arguments", 0);
        return JS_FALSE;
    }

    /* Argument in String konvertieren */
    fileName = JS_ValueToString(cx, argv[0], &length);
    if (fileName == NULL){
        JS_ReportError(cx, "The argument must be a string", 0);
        return JS_FALSE;
    }

    /* String (Dateinamen) zum Öffnen und Lesen einer Datei verwenden */
    fileContents = exerciseLeftToTheReader(fileName);

    /* Dateiinhalt in rval speichern, dabei handelt es sich um den
    * an die aufrufende Funktion zurückgegebenen Rückgabewert */
    success = JS_StringToValue(cx, fileContents, 0, *rval);
    free(fileContents);

    /* true zum Fortfahren zurückgeben oder false, um Skript abubrechen */
    return success;
}

```

Damit die Funktion `readContentsOfFile()` keinen JavaScript-Fehler auslöst, sondern wie beabsichtigt ausgeführt werden kann, müssen Sie die Funktion im JavaScript-Interpreter registrieren. Verwenden Sie dazu die Funktion `MM_Init()` in Ihrer Bibliothek. Wenn Dreamweaver die Bibliothek beim Starten lädt, wird die Funktion `MM_Init()` aufgerufen. Sie fragt drei Datenelemente ab:

- den JavaScript-Namen der Funktion
- einen Zeiger auf die Funktion
- die von der Funktion erwartete Anzahl von Argumenten

Nachfolgend ein Beispiel für die Funktion `MM_Init()` für `myLibrary`:

```

void
MM_Init()
{
    JS_DefineFunction("readContentsOfFile", readContentsOfFile, 1);
}

```

Ihre Bibliothek muss genau eine Instanz des folgenden Makros enthalten:

```

/* MM_STATE ist ein Makro, das auf einige Definitionen erweitert wird, die
 * für die Interaktion mit Dreamweaver gebraucht werden. Dieses Makro muss
 * genau einmal in Ihrer Bibliothek definiert werden. */
MM_STATE

```

Hinweis: Die Bibliothek kann entweder in C oder C++ implementiert werden. Die Datei, die die Funktion `MM_Init()` und das Makro `MM_STATE` enthält, muss jedoch in C implementiert werden. Der C++-Compiler verändert Funktionsnamen, sodass Dreamweaver die Funktion `MM_Init()` nicht finden kann.

C-Level-Erweiterbarkeit und der JavaScript-Interpreter

Es gibt drei Anlässe, bei denen der C-Code in Ihrer Bibliothek mit dem JavaScript-Interpreter von Dreamweaver interagieren muss:

- beim Start, wenn die Bibliotheksfunktionen registriert werden;
- beim Aufruf der Funktion, wenn die von JavaScript an C zu übergebenden Argumente analysiert werden;
- bevor die Funktion den Rückgabewert verpackt.

Zu diesem Zweck definiert der Interpreter mehrere Datentypen und stellt eine API bereit. Die Definitionen für die in diesem Abschnitt aufgeführten Datentypen und Funktionen befinden sich in der Datei **mm_jsapi.h**. Damit die Bibliothek ordnungsgemäß funktioniert, müssen Sie die Datei **mm_jsapi.h** mit der folgenden Zeile am Anfang einer jeden Datei in der Bibliothek einfügen:

```
#include "mm_jsapi.h"
```

Zusammen mit der Datei **mm_jsapi.h** wird die Datei **mm_jsapi_environment.h** aufgenommen, welche die **MM_Environment**-Struktur definiert.

Datentypen

Der JavaScript-Interpreter definiert folgende Datentypen.

typedef struct JSContext JSContext

An die C-Level-Funktion wird ein Zeiger für diesen unbekannten Datentyp übergeben. Einige API-Funktionen akzeptieren diesen Zeiger als Argument.

typedef struct JSObject JSObject

An die C-Level-Funktion wird ein Zeiger für diesen unbekannten Datentyp übergeben. Dieser Datentyp repräsentiert ein Objekt (Array-Objekt oder anderer Objekttyp).

typedef struct jsval jsval

Eine unbekannte Datenstruktur kann einen Integerwert oder einen Zeiger für eine Fließkommazahl, einen String oder ein Objekt enthalten. Einige API-Funktionen können zum Einlesen der Werte der Funktionsargumente verwendet werden (indem der Inhalt von **JSVAL** gelesen wird), während andere API-Funktionen zum Schreiben des Rückgabewerts dienen (durch Schreiben von **JSVAL**).

typedef enum { JS_FALSE = 0, JS_TRUE = 1 } JSBool

Ein einfacher Datentyp, der einen Booleschen Wert enthält.

Die C-Level-API

Die C-Level-Erweiterbarkeits-API umfasst folgende Funktionen:

typedef JSBool (*JSNative)(JSContext *cx, JSObject *obj, unsigned int argc, jsval *argv, jsval *rval)

Beschreibung

Diese Funktionssignatur beschreibt C-Level-Implementierungen von JavaScript-Funktionen in folgendem Kontext:

- *cx* ist ein Zeiger auf eine unbekannte *JSContext*-Struktur, der an einige Funktionen der JavaScript-API übergeben werden muss. Diese Variable enthält den Ausführungskontext des Interpreters.
- *obj* ist ein Zeiger auf das Objekt, in dessen Kontext das Skript ausgeführt wird. Während der Skriptausführung entspricht das Schlüsselwort *this* dem Objekt.
- *argc* ist die Anzahl der Argumente, die an die Funktion übergeben werden.
- Der Zeiger *argv* verweist auf ein Array von *JSVal*-Strukturen. Das Array hat eine Länge von *argc* Elementen.
- *rval* ist ein Zeiger auf eine einzelne *JSVal*-Struktur. Der Rückgabewert der Funktion sollte nach **rval* geschrieben werden.

Wenn erfolgreich, gibt die Funktion *JS_TRUE* zurück, andernfalls *JS_FALSE*. Wenn die Funktion den Wert *JS_FALSE* zurückgibt, wird die Ausführung des aktuellen Skripts gestoppt und eine Fehlermeldung angezeigt.

JSBool JS_DefineFunction()

Beschreibung

Diese Funktion registriert eine C-Level-Funktion mit dem JavaScript-Interpreter von Dreamweaver. Nachdem *JS_DefineFunction()* die C-Levelfunktion erfasst hat, die Sie im *call*-Argument angeben, können Sie sie in einem JavaScript-Skript aufrufen, indem Sie mit dem von Ihnen im *name*-Argument angegebenen Namen darauf verweisen. Bei dem Argument *name* ist die Groß-/Kleinschreibung zu beachten.

Meist wird diese Funktion über *MM_Init()* aufgerufen, das Dreamweaver beim Starten aktiviert.

Argumente

*char *name, JSNative call, unsigned int nargs*

- *name* ist der Name der Funktion, der in JavaScript verwendet werden soll.
- *call* ist ein Zeiger auf eine C-Level-Funktion. Die Funktion muss die gleichen Argumente wie *readContentsOfFile* akzeptieren und mittels *JSBool* auf eine erfolgreiche bzw. fehlerhafte Ausführung hinweisen.
- *nargs* ist die von der Funktion erwartete Zahl von Argumenten.

Rückgabewerte

Ein Boolescher Wert: *JS_TRUE* weist darauf hin, dass der Vorgang erfolgreich war. *JS_FALSE* weist darauf hin, dass der Vorgang fehlgeschlagen ist.

char *JS_ValueToString()

Beschreibung

Diese Funktion extrahiert ein Funktionsargument aus einer *JSVal*-Struktur, konvertiert es in einen String (sofern möglich) und gibt den konvertierten Wert zurück.

Hinweis: Sie sollten den zurückgegebenen Pufferzeiger nicht verändern, da sonst die Datenstrukturen des JavaScript-Interpreters beschädigt werden. Wenn Sie den String verändern möchten, müssen Sie die Zeichen in einen anderen Puffer kopieren und einen neuen JavaScript-String erstellen.

Argumente

`JSCContext *cx, jsval v, unsigned int *pLength`

- `cx` ist der unbekannte `JSCContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- `v` ist die `JSVal`-Struktur, aus der der String extrahiert werden soll.
- `pLength` ist ein Zeiger auf einen nicht Ganzzahlwert ohne Vorzeichen. Die Funktion setzt `*pLength` mit der Länge des Strings in Byte gleich.

Rückgabewerte

Ein Zeiger auf einen mit null abgeschlossenen String bei Erfolg bzw. auf `null` bei einem Fehler. Die Aufrufoutine darf diesen String nach der Bearbeitung nicht freigeben.

JSBool JS_ValueToInteger()

Beschreibung

Extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in einen Integerwert (sofern möglich), und gibt den konvertierten Wert zurück.

Argumente

`JSCContext *cx, jsval v, long *lp`

- `cx` ist der unbekannte `JSCContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- `v` ist die `JSVal`-Struktur, aus der der String extrahiert werden soll.
- `lp` ist ein Zeiger auf einen 4 Byte langen Ganzzahlwert. Diese Funktion speichert den konvertierten Wert in `*lp`.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToDouble()

Beschreibung

Diese Funktion extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in ein Doppelwort (sofern möglich) und gibt den konvertierten Wert zurück.

Argumente

`JSCContext *cx, jsval v, double *dp`

- `cx` ist der unbekannte `JSCContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- `v` ist die `JSVal`-Struktur, aus der das Doppelwort extrahiert werden soll.
- `dp` ist ein Zeiger auf ein 8 Byte langes Doppelwort. Diese Funktion speichert den konvertierten Wert in `*dp`.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToBoolean()

Beschreibung

Extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in einen Booleschen Wert (sofern möglich), und gibt den konvertierten Wert zurück.

Argumente

`JSContext *cx, jsval v, JSBool *bp`

- `cx` ist der unbekannte `JSContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- `v` ist die `JSVal`-Struktur, aus der der Boolesche Wert extrahiert werden soll.
- Das Argument `bp` ist ein Zeiger auf einen `JSBool`-Booleschen Wert. Diese Funktion speichert den konvertierten Wert in `*bp`.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToObject()

Beschreibung

Extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in ein Objekt (sofern möglich), und gibt den konvertierten Wert zurück. Sollte es sich bei dem Objekt um ein Array handeln, verwenden Sie `JS_GetArrayLength()` und `JS_GetElement()`, um den Inhalt zu lesen.

Argumente

`JSContext *cx, jsval v, JSObject **op`

- `cx` ist der unbekannte `JSContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- `v` ist die `JSVal`-Struktur, aus der das Objekt extrahiert werden soll.
- Das Argument `op` ist ein Zeiger auf einen `JSObject`-Zeiger. Diese Funktion speichert den konvertierten Wert in `*op`.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_StringToValue()

Beschreibung

Diese Funktion speichert einen Stringrückgabewert in einer `JSVal`-Struktur. Weist ein neues JavaScript-String-Objekt zu.

Argumente

`JSContext *cx, char *bytes, size_t sz, jsval *vp`

- `cx` ist der unbekannte `JSContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- Das Argument `bytes` ist der in einer `JSVal`-Struktur zu speichernde String. Die Stringdaten werden kopiert, sodass der aufrufende Code den String freigeben sollte, sobald er nicht mehr benötigt wird. Wenn die Größe des Strings nicht angegeben wird (siehe Argument `sz`), muss der Wert mit null-abgeschlossen werden.
- `sz` ist die Größe des Strings (in Byte). Wenn `sz` 0 ist, wird die Länge des mit null-abgeschlossenen Strings automatisch berechnet.
- Das Argument `vp` ist ein Zeiger auf die `JSVal`-Struktur, in die der Inhalt des Strings kopiert werden soll.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_DoubleToValue()

Beschreibung

Diese Funktion speichert einen Fließkommazahl-Rückgabewert in einer `JSVal`-Struktur.

Argumente

`JSContext *cx, double dv, jsval *vp`

- `cx` ist der unbekannte `JSContext`-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- Das Argument `dv` ist eine 8 Byte lange Fließkommazahl.
- Das Argument `vp` ist ein Zeiger auf die `JSVal`-Struktur, in die der Inhalt des Doppelworts kopiert werden soll.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSVal JS_BooleanToValue()

Beschreibung

Diese Funktion speichert einen Booleschen Rückgabewert in einer `JSVal`-Struktur.

Argumente

`JSBool bv`

- Das Argument `bv` ist ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

Rückgabewerte

Eine `JSVal`-Struktur mit dem Booleschen Wert, den Sie als Argument an die Funktion weitergegeben haben.

JSVal JS_IntegerToValue()

Beschreibung

Diese Funktion konvertiert einen langen Ganzzahlwert in eine `JSVal`-Struktur.

Argumente

`lv`

Das Argument `lv` ist der lange Ganzzahlwert, den Sie in eine `JSVal`-Struktur konvertieren möchten.

Rückgabewerte

Eine `JSVal`-Struktur mit der Ganzzahl, die Sie als Argument an die Funktion weitergegeben haben.

JSVal JS_ObjectToValue()

Beschreibung

Diese Funktion speichert einen Objekt-Rückgabewert in JSVal. Verwenden Sie JS_NewArrayObject(), um ein Array-Objekt zu erstellen, und JS_SetElement(), um dessen Inhalt zu definieren.

Argumente

JSObject *obj

Das Argument *obj* ist ein Zeiger auf das JSObject-Objekt, das Sie in eine JSVal-Struktur konvertieren möchten.

Rückgabewerte

Eine JSVal-Struktur mit dem Objekt, das Sie als Argument an die Funktion weitergegeben haben.

char *JS_ObjectType()

Beschreibung

Wenn eine Objektreferenz vorhanden ist, gibt JS_ObjectType() den Klassennamen des Objekts zurück. Für ein DOM-Objekt beispielsweise gibt die Funktion den Wert „Document“ zurück. Ist das Objekt ein Knoten im Dokument, gibt die Funktion den Wert „Element“ zurück. Für ein Array gibt die Funktion den Wert „Array“ zurück.

Hinweis: Sie sollten den zurückgegebenen Pufferzeiger nicht verändern, da sonst die Datenstrukturen des JavaScript-Interpreters beschädigt werden.

Argumente

JSObject *obj

In der Regel wird dieses Argument an JS_ValueToObject() übergeben und konvertiert.

Rückgabewerte

Ein Zeiger auf einen mit Null abgeschlossenen String. Dieser String sollte vom aufrufenden Code nach Abschluss nicht freigegeben werden.

JSObject *JS_NewArrayObject()

Beschreibung

Diese Funktion erstellt ein neues Objekt mit einem Array von JSVals.

Argumente

JSContext *cx, unsigned int length, jsval *v

- *cx* ist der unbekannte JSContext-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *length* ist die Anzahl der Elemente, die das Array enthalten kann.
- *v* ist ein optionaler Zeiger auf den im Array zu speichernden Wert jsvals. Wenn der Rückgabewert nicht null lautet, ist *v* ein Array mit *length*-Elementen. Wenn der Rückgabewert null lautet, ist der anfängliche Inhalt des Array-Objekts nicht definiert (und kann mittels JS_SetElement() gesetzt werden).

Rückgabewerte

Ein Zeiger auf ein neues Array-Objekt bzw. auf null bei einem Fehler.

long JS_GetArrayLength()

Beschreibung

Wenn ein Zeiger auf ein Array-Objekt weist, wird die Anzahl der Array-Elemente von dieser Funktion abgefragt.

Argumente

*JSContext *cx, JSObject *obj*

- *cx* ist der unbekannte *JSContext*-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *obj* ist ein Zeiger auf ein Array-Objekt.

Rückgabewerte

Die Anzahl der Array-Elemente. Beim Auftreten eines Fehlers wird -1 zurückgegeben.

JSBool JS_GetElement()

Beschreibung

Diese Funktion liest ein einzelnes Element aus einem Array-Objekt.

Argumente

*JSContext *cx, JSObject *obj, unsigned int index, jsval *v*

- *cx* ist der unbekannte *JSContext*-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *obj* ist ein Zeiger auf ein Array-Objekt.
- *index* ist ein Integerindex im Array. Das erste Element ist *index* 0, und das letzte Element ist *index* (*length* - 1).
- *v* ist ein Zeiger auf den Wert *jsval*, in den der Inhalt der *JSVal*-Struktur des Arrays kopiert werden soll.

Rückgabewerte

Ein Boolescher Wert: *JS_TRUE* weist darauf hin, dass der Vorgang erfolgreich war. *JS_FALSE* weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_SetElement()

Beschreibung

Diese Funktion schreibt ein einzelnes Element eines Array-Objekts.

Argumente

*JSContext *cx, JSObject *obj, unsigned int index, jsval *v*

- *cx* ist der unbekannte *JSContext*-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *obj* ist ein Zeiger auf ein Array-Objekt.
- *index* ist ein Integerindex im Array. Das erste Element ist *index* 0, und das letzte Element ist *index* (*length* - 1).
- *v* ist ein Zeiger auf die *JSVal*-Struktur, deren Inhalt in den Wert *JSVal* des Arrays kopiert werden soll.

Rückgabewerte

Ein Boolescher Wert: *JS_TRUE* weist darauf hin, dass der Vorgang erfolgreich war. *JS_FALSE* weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ExecuteScript()

Beschreibung

Diese Funktion kompiliert einen JavaScript-String und führt ihn aus. Falls das Skript einen Rückgabewert generiert, wird dieser in **rval* zurückgegeben.

Argumente

*JSContext *cx, JSObject *obj, char *script, unsigned int sz, JSVal *rval*

- *cx* ist der unbekannte *JSContext*-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *obj* ist ein Zeiger auf das Objekt, in dessen Kontext das Skript ausgeführt wird. Während der Skriptausführung entspricht das Schlüsselwort `this` dem Objekt. In der Regel handelt es sich dabei um den Zeiger *JSObject*, der an die JavaScript-Funktion übergeben wird.
- *script* ist ein String, der JavaScript-Code enthält. Wenn die Größe des Strings nicht angegeben wird (siehe Argument *sz*), muss der Wert mit null abgeschlossen werden.
- *sz* ist die Größe des Strings (in Byte). Wenn *sz* 0 ist, wird die Länge des mit null abgeschlossenen Strings automatisch berechnet.
- *rval* ist ein Zeiger auf eine einzelne *JSVal*-Struktur. Der Rückgabewert der Funktion wird in **rval* gespeichert.

Rückgabewerte

Ein Boolescher Wert: *JS_TRUE* weist darauf hin, dass der Vorgang erfolgreich war. *JS_FALSE* weist darauf hin, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ReportError()

Beschreibung

Diese Funktion beschreibt die Ursache für einen Skriptfehler. Rufen Sie diese Funktion vor der Rückgabe von *JS_FALSE* für einen Skriptfehler auf, um dem Benutzer Informationen über die Ursache der Nichtausführung des Skripts zu geben (Beispiel: „Falsche Anzahl von Argumenten“).

Argumente

*JSContext *cx, char *error, size_t sz*

- *cx* ist der unbekannte *JSContext*-Zeiger, der an die JavaScript-Funktion übergeben wurde.
- *error* ist ein String, der die Fehlermeldung enthält. Der String wird kopiert und sollte vom aufrufenden Code freigegeben werden, sobald er nicht mehr benötigt wird. Wenn die Größe des Strings nicht angegeben wird (siehe Argument *sz*), muss der Wert mit null abgeschlossen werden.
- *sz* ist die Größe des Strings (in Byte). Wenn *sz* 0 ist, wird die Länge des mit null abgeschlossenen Strings automatisch berechnet.

Rückgabewerte

Ein Boolescher Wert: *JS_TRUE* weist darauf hin, dass der Vorgang erfolgreich war. *JS_FALSE* weist darauf hin, dass der Vorgang fehlgeschlagen ist.

API für Dateizugriff- und Mehrbenutzerkonfiguration

Adobe empfiehlt, das Dateisystem immer mit der API für Dateizugriff- und Mehrbenutzerkonfiguration über C-Level-Erweiterungen aufzurufen. Bei Dateien, die keine Konfigurationsdateien sind, rufen die Funktionen die jeweilige Datei oder den jeweiligen Ordner auf.

Dreamweaver unterstützt Mehrbenutzerkonfigurationen für die Betriebssysteme Windows XP, Windows 2000 und Mac OS X.

In der Regel installieren Sie Dreamweaver in einem Ordner mit eingeschränktem Zugriff, z. B. unter Windows in **C:\Programme**. Im Ordner **Configuration** von Dreamweaver können daher nur Benutzer mit Administratorrechten Änderungen vornehmen. Damit auch Benutzer in Mehrbenutzersystemen eigene Konfigurationen anlegen und verwalten können, erstellt Dreamweaver einen separaten Konfigurationsordner für jeden Benutzer. Jedes Mal, wenn Dreamweaver oder eine JavaScript-Erweiterung in den Konfigurationsordner von Dreamweaver schreibt, verwendet Dreamweaver automatisch den Konfigurationsordner des Benutzers. Ein Benutzer kann so seine eigene Dreamweaver-Konfiguration festlegen, ohne dabei die Einstellungen der anderen Benutzer zu beeinträchtigen.

Dreamweaver erstellt den Benutzer-Konfigurationsordner an einem Speicherort, auf den der jeweilige Benutzer vollen Schreib- und Lesezugriff hat. Der Speicherort des Ordners **Configuration** auf dem Computer des Benutzers ist vom Betriebssystem abhängig.

Unter Windows 2000 und Windows XP:

```
<Laufwerk>:\Dokumente und Einstellungen\<Benutzername>\Anwendungsdaten\Adobe\~  
Dreamweaver 9\Configuration
```

***Hinweis:** In Windows XP befindet sich dieser Ordner möglicherweise in einem versteckten Ordner.*

Unter Mac OS X:

```
<Laufwerk>:Benutzer:<Benutzername>:Library:Application Support:Adobe:Dreamweaver 9:Configuration
```

Oft öffnen JavaScript-Erweiterungen Dateien und schreiben in den Konfigurationsordner. JavaScript-Erweiterungen können auf das Dateisystem über DWFile, MMNotes oder über die Weitergabe einer URL an `dreamweaver.getDocumentDOM()` zugreifen. Wenn eine Erweiterung auf das Dateisystem in einem Ordner **Configuration** zugreift, verwendet es dazu in der Regel `dw.getConfigurationPath()` und fügt den Dateinamen hinzu. Manchmal ruft es aber auch von `dom.URL` den Pfad eines geöffneten Dokuments ab und fügt dann den Dateinamen hinzu. Eine Erweiterung kann den Pfad auch über `dom.URL` abrufen und daraus den Dateinamen extrahieren. Die Funktionen `dw.getConfigurationPath()` und `dom.URL` geben immer eine URL im Ordner **Configuration** von Dreamweaver zurück, selbst wenn das Dokument im Ordner **Configuration** eines Benutzers abgelegt ist.

Immer wenn eine JavaScript-Erweiterung eine Datei im Dreamweaver-Ordner **Configuration** öffnet, unterbricht Dreamweaver den Zugriff und überprüft zuerst den Ordner **Configuration** des Benutzers. Wenn eine JavaScript-Erweiterung mithilfe von DWFile oder MMNotes Daten im Ordner **Configuration** von Dreamweaver speichert, fängt Dreamweaver den Aufruf ab und leitet ihn in den Ordner **Configuration** des Benutzers um.

Wenn ein Benutzer in Windows 2000 oder Windows XP die Datei **file:///C:/Programme/Adobe/Adobe Dreamweaver CS3/Configuration/Objects/Common/Table.htm** aufruft, sucht Dreamweaver stattdessen nach der Datei **Table.htm** im Ordner **C:\Dokumente und Einstellungen\Benutzername\adobe\Dreamweaver 9\Configuration\Objects\Common** und verwendet diese Datei, sofern sie vorhanden ist.

C-Level-Erweiterungen bzw. gemeinsame Bibliotheken müssen über eine API für Dateizugriff- und Mehrbenutzerkonfiguration Schreib- und Lesezugriff auf den Dreamweaver-Ordner **Configuration** herstellen. Mithilfe der API für Dateizugriff- und Mehrbenutzerkonfiguration kann Dreamweaver Lese- und Schreibzugriff auf den Konfigurationsordner des Benutzers herstellen. Darüber hinaus wird sichergestellt, dass Dateibearbeitungen nicht aufgrund fehlender Zugriffsberechtigungen fehlschlagen. Wenn Ihre C-Level-Erweiterung auf Dateien im Dreamweaver-Ordner **Configuration** zugreift, die über JavaScript mit DWFile-, MMNotes- oder DOM-Bearbeitungen erstellt wurden, müssen Sie unbedingt mit der API für Dateizugriff- und Mehrbenutzerkonfiguration arbeiten, da diese Dateien möglicherweise im Ordner **Configuration** des Benutzers abgelegt sind.

Hinweis: Die meisten JavaScript-Erweiterungen müssen nicht eigens angepasst werden, um Schreibzugriff auf den Ordner **Configuration** des Benutzers zu erhalten. Nur gemeinsam genutzte C-Bibliotheken, die in diesen Ordner schreiben, müssen aktualisiert werden, um die Funktionen der API für Dateizugriff- und Mehrbenutzerkonfiguration zu verwenden.

Wenn Sie eine Datei aus dem Dreamweaver-Ordner **Configuration** löschen, nimmt Dreamweaver einen Eintrag in eine Maskendatei auf, um anzuzeigen, welche Dateien im Ordner **Configuration** nicht in der Benutzeroberfläche angezeigt werden sollen. Maskendateien bzw. Maskenordner werden in Dreamweaver nicht angezeigt, auch wenn sie im Ordner vorhanden sind.

Wenn Sie z. B. im Bedienfeld **Codefragmente** den Ordner **javascript** und die Datei **onapixelborder.csn** löschen, schreibt Dreamweaver eine Datei namens **mm_deleted_files.xml** in den Ordner **Configuration** des Benutzers, die folgendermaßen aussieht:

```
<?xml version = "1.0" encoding="utf-8" ?>
  <deleteditems>
    <item name="snippets/javascript/" />
    <item name="snippets/html/onapixelborder.csn" />
  </deleteditems>
```

Wenn Dreamweaver Daten in das Bedienfeld **Codefragmente** übernimmt, werden alle Dateien im Ordner **Configuration\Snippets** des Benutzers und den gleichnamigen Ordnern in Dreamweaver gelesen. Ausgenommen sind der Ordner **Configuration\Snippets\javascript** und die Datei **Configuration\Snippets\html\onapixelborder.csn**. Die dabei erstellte Liste wird in das Bedienfeld **Codefragmente** übernommen.

Wenn eine C-Level-Erweiterung die Funktion `MM_ConfigFileExists()` für die URL `file:///c:/Programme/Adobe/Dreamweaver CS3/Configuration/Snippets/JavaScript/onapixelborder.csn` aufruft, lautet der Rückgabewert `false`. Versucht eine JavaScript-Erweiterung, die Funktion `dw.getDocumentDom("file:///c:/Programme/Adobe/Dreamweaver CS3/Configuration/Snippets/JavaScript/onapixelborder.csn")` aufzurufen, lautet der Rückgabewert `null`.

Durch Bearbeiten der Datei **mm_deleted_files.xml** können Sie verhindern, dass in Dreamweaver bestimmte Dateien in der Benutzeroberfläche angezeigt werden, wie z. B. Objekte, Standardinhalt eines neuen Dialogfelds usw. Sie können die Funktion `MM_DeleteConfigfile()` aufrufen, um Dateipfade zur Datei **mm_deleted_files.xml** hinzuzufügen.

JS_Object MM_GetConfigFolderList()

Verfügbarkeit

Dreamweaver MX

Beschreibung

Diese Funktion ruft für den angegebenen Ordner eine Liste mit Dateien und/oder Ordnern ab. Wenn Sie einen Konfigurationsordner angeben, ruft die Funktion eine Liste der Unterordner in den Ordnern **Configuration** des Benutzers und von Dreamweaver ab, die von der Datei **mm_deleted_files.xml** gefiltert werden.

Argumente

`char *fileURL, char *constraints`

- `char *fileURL` verweist auf den String mit dem Namen des Ordners, dessen Inhalt Sie abrufen. Der String muss dem URL-Format „file://“ entsprechen. Die Funktion akzeptiert als gültige Platzhalterzeichen Sternchen (*) und Fragezeichen (?) in dem URL-String „file://“. Mithilfe von Sternchen (*) können Sie ein oder mehrere unbekannte Zeichen und mit Fragezeichen (?) ein einzelnes unbekanntes Zeichen ausdrücken.
- Das Argument `char *constraints` kann `files` oder `directories` oder `null` sein. Wenn Sie `null` angeben, gibt die Funktion `MM_GetConfigFolderList()` sowohl Dateien als auch Ordner zurück.

Rückgabewerte

`JSObject` ist ein Array mit den Dateien oder Ordnern im Ordner **Configuration** des Benutzers oder von Dreamweaver, die von der Datei **mm_deleted_files.xml** gefiltert werden.

Beispiele

```
JSObject *jsobj_array;
jsobj_array = MM_GetConfigFolderList("file:///~
c:/Program Files/Adobe/Adobe Dreamweaver CS3/Configuration", "directories" );
```

JSBool MM_ConfigFileExists()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion überprüft, ob die angegebene Datei vorhanden ist. Befindet sich die Datei in einem Konfigurationsordner, überprüft die Funktion, ob sie im Ordner **Configuration** des Benutzers oder von Dreamweaver abgelegt ist. Die Funktion überprüft auch, ob der Dateiname in der Datei **mm_deleted_files.xml** aufgeführt wird. Ist dies der Fall, gibt die Funktion `false` zurück.

Argumente

`char *fileUrl`

`char *fileUrl` verweist auf den String für die gewünschte Datei im URL-Format „file://“.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

Beispiel

```
char *dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3/
Configuration/Extensions.txt";

int fileno = 0;
if (MM_ConfigFileExists(dwConfig))
{
    fileno = MM_OpenConfigFile(dwConfig, "read");
}
```

int MM_OpenConfigFile()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion öffnet die Datei und gibt ein Datei-Handle des Betriebssystems zurück. Sie können dieses Datei-Handle in Aufrufen an Betriebssystem-Funktionen verwenden. Sie müssen das Datei-Handle mit einem Aufruf an die Funktion `_close` des Systems schließen.

Wenn die Datei eine Konfigurationsdatei ist, sucht sie die Datei im Ordner **Configuration** des Benutzers oder von Dreamweaver. Wenn Sie die Konfigurationsdatei zum Schreiben öffnen, erstellt die Funktion die Datei im Ordner **Configuration** des Benutzers, auch wenn die Datei im Ordner **Configuration** von Dreamweaver abgelegt ist.

Hinweis: Möchten Sie die Datei vor dem Schreiben lesen, können Sie sie im Modus "read" öffnen. Zum Schreiben schließen Sie dann das read-Handle und öffnen die Datei im Modus "write" oder "append".

Argumente

*char *fileURL, char *mode*

- Das Argument *char *fileURL* verweist auf einen String mit dem Namen der Datei, die Sie öffnen, im URL-Format „file://“. Wird dabei ein Pfad im Dreamweaver-Ordner **Configuration** festgelegt, ermittelt `MM_OpenConfigFile()` den Pfad, bevor die Datei geöffnet wird.
- *char *mode* verweist auf einen String, der festlegt, wie Sie die Datei öffnen möchten. Sie können zwischen `null` oder dem Modus "read", "write" oder "append" wählen. Legen Sie "write" für eine Datei fest, die noch nicht angelegt wurde, wird sie von der Funktion `MM_OpenConfigFile()` erstellt. Legen Sie "write" fest, öffnet `MM_OpenConfigFile()` die Datei mit Exklusivzugriff. Legen Sie "read" fest, öffnet `MM_OpenConfigFile()` die Datei mit allgemeinem Zugriff.

Öffnen Sie die Datei im Modus "write", werden bereits vorhandene Daten darin gekürzt, bevor Sie in die Datei schreiben können. Öffnen Sie die Datei im Modus "append", werden alle neu von Ihnen geschriebenen Daten an das Dateiende angehängt.

Rückgabewerte

Eine Ganzzahl, die das Datei-Handle des Betriebssystems für diese Datei darstellt. Gibt -1 zurück, wenn die Datei nicht gefunden werden kann oder nicht existiert.

Beispiel

```
char *dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3/
    Configuration/Extensions.txt";
int = fileno;
if(MM_ConfigFileExists(dwConfig))
{
    fileno = MM_OpenConfigFile(dwConfig, "read");
}
```

JSBool MM_GetConfigFileAttributes()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion sucht die Datei und gibt deren Attribute zurück. Sie können alle Argumente, außer *fileURL* auf `null` setzen, wenn Sie den Wert nicht benötigen.

Argumente

*char *fileURL, unsigned long *attrs, unsigned long *filesize,
unsigned long *modtime, unsigned long *createtime*

- Das Argument *char *fileURL* verweist auf den String mit dem Namen der Datei, deren Attribute Sie abrufen. Sie müssen dieses Argument im URL-Format „file://“ angeben. Wenn *fileURL* einen Pfad im Dreamweaver-Ordner **Configuration** angibt, löst die Funktion `MM_GetConfigFileAttributes()` den Pfad vor dem Öffnen der Datei auf.
- *unsigned long *attrs* ist die Adresse einer Ganzzahl mit den zurückgegebenen Attributen (Informationen zur verfügbaren Attributen finden Sie unter „JSBool MM_SetConfigFileAttributes()“ auf Seite 345).
- Das Argument *unsigned long *filesize* ist die Adresse einer Ganzzahl, in der die Funktion die Dateigröße in Byte zurückgibt.
- Das Argument *unsigned long *modtime* ist die Adresse einer Ganzzahl, in der die Funktion die Uhrzeit der letzten Änderung der Datei zurückgibt. Die Uhrzeit wird im Format der Systemzeit zurückgegeben. Weitere Informationen über die Systemzeit finden Sie unter `DWfile.getModificationDate()` im *Dreamweaver API-Referenzhandbuch*.
- Das Argument *unsigned long *createtime* ist die Adresse einer Ganzzahl, in der die Funktion die Uhrzeit der Erstellung der Datei zurückgibt. Die Uhrzeit wird im Format der Systemzeit zurückgegeben. Weitere Informationen über die Systemzeit finden Sie unter `DWfile.getCreationDate()` im *Dreamweaver API-Referenzhandbuch*.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist. Gibt `JS_FALSE` zurück, wenn die Datei nicht existiert oder ein Fehler beim Abrufen der Attribute auftritt.

Beispiel

```
char dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3/
    Configuration/Extensions.txt";
unsigned long attrs;
unsigned long filesize;
unsigned long modtime;
unsigned long createtime;
MM_GetConfigAttributes(dwConfig, &attrs, &filesize, &modtime, &createtime);
```

JSBool MM_SetConfigFileAttributes()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion legt die von Ihnen angegebenen Attribute für die Datei fest, wenn diese sich von den aktuellen Attributen unterscheiden.

Befindet sich die festgelegte Datei-URL im Ordner **Configuration** von Dreamweaver, kopiert diese Funktion die Datei zuerst in den Ordner **Configuration** des Benutzers, bevor sie die Attribute festlegt. Sind diese Attribute identisch mit den aktuellen Dateiattributen, wird die Datei nicht kopiert.

Argumente

*char *fileURL, unsigned long attrs*

- *char *fileURL* verweist auf einen String mit dem Namen der Datei, deren Attribute Sie festlegen möchten, im URL-Format „file://“.
- *unsigned long attrs* legt die Attribute für die Datei fest. Sie können die Attribute mit einem logischen ODER für eine der folgenden Konstanten festlegen:

```
MM_FILEATTR_NORMAL
MM_FILEATTR_RDONLY
MM_FILEATTR_HIDDEN
MM_FILEATTR_SYSTEM
MM_FILEATTR_SUBDIR
```

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist. Gibt `JS_FALSE` zurück, wenn die Datei nicht vorhanden oder zum Löschen markiert ist.

Beispiel

```
char *dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3/
    Configuration/Extensions.txt";
unsigned long attrs;
attrs = (MM_FILEATTR_NORMAL | MM_FILEATTR_RDONLY);
int fileno = 0;
if(MM_SetConfigFileAttrs(dwConfig, attrs))
{
    fileno = MM_OpenConfigFile(dwConfig);
}
```

JSBool MM_CreateConfigFolder()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion legt am festgelegten Ort einen Ordner an.

Wenn *fileURL* einen Unterordner des Ordners **Configuration** von Dreamweaver angibt, legt die Funktion diesen im Ordner **Configuration** des Benutzers an. Wenn *fileURL* keinen Unterordner des Ordners **Configuration** von Dreamweaver angibt, legt die Funktion den festgelegten Ordner einschließlich aller übergeordneten Ordner im Pfad an, soweit diese nicht bereits vorhanden sind.

Argumente

*char *fileURL*

- *char *fileURL* verweist auf einen String im URL-Format „file://“ für den Konfigurationsordner, den Sie anlegen möchten.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

Beispiel

```
char *dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3  
/Configuration/Extensions.txt";  
MM_CreateConfigFolder(dwConfig);
```

JSBool MM_RemoveConfigFolder()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion entfernt den Ordner sowie seine Dateien und Unterordner. Befindet sich der Ordner im Ordner **Configuration** von Dreamweaver, wird dieser Unterordner in der Datei **mm_deleted_files.xml** zum Löschen markiert.

Argumente

*char *fileURL*

- *char *fileURL* verweist auf einen String für den zu entfernenden Ordner im URL-Format „file://“.

Rückgabewerte

Ein Boolescher Wert: `JS_TRUE` weist darauf hin, dass der Vorgang erfolgreich war. `JS_FALSE` weist darauf hin, dass der Vorgang fehlgeschlagen ist.

Beispiel

```
char *dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3  
/Configuration/Objects";  
MM_RemoveConfigFolder(dwConfig);
```


JSBool MM_DeleteConfigFile()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion löscht die Datei, falls vorhanden. Wenn die Datei in einem Unterordner des Ordners **Configuration** von Dreamweaver vorhanden ist, markiert die Funktion sie in **mm_deleted_files.xml** als zu löschende Datei.

Wenn das Argument *fileURL* keinen Ordner im Konfigurationsordner von Dreamweaver angibt, löscht die Funktion die angegebene Datei.

Argumente

*char *fileURL*

- *char *fileURL* verweist auf einen String für den zu entfernenden Konfigurationsordner im URL-Format „file://“.

Rückgabewerte

Ein Boolescher Wert: **JS_TRUE** weist darauf hin, dass der Vorgang erfolgreich war. **JS_FALSE** weist darauf hin, dass der Vorgang fehlgeschlagen ist.

Beispiel

```
char dwConfig = "file:///c:/Program Files/Adobe/Adobe Dreamweaver CS3
/Configuration/Objects/insertbar.xml";
MM_DeleteConfigFile(dwConfig);
```

C-Funktion über JavaScript aufrufen

Nachdem Sie sich mit C-Level-Erweiterungen in Dreamweaver und den Zusammenhängen mit bestimmten Datentypen und Funktionen vertraut gemacht haben, können Sie lernen, eine Bibliothek zu erstellen und eine Funktion aufzurufen.

Im nächsten Beispiel werden die folgenden fünf Dateien benötigt, die sich im Anwendungsordner **Tutorial_assets\Extending** von Dreamweaver als Archive für Macintosh und Windows befinden:

- **mm_jsapi.h** ist eine Header-Datei. Sie enthält die Definitionen für alle Datentypen und Funktionen (siehe „C-Level-Erweiterbarkeit und der JavaScript-Interpreter“ auf Seite 333).
- Die Datei **mm_jsapi_environment.h** definiert die **MM_Environment.h**-Struktur.
- Die Datei **MMInfo.h** gewährt Zugriff auf die Design Notes-API.
- Die Beispieldatei **Sample.c** definiert die Funktion `computeSum()`.
- Die Datei **Sample.mak** ist ein Makefile, mit dem Sie unter Verwendung von Microsoft Visual C++ die Quelldatei **Sample.c** in eine DLL integrieren können. Die entsprechende Datei zum Erzeugen eines Mach-O-Bundles mit Metrowerks CodeWarrior ist **Sample.mcp**, und **Sample.xcode** ist die Datei für Apple Xcode. Wenn Sie mit einem anderen Tool arbeiten, können Sie das Makefile selbst erstellen.

So erstellen Sie die DLL mit VS.Net 2003 unter Windows:

- 1 Wählen Sie **Datei > Öffnen >**, und wählen Sie **Sample.mak** aus. Dabei muss als Dateityp **Alle Dateien (*.*)** eingestellt sein. (VS.Net 2003 öffnet MAK-Dateien nicht direkt.) Sie werden dann aufgefordert, die Konvertierung des Projekts in das neue Format zu bestätigen.
- 2 Wählen Sie **Erstellen > Projektmappe neu erstellen**.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei **Sample.dll** im gleichen Ordner wie **Sample.mak** (bzw. in einem Unterordner).

So erstellen Sie die DLL mit Microsoft Visual C++ unter Windows:

1 Wählen Sie in Microsoft Visual C++ den Befehl **Datei > Arbeitsbereich** öffnen, und wählen Sie die Datei **Sample.mak** aus.

2 Wählen Sie **Erstellen > Alles neu erstellen**.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei **Sample.dll** im gleichen Ordner wie **Sample.mak** (bzw. in einem Unterordner).

So erstellen Sie die gemeinsame Bibliothek mit Metrowerks CodeWarrior 9 (oder höher) auf dem Macintosh:

1 Öffnen Sie die Datei **Sample.mcp**.

2 Erstellen Sie das Projekt (**Project > Make**), um ein Mach-O-Bundle zu erzeugen.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei **Sample.bundle** im gleichen Ordner wie **Sample.mcp**.

***Hinweis:** Das generierte Mach-O-Bundle kann nur ab Dreamweaver 8 verwendet werden. In älteren Dreamweaver-Versionen wird es nicht erkannt.*

So erstellen Sie die gemeinsame Bibliothek mit Apple Xcode 1.5 (oder höher) auf dem Macintosh:

1 Öffnen Sie die Datei **Sample.xcode**.

2 Erstellen Sie das Projekt (**Build > Build**), um ein Mach-O-Bundle zu generieren.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei **Sample.bundle** im Build-Ordner neben der Datei **Sample.xcode**.

***Hinweis:** Das generierte Mach-O-Bundle kann nur ab Dreamweaver 8 verwendet werden. In älteren Dreamweaver-Versionen wird es nicht erkannt.*

So rufen Sie die Funktion computeSum() über das Objekt „Insert Horizontal Rule“ auf:

1 Erstellen Sie den Unterordner **JSExtensions** im Ordner **Configuration** des Anwendungsordners von Dreamweaver.

2 Kopieren Sie die Datei **Sample.dll** (Windows) bzw. **Sample.bundle** (Macintosh) in den Ordner **JSExtensions**.

3 Öffnen Sie die Datei **HR.htm** mit einem Texteditor. Die Datei befindet sich im Ordner **Configuration\Objects\Common**.

4 Tragen Sie die Zeile `alert(Sample.computeSum(2,2));` wie folgt in die Funktion `objectTag()` ein:

```
function objectTag() {  
    // Gibt das einzufügende HTML-Tag zurück  
    alert(Sample.computeSum(2,2));  
    return "<HR>";  
}
```

5 Speichern Sie die Datei, und starten Sie Dreamweaver neu.

Um die Funktion `computeSum()` auszuführen, wählen Sie **Einfügen > HTML > Horizontale Linie**.

Ein Dialogfeld mit der Zahl 4, d. h. dem Ergebnis der Addition der Zahlen 2 und 2, wird eingeblendet.

Kapitel 24: Der Ordner „Shared“

Der Ordner **Shared** ist der zentrale Speicherplatz für Dienstprogrammfunktionen, Klassen und Bilder, die von allen Erweiterungen verwendet werden. Jede Erweiterung kann auf die Dateien in den Unterordnern des Ordners **Shared** verweisen, und Sie können gemeinsame benutzerdefinierte Dienstprogramme zu denen hinzufügen, die bereits in Adobe Dreamweaver CS3 bereitgestellt werden. Die Konfigurationsordner für mehrere Benutzer, die in Windows XP, Windows 2000 und Mac OS X installiert sind, verfügen ebenfalls über den Ordner **Shared** für einzelne Benutzeranpassungen. Wenn Sie beispielsweise eine Erweiterung von Adobe Exchange installieren, stellen Sie möglicherweise fest, dass die neue Erweiterung Inhalt zur Ihrem Benutzerordner **Configuration\Shared** hinzufügt statt zum Ordner **Configuration\Shared** von Dreamweaver. Weitere Informationen über die Dreamweaver-Konfigurationsordner auf einem Mehrbenutzer-Computer finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 70.

Inhalt des Ordners „Shared“

Der Ordner **Shared** verfügt über Unterordner, die von mehreren Erweiterungen verwendete Dateien enthalten, einschließlich Funktionen zum Durchsuchen der Ordnerstruktur eines Benutzers, Einfügen eines Struktursteuerelements, Erstellen bearbeitbarer Tabellen und weiterer Funktionen.

Hinweis: Die JavaScript-Dateien im Ordner **Shared** verfügen über Kommentare im Code, die Informationen zu den enthaltenen Funktionen liefern.

Sie sollten sich nicht nur die JavaScript-Dateien im Ordner **Shared** ansehen, sondern auch nach HTML-Dateien im Konfigurationsordner suchen, die diese JavaScript-Dateien enthalten, um zu überprüfen, wie diese verwendet werden.

In der Regel verwenden Sie die Funktionen und Ressourcen in den Ordnern **Common** und **MM**, oder Sie fügen Ressourcen zum Ordner **Common** zur Verwendung mit neuen Erweiterungen hinzu. Sie sollten zunächst immer im Ordner **Shared\Common\Scripts** nach Dienstprogrammen und Funktionen suchen. Dabei handelt es sich um die aktuellsten Funktionen und Dienstprogramme, die auch die formale Schnittstelle zum Ordner **Shared** umfassen. Die Dateien in anderen Ordnern verwenden Sie auf eigene Gefahr, da diese unter Umständen nicht mehr aktuell sind.

Der Ordner **Shared** enthält insbesondere die im Folgenden aufgeführten nützlichen Ordner.

Der Ordner „Common“

Der Ordner **Common** verfügt über gemeinsame Skripts und Klassen für Drittanbieter-Erweiterungen.

Datei	Beschreibung
CodeBehindMgr.js	Enthält Funktionen zur Erstellung von CodeBehind-Dokumenten. Mit CodeBehind-Dokumenten können Sie spezielle Seiten erstellen, die den Code für die Logik einer Benutzeroberfläche vom Code für das Design der Benutzeroberfläche trennen. Die in dieser Datei definierten Methoden von <code>JSCodeBehindMgr</code> können neue CodeBehind-Dokumente erstellen und die Verknüpfung zu Entwurfsdokumenten verwalten.
ColumnValueNodeClass.js	Enthält Funktionen zum Zuordnen von Datenbankspalten zu Werten. Mithilfe der in dieser Datei definierten Methoden von <code>ColumnValueNode</code> können Sie verschiedene Werte und Eigenschaften einer Datenbankspalte abrufen und festlegen. Dreamweaver verwendet diese Speicherklasse bei der Anwendung und Untersuchung von Objekten für Bearbeitungsfunktionen (Objekte zum Einfügen und Aktualisieren von Datensätzen) sowie bei der Arbeit mit der <code>SQLStatement</code> -Klasse.
CompilerClass.js	Enthält Funktionen für eine Basisklasse, die von <code>CompilerASPNetCSharp</code> und <code>CompilerASPVBNet</code> verwendet wird, aber nicht zur Unterstützung weiterer Compiler erweitert werden konnte.
DataSourceClass.js	Enthält Funktionen, die die Rückgabestruktur für „findDynamicSources()“ auf Seite 285 definieren.

Datei	Beschreibung
DBTreeControlClass.js	Enthält Funktionen zum Erstellen eines Datenbankstruktur-Steuerelements. Diese Klasse wird zum Erstellen und zur Interaktion mit einem Datenbankstruktur-Steuerelement verwendet. Um ein Datenbankstruktur-Steuerelement wie das in den Serververhalten erweiterter Datensatzgruppen zu erstellen, erstellen Sie eine spezielle <code><select></code> -Liste mit <code>type="mmdatabasetree"</code> in Ihrer HTML-Datei. Fügen Sie eine <code>DBTreeControl</code> -Klasse an das HTML-Steuerelement an, indem Sie den <code><select></code> -Listennamen an den Klassenkonstruktor übergeben. Verwenden Sie anschließend die <code>DBTreeControl</code> -Funktionen zur Manipulation des Steuerelements.
dotNetUtils.js	Enthält Funktionen zur Unterstützung der Arbeit mit Objekteigenschaftsinspektoren sowie Serververhalten für ASP.NET-Formularsteuerelemente, die übersetzt werden.
dwscrip.js	In dieser Hauptdatei finden Sie zahlreiche nützliche Funktionen für alle Dreamweaver-Erweiterungen. Sie enthält Funktionen zur Arbeit mit Strings, Dateien, Design Notes usw.
dwscripExtData.js	Diese Datei ist eine Erweiterung der Datei dwscrip.js . Diese Datei unterstützt die Arbeit mit Serververhalten, insbesondere den Serververhalten-EDML-Dateien. Sie wird von Dreamweaver sehr häufig bei der Implementierung von Serververhalten verwendet.
dwscripServer.js	Diese Datei ist eine Erweiterung der Datei dwscrip.js . Sie enthält Funktionen für bestimmte Servermodelle. Viele dieser Funktionen werden bei der Arbeit mit Serververhalten verwendet.
GridControlClass.js	Verwenden Sie diese Klasse zum Erstellen und Manipulieren eines bearbeitbaren Rasters. Erstellen Sie eine spezielle Auswahlliste in Ihrem HTML-Code, und fügen Sie in JavaScript diese Klasse an, um das Raster zu manipulieren.
ImageButtonClass.js	Diese Klasse vereinfacht die Steuerung des Aussehens einer Schaltfläche in folgenden Situationen: gedrückte Schaltfläche, Mauszeiger über der gedrückten Schaltfläche, Mauszeiger über der Schaltfläche, deaktivierte gedrückte Schaltfläche.
ListControlClass.js	Enthält Funktionen zur Verwaltung eines <code><select></code> -Tags, auch als Listensteuerelemente bezeichnet. Die Methoden des Objekts <code>ListControl</code> in dieser Datei rufen den Wert des Steuerelements <code>SELECT</code> ab, legen ihn fest und ändern ihn.
PageSettingsASPNet.js	Enthält Funktionen, die die Eigenschaften eines ASP.NET-Dokuments festlegen.
RadioGroupClass.js	Enthält Funktionen, die eine Optionsschaltergruppe definieren und verwalten. Die Methoden des Objekts <code>RadioGroup</code> in dieser Datei legen die Werte und das Verhalten einer Optionsschaltergruppe fest und rufen sie ab. Sie fügen diese Klasse an die Optionsschalter in Ihrem HTML-Code an, um deren Verhalten zu steuern.
SBDDatabaseCallClass.js	Eine Unterklasse der Klasse <code>ServerBehavior</code> . Diese Klasse enthält für Datenbankaufrufe spezifische Funktionen, wie beispielsweise zum Aufruf einer gespeicherten Prozedur, zum Verwenden von SQL zur Rückgabe einer Datensatzgruppe usw. Hierbei handelt es sich um eine abstrakte Basisklasse, d. h., sie kann nicht allein erstellt und verwendet werden. Um sie verwenden zu können, müssen Sie Unterklassen zu <code>SBDDatabaseCall</code> bilden und die Platzhalterfunktionen implementieren. Dreamweaver verwendet diese Klasse zur Implementierung ihrer Datensatzgruppe und gespeicherten Prozedur-Serververhalten.
ServerBehaviorClass.js	Enthält Funktionen, die Informationen über Serververhalten an Dreamweaver übermitteln. Sie können diese Klasse bei der Implementierung eigener Serververhalten als Unterklasse verwenden.
ServerSettingsASPNet.js	Enthält Funktionen, die Eigenschaften eines ASP.NET-Servers speichern.
SQLStatementClass.js	Enthält Funktionen, mit denen Sie SQL-Anweisungen wie <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> und <code>DELETE</code> sowie gespeicherte Prozeduranweisungen erstellen und bearbeiten können.
tagDialogsCmn.js	Enthält Funktionen, mit denen Sie benutzerdefinierte Tag-Dialogfelder erstellen können. Die in dieser Datei definierten Methoden des Objekts <code>tagDialog</code> ändern Attribute und Werte für ein bestimmtes Tag.
TagEditClass.js	Enthält Funktionen, die Tags bearbeiten, ohne das DOM der aktuellen Seite zu ändern. Die in dieser Datei definierten Methoden des Objekts <code>TagEdit</code> rufen den Wert, die Attribute und untergeordneten Elemente des Tags ab und legen sie fest.
TreeControlClass.js	Enthält Funktionen zur Verwaltung eines Struktursteuerelements in Dreamweaver. Die in dieser Datei definierten Methoden des Objekts <code>TreeControl</code> rufen die Werte in der Struktur ab, legen sie fest und ordnen sie neu. Sie fügen diese Klassen an ein spezielles <code>MM: TREECONTROL</code> -Tag in Ihrem HTML-Code an, um das Struktursteuerelement zu verwalten.
XMLPropSheetClass.js	Enthält Funktionen, die den Speicherort und die Werte einer XML-Eigenschaftenseite verwalten.

Der Ordner „MM“

Der Ordner **MM** enthält die gemeinsamen Skripts, Bilder und Klassen, die von den im Lieferumfang von Dreamweaver enthaltenen Erweiterungen verwendet werden, einschließlich der Skripts zum Aufbau einer Navigationsleiste, der Angabe von Vorlade-Aufrufen und der Definition von Kurzbefehlsdefinitionen.

Der Unterordner „Scripts subfolder“

Der Unterordner **Scripts** enthält die folgenden Dienstprogrammfunktionen:

Datei	Beschreibung
CFCutilities.js	Enthält Dienstprogrammfunktionen, die sich auf ColdFusion-Komponenten beziehen. Funktionen analysieren Attribute innerhalb des öffnenden Tags eines bestimmten Knotens, analysieren eine CFC-Struktur, rufen das aktuelle URL-DOM ab, rufen das CFC-DOM ab usw.
event.js	Enthält Funktionen zum Registrieren von Ereignissen, Benachrichtigen von Parteien über Ereignisse aus der Datei <code>menus.xml</code> und Hinzufügen von Ereignisbenachrichtigungen zur Datei <code>menus.xml</code> .
FlashObjects.js	Enthält Funktionen zum Aktualisieren des Farbwählers, Überprüfen auf Hex-Farbe, Überprüfen auf einen absoluten Link, Hinzufügen einer Erweiterung zu einem Dateinamen, Erstellen von Fehlermeldungen, Festlegen von Flash-Attributen, Überprüfen eines Links auf ein Flash-Objekt usw.
insertFireworksHTML.js	Enthält Funktionen zum Einfügen von Adobe Fireworks CS3 HTML-Code in Dreamweaver-Dokumente. Diese Funktionen überprüfen, ob es sich bei dem aktuellen Dokument um ein Fireworks-Dokument handelt, fügen Fireworks HTML-Code an der Einfügemarke ein, aktualisieren den Fireworks-Stilblock in Dreamweaver usw. Umfasst darüber hinaus verwandte Dienstprogrammfunktionen.
jumpMenuUI.js	Enthält Funktionen zur Verwendung mit dem Sprungmenü-Objekt und Sprungmenü-Verhalten. Diese Funktionen füllen Menüoptionen aus, erstellen eine Optionsbezeichnung, fügen eine Option hinzu, löschen eine Option usw.
keyCodes.js	Enthält ein Array von Tastatur-Tastencodes.
navBar.js	Enthält Klassen und Funktionen für die Arbeit mit der Navigationsleiste und Navigationsleistenelementen. Umfasst Funktionen zum Hinzufügen, Löschen und Manipulieren von Navigationsleistenelementen.
NBInit.js	Enthält Funktionen für Navigationsleisten-Bildverhalten.
pageEncodings.js	Definiert verschiedene Sprachcodes.
preload.js	Enthält Funktionen zum Hinzufügen und Löschen von Vorlade-Bildauffufen an die Prozedur <code>BODY/onLoad MM_preloadImages</code> .
RecordsetDialogClass.js	Enthält die statische Klasse und Funktionen zur Anzeige der Benutzeroberfläche für das Datensatzgruppe-Serververhalten. Diese Funktionen legen fest, ob die einfache oder die erweiterte Oberfläche angezeigt wird. Enthält außerdem Funktionen, die von den Benutzeroberflächen-Implementationen gemeinsam verwendet werden, und verwaltet die Umschaltung zwischen den Benutzeroberflächen.
sbUtils.js	Enthält gemeinsam verwendete Funktionen zur Verwendung in Adobe-Serververhalten. Die Klasse <code>dws-scripts</code> im Ordner Configuration\Shared\Common\Scripts enthält Dienstprogramme für allgemeinere Zwecke.
setText.js	Enthält Funktionen zum Umwandeln eines Ausdruck-Strings mit <code>escape</code> oder <code>unescape</code> und Extrahieren eines Ausdruck-Strings.
sortTable.js	Enthält Funktionen zum Initialisieren und Sortieren einer Tabelle sowie Funktionen zum Sortieren eines Arrays, zum Ändern des Mauszeigers in ein Handsymbol oder einen Zeiger und zur Überprüfung des Typs und der Version des Browsers.

Der Ordner **Scripts** enthält außerdem zwei Unterordner, **Class** und **CMN**.

Der Ordner „Class“

Der Ordner **Class** beinhaltet die folgenden Dienstprogrammfunktionen:

Datei	Beschreibung
classCheckbox.js	Unterstützt die Manipulation eines Kontrollkästchen-Steuerelements in der HTML-Erweiterung.
FileClass.js	Enthält eine Klasse, die eine Datei im Dateisystem repräsentiert. Die Pfade werden zur plattformübergreifenden Kompatibilität als URLs dargestellt. Zu den Methoden zählen <code>toString()</code> , <code>getName()</code> , <code>getSimpleName()</code> , <code>getExtension()</code> , <code>getPath()</code> , <code>setPath()</code> , <code>isAbsolute()</code> , <code>getAbsolutePath()</code> , <code>getParent()</code> , <code>getAbsolutePathParent()</code> , <code>exists()</code> , <code>getAttributes()</code> , <code>canRead()</code> , <code>canWrite()</code> , <code>isFile()</code> , <code>isFolder()</code> , <code>listFolder()</code> , <code>createFolder()</code> , <code>getContents()</code> , <code>setContents()</code> , <code>copyTo()</code> und <code>remove()</code> .
GridClass.js	Enthält eine Klasse, die MM: TREECONTROL verwaltet.
GridControlClass.js	Ältere Version von <code>GridControlClass</code> im Ordner Common . Siehe die Datei GridControlClass.js im Ordner Shared\Common\Scripts .
ImageButtonClass.js	Ältere Version von <code>ImageButtonClass</code> im Ordner Common . Siehe Datei ImageButtonClass.js im Ordner Shared\Common\Scripts .
ListControlClass.js	Ältere Version von <code>ListControlClass</code> im Ordner Common . Siehe die Datei Shared\Common\Scripts>ListControlClass.js .
NameValuePairClass.js	Erstellt und verwaltet eine Liste mit Name-Wert-Paaren. Namen können beliebige Zeichen enthalten. Werte können leer sein, jedoch nicht auf <code>null</code> gesetzt sein, was dem Löschen entspricht.
PageControlClass.js	Beispiel einer Seitenklasse, die mit der Klasse <code>TabControl</code> verwendet wird. Weitere Informationen finden Sie in der Beschreibung von <code>TabControlClass.js</code> .
PreferencesClass.js	Enthält ein Objekt und Methoden, die alle Voreinstellungsinformationen für einen Befehl enthalten.
RadioGroupClass.js	Ältere Version von <code>RadioGroupClass</code> im Ordner Common . Siehe die Datei RadioGroupClass.js im Ordner Shared\Common\Scripts .
TabControlClass.js	Unterstützt die Erstellung einer Erweiterung mit mehreren Registerkartenansichten, <code>page.lastUnload()</code>

Der Ordner „CMN“

Der Ordner CMN beinhaltet die folgenden Dienstprogrammfunktionen:

Datei	Beschreibung
dateID.js	Enthält zwei Funktionen: <code>createDateID()</code> und <code>decipherDateID()</code> . Liegen drei Strings vor ("dayFormat", "dateFormat" und "timeFormat"), erstellt <code>createDateID()</code> eine Kennung für sie. Liegt ein Datums-Array vor, gibt <code>decipherDateID()</code> ein Array mit drei Elementen zurück: <code>dayFormat</code> , <code>dateFormat</code> und <code>timeFormat</code> .
displayHelp.js	Enthält eine Funktion, die das angegebene Hilfedokument anzeigt.
docInfo.js	Enthält Funktionen, die Informationen über das Benutzerdokument bereitstellen. Die von Funktionen durchgeführten Operationen umfassen die Rückgabe eines Arrays von Objektreferenzen für einen bestimmten Browsertyp und ein bestimmtes Browser-Tag, die Rückgabe aller Instanzen eines bestimmten Tag-Namens, die Suche nach einem Tag, das die aktuelle Auswahl umbricht, usw.
DOM.js	Enthält allgemeine Hilfsfunktionen zur Arbeit mit dem Dreamweaver-DOM. Umfasst Funktionen, die den Stammknoten des aktiven Dokuments abrufen, ein bestimmtes benanntes Tag suchen, eine Liste von Knoten von der angegebenen Startknoten aus erstellen, überprüfen, ob ein bestimmtes Tag in einem anderen Tag enthalten ist, verschiedene Operationen für Verhaltensfunktionen durchführen usw.
enableControl.js	Enthält eine Funktion, <code>setEnabled()</code> , die je nach erhaltenem Argument ein Steuerelement aktiviert oder deaktiviert. Es ist möglich, ein Steuerelement, das bereits aktiviert ist, zu aktivieren, und ein Steuerelement, das bereits deaktiviert ist, zu deaktivieren.
errmsg.js	Enthält Protokollierungsfunktionen zum Sammeln von Tracing-Ausgaben in ein Array von Protokollseiten, die in einem Dialogfeld angezeigt werden.
file.js	Enthält Funktionen, die sich auf Dateioperationen beziehen. Diese Funktionen ermöglichen es dem Benutzer, nach lokalen Dateinamen zu suchen, den relativen Pfad in den URL-Pfad der Datei zu konvertieren, den Dateinamen des aktuellen Dokuments zurückzugeben, festzustellen, ob ein bestimmtes Dokument auf der aktuellen Site gespeichert wurde (und den dokumentrelativen Pfad zurückzugeben) oder ob eine bestimmte Datei zurzeit geöffnet ist.

Datei	Beschreibung
form.js	Enthält Funktionen, die einem gegebenen Textstring ein Formular hinzufügen, wenn im aktuellen Dokument oder AP-Element noch kein Formular vorhanden ist. Umfasst Funktionen, die feststellen, ob ein Objekt ein AP-Element ist und ob sich die Einfügemarke in einem Formular befindet.
handler.js	Enthält Funktionen, die eine Funktion für eine Ereignisprozedur abrufen, einer Ereignisprozedur eine Funktion hinzufügen und eine Funktion aus einer Ereignisprozedur löschen.
helper.js	Enthält ein paar nützliche Funktionen, die Kodierung ersetzen, Anführungszeichen (") mit unescape umwandeln, überprüfen, ob sich ein Knoten in einem Auswahlbereich befindet, und nach doppelten Objekt-namen suchen.
insertion.js	Enthält die Funktion <code>insertIntoDocument()</code> , die einen Textstring an der Einfügemarke in das Dokument einfügt. Enthält außerdem die unterstützenden Funktionen <code>getHigherBlockTag()</code> und <code>arrContains()</code> . Die Funktion <code>getHigherBlockTag()</code> ruft den nächsthöheren <code>blockTag</code> ab, wie im <code>blockTags</code> -Array definiert, und die Funktion <code>arrContains()</code> sucht in einem Array nach einem bestimmten Element.
localText.js	Reservierte Variable, die nicht für die allgemeine Verwendung vorgesehen ist. Verwenden Sie stattdessen Startup\mminit.htm oder die Strings in den Configuration\Strings*.xml -Dateien von Dreamweaver.
menuitem.js	Enthält Funktionen, die Werte zu einem aufgeführten Menüelement hinzufügen oder daraus löschen.
niceName.js	Enthält Funktionen, die ein Array von Objektreferenzen in ein Array einfacher Namen konvertieren.
quickString.js	Enthält Funktionen, die kleinere Strings zusammenfassen, ohne jedes Mal eine Speicherzuweisung vorzunehmen.
string.js	Enthält einen generischen Satz von Funktionen für die Manipulation und Analyse von Textstrings. Zu den Funktionen gehören: <code>extractArgs()</code> , <code>escQuotes()</code> , <code>unescQuotes()</code> , <code>quoteMeta()</code> , <code>errMsg()</code> , <code>badChars()</code> , <code>getParam()</code> , <code>quote()</code> , <code>stripSpaces()</code> , <code>StripChars()</code> , <code>AllInRange()</code> , <code>reformat()</code> , <code>trim()</code> , <code>createDisplayString()</code> , <code>entityNameEncode()</code> , <code>entityNameDecode()</code> , <code>stripAccelerator()</code> und <code>Sprintf()</code> .
TemplateUtils.js	Enthält Dienstprogrammfunktionen für Dreamweaver-Vorlagen. Funktionen fügen einen bearbeitbaren Bereich oder einen sich wiederholenden Bereich in ein Dokument ein, überprüfen ein Dokument auf einen bestimmten bearbeitbaren Bereich usw.
UI.js	Enthält generische Funktionen zur Steuerung der Benutzeroberfläche. Diese Funktionen suchen nach einem bestimmten Objekt im aktuellen Dokument, laden ausgewählte Listenoptionen mit lokalisierten Strings, geben den Attributwert für eine ausgewählte Option zurück und brechen den Text in Nachrichten für eine Warnmeldung um.

Andere Ordner

Nachfolgend werden andere wichtige Unterordner des Ordners **Shared** beschrieben:

• Steuerelemente

Der Ordner **Controls** enthält die Elemente, die zum Aufbau eines Serververhaltens verwendet werden. Zu diesen Steuerelementen zählen Benutzeroberflächen für Text- und Datensatzgruppenmenüs.

Hinweis: Diese Steuerelemente werden vom Dreamweaver-Serververhaltensgenerator und vielen anderen Serververhalten von Dreamweaver verwendet, wobei einige auch hilfreich für die Verwaltung von Steuerelementen in Ihrer Erweiterung sind.

• Fireworks

Der Ordner **Fireworks** enthält die unterstützenden Dateien für die Fireworks-Integration.

• UltraDev

Dreamweaver verwendet diesen Ordner hauptsächlich zur Gewährleistung der Abwärtskompatibilität. Er sollte nicht für neue Erweiterungen verwendet werden. Verwenden Sie den Dreamweaver-Ordner **Configuration\Shared\Common**, in dem die meisten dieser Funktionen ebenfalls enthalten sind. Weitere Informationen finden Sie unter „Der Ordner „Common““ auf Seite 349.

Den Ordner „Shared“ verwenden

Suchen Sie als Erstes im Dreamweaver-Ordner **Configuration\Shared\Common** nach nützlichem Erweiterungscode, da dieser Ordner die aktuellsten und am häufigsten verwendeten Funktionen enthält.

Erweiterungen können die Ressourcen im Ordner **Shared** für eigene Funktionen verwenden. Ein Objekt, ein Befehl oder eine andere Erweiterung kann eine der JavaScript-Dateien im Ordner **Shared** als Quelldatei in einem `script`-Tag angeben und die Funktion dann im Body-Abschnitt der Datei oder einer anderen enthaltenen JavaScript-Datei verwenden. Objekte und Befehle können sogar mehrere JavaScript-Dateien miteinander verknüpfen, und diese JavaScript-Dateien können Ressourcen im Ordner **Shared** nutzen.

Öffnen Sie beispielsweise die Hypertext-Objektdatei (Hyperlink.htm) im Anwendungsordner **Configuration/Objects/Common**. Beachten Sie, dass das `head`-Tag der Datei die folgenden Zeilen enthält:

```
<script language="javascript" src="../../Shared/Common/Scripts/ListControlClass.js"></script>
<script language="javascript" src="Hyperlink.js"></script>
```

Wenn Sie die zugehörige Datei **Hyperlink.js** öffnen, sehen Sie die folgenden Zeilen:

```
LIST_LINKS = new ListControl('linkPath');

und

LIST_TARGETS = new ListControl('linkTarget');
```

Mit den `new listControl`-Deklarationen definiert die Datei **Hyperlink.js** zwei neue `ListControl`-Objekte. Der Code in der Datei **Hyperlink.htm** fügt diese dann an die `SELECT`-Steuerelemente im Formular wie folgt an:

```
<td align="left"> <input name="linkText" type="text" class="basicTextField" value="">

und

<td align="left" nowrap><select name="linkPath" class="basicTextField" editable="true">
```

Jetzt kann das Skript **Hyperlink.js** Methoden aufrufen oder Eigenschaften von den Objekten `LIST_LINKS` und `LIST_TARGETS` abrufen, um mit den `SELECT`-Steuerelementen im Formular zu interagieren.

Index

A

- action-Tag 138
- activate-Tag 139
- addDynamicSource() 283
- Aktionsdateien 221
- alert() 88
- analyzeServerBehavior() 238
- Andocken von Symbolleisten 158
- anpassen
 - Anzeige von Dialogfeldern 6
 - Arbeitsbereich-Layouts 22
 - Bearbeiten von
 - Konfigurationsdateien 5
 - Dreamweaver 1
 - Drittanbieter-Tags 5
 - Einfügeleiste 5
 - in einer Mehrbenutzerumgebung 12
 - Interpretation von Drittanbieter-Tags 8
 - Komponenten (Bedienfeld) 296
 - Menüs 5
 - Seitendesigns 6
 - Standarddokumente 6
- Anzeige von Dialogfeldern 6
- API für C-Erweiterbarkeit
 - MM_CreateConfigFolder() 346
 - MM_DeleteConfigFile() 347
 - MM_RemoveConfigFolder() 346
 - MM_SetConfigFileAttributes() 345
- API für Probleme bei der Browserkompatibilitätsprüfung 117
- API-Funktionen für Berichte
 - beginReporting() 186
 - commandButtons() 187
 - configureSettings() 187
 - endReporting() 186
 - processfile() 186
 - windowDimensions() 188
- API-Funktionen für das Bedienfeld „Komponenten“
 - getCodeViewDropCode() 300
 - getComponentChildren() 298
 - getContextMenuId() 299
 - getSetupSteps() 300
 - handleDesignViewDrop() 302
 - handleDoubleClick() 303
 - setupStepsCompleted() 302
- toolbarControls() 304
- API-Funktionen für Datenquellen
 - addDynamicSource() 283
 - deleteDynamicSource() 283
 - displayHelp() 284
 - editDynamicSource() 284
 - findDynamicSources() 285
 - generateDynamicDataRef() 285
 - generateDynamicSourceBindings() 286
 - Informationen 275
 - inspectDynamicDataRef() 287
- API-Funktionen für Datenübersetzer
 - getTranslatorInfo() 327
 - liveDataTranslateMarkup() 330
 - translateDOM() 328
 - translateMarkup() 329
- API-Funktionen für den Eigenschaftensinspektor
 - canInspectSelection() 206
 - displayHelp() 207
 - inspectSelection() 207
- API-Funktionen für den Tag-Editor
 - applyTag() 199
 - inspectTag() 198
 - validateTag() 199
- API-Funktionen für die C-Level-Erweiterbarkeit
 - JS_BooleanToValue() 337
 - JS_DefineFunction() 334
 - JS_DoubleToValue() 337
 - JS_ExecuteScript() 340
 - JS_GetArrayLength() 339
 - JS_GetElement() 339
 - JS_IntegerToValue() 337
 - JS_NewArrayObject() 338
 - JS_ObjectToValue() 338
 - JS_ObjectType() 338
 - JS_ReportError() 340
 - JS_SetElement() 339
 - JS_StringToValue() 336
 - JS_ValueToBoolean() 336
 - JS_ValueToDouble() 335
 - JS_ValueToInteger() 335
 - JS_ValueToObject() 336
 - JS_ValueToString() 334
 - JSNative 334
 - MM_ConfigFileExists() 343
 - MM_GetConfigFileAttributes() 344
 - MM_GetConfigFolderList() 342
 - MM_OpenConfigFile() 343
- API-Funktionen für Menübefehle
 - canAcceptCommand() 153
 - commandButtons() 153
 - getDynamicContent() 154
 - isCommandChecked() 154
 - receiveArguments() 155
 - setMenuText() 156
 - windowDimensions() 156
- API-Funktionen für schwebende Bedienfelder
 - displayHelp() 215
 - documentEdited() 215
 - getDockingSide() 216
 - initialPosition() 216
 - initialTabs() 217
 - isATarget() 217
 - isAvailableInCodeView() 218
 - isResizable() 218
 - selectionChanged() 212, 218
- API-Funktionen für Serverformate
 - applyFormat() 291
 - applyFormatDefinition() 292
 - canRecognizeDocument() 307
 - deleteFormat() 292
 - formatDynamicDataRef() 293
 - getFileExtensions() 308
 - getLanguageSignatures() 308
 - getServerExtension() 309
 - getServerInfo() 310
 - getServerLanguages() 310
 - getServerModelDelimiters() 311
 - getServerModelDisplayName() 311
 - getServerModelExtDataNameUD4() 311
 - getServerModelFolderName() 312
 - getServerSupportsCharset() 312
 - getVersionArray() 312
 - Informationen 307
 - inspectFormatDefinition() 293
- API-Funktionen für Serververhalten
 - analyzeServerBehavior() 238
 - applyServerBehavior() 239
 - canApplyServerBehavior() 239
 - copyServerBehavior() 240

- deleteServerBehavior() 240
- displayHelp() 241
- findServerBehaviors() 241
- inspectServerBehavior() 241
- pasteServerBehavior() 242
- API-Funktionen für
 - Symbolleistenbefehle
 - canAcceptCommand() 174
 - getCurrentValue() 174
 - getDynamicContent() 175
 - getMenuID() 176
 - getUpdateFrequency() 177
 - isCommandChecked() 177
 - isDOMRequired() 178
 - receiveArguments() 179
 - showIf() 180
- APIs für Erweiterungen, Typen 67
- APIs, Typen
 - Befehle 129
 - Berichte 186
 - C-Level-Erweiterbarkeit 334
 - Datenformatierung 289
 - Datenquellen 283
 - Eigenschafteninspektor 206
 - Ergebnisfenster 183
 - Komponenten (Bedienfeld) 298
 - Menübefehle 153
 - Objekte 111
 - Probleme 119
 - schwebendes Bedienfeld 215
 - Serverformate 291
 - Servermodell 307
 - Serververhalten 238
 - Symbolleistenbefehl 174
 - Tag-Editor 198
- API-Typen
 - Datenübersetzer 313
 - Verhalten 226
- applyBehavior() 226
- applyFormat() 291
- applyFormatDefinition() 292
- applySB() 243
- applyServerBehavior() 239
- applyTag() 199
- appName-Eigenschaft 93
- appVersion-Eigenschaft 93
- Arbeitsbereich
 - Informationen 158
 - Layouts, anpassen 22
- Argumente
 - receiveArguments() 155
 - vom menuItem-Tag übergeben 144
- arguments-Attribut 173
- array-Objekt 88
- Attribute
 - Argumente 173
 - Breite 170
 - checked 100, 171
 - colorRect 171
 - command 100, 173
 - disabledImage 170
 - domRequired 171
 - enabled 100, 171
 - file 100, 171
 - id 99, 169
 - image 99
 - image-Objekt 169
 - label 170
 - menu_ID 171
 - overImage 170
 - showIf 169
 - Symbolleistenelement-Tags 169
 - tooltip 170
 - update 172
 - value 172
- attributes-Eigenschaft 91
- attributes-Tag 266
- Attributübersetzer
 - Beispielcode 320
 - Debugging 319
 - erstellen 314
 - Informationen 314
- B**
- Bearbeiten
 - Menüelemente 140
 - Schemen, Codefarben 60
- Bedienfelderweiterungen, Informationen 68
- Beenden-Befehle 70
- Befehle
 - Beispielcode 124
 - Benutzererfahrung 123
 - Flash-SWF-Dateien hinzufügen 84
 - in Menüs einfügen 124
 - Menübefehle 133
 - Symbolleiste 158
- Befehls-API-Funktionen
 - canAcceptCommand() 129
 - commandButtons() 130
 - isDOMRequired() 130
 - receiveArguments() 131
 - windowDimensions() 131
- Befehlserweiterungen, Informationen 67
- beginReporting() 186
- behaviorFunction() 227
- Beispiele
 - Attributübersetzer 320
 - dynamische Menüs 147
 - eigenständige Berichte 184
 - einfache Symbolleiste 159
 - Erweiterung für das Menü Befehle 124
 - Probleme 118
 - schwebendes Bedienfeld 210
 - Site-Berichte 182
 - Tag-Editor 195
 - Verhaltenserweiterung: 222
- benutzerdefinierte JavaScript-Steuerelemente 77
- Benutzeroberflächen von Erweiterungen 75
- Berichte
 - eigenständige 184
 - Site 181
- Berichtserweiterungen, Informationen 68
- Betriebssystem des Benutzers 93
- Bindungen-Inspektor 275
- Block/Tag 234
- Block/Tag-Übersetzer
 - Beispielcode 323
 - Debugging 319
 - Informationen 314
- blockEnd-Tag, Codefarben 45
- blockStart-Attribut
 - Beschreibung 55
 - customText-Wert 55
 - innerTag-Wert 56
 - innerText-Wert 55
 - nameTagScript-Wert 57
 - nameTag-Wert 56
 - outerTag-Wert 56
- blockStart-Tag, Codefarben 45
- blur() 88
- body-Eigenschaft 90
- boolean-Objekt 88
- brackets-Tag, Codefarben 46
- Browserkompatibilität ermitteln 117
- Browserprofile
 - css-support-Tag 64
 - property-Tag 64
 - value-tag 65
- button-Objekt 88
- button-Tag 98, 165

C

- canAcceptBehavior() 228
- canAcceptCommand() 148, 153, 159, 174
- canApplyServerBehavior() 239
- canDrag--Attribut 99
- canInsertObject() 111
- canRecognizeDocument() 307
- category-Tag 97
- C-Funktionen
 - Aufruf aus JavaScript 347
 - in der Datei mm_jsapi.h 333
- charEnd-Tag, Codefarben 46
- charEsc-Tag, Codefarben 47
- charStart-Tag, Codefarben 46
- checkbox-Objekt 88
- checkboxbutton-Tag 98, 165
- checked-Attribut 100, 171
- childNodes-Eigenschaft
 - von comment-Objekten 93
 - von document-Objekten 90
 - von tag-Objekten 91
 - von text-Objekten 92
- clearInterval() 88
- clearTimeout() 88
- C-Level-Erweiterbarkeit, in Übersetzern 313, 333
- close() 88
- closeTag-Tag 267
- code hints
 - function-Tag 39
- Codefarben
 - Beispiele 61
 - blockEnd-Tag 45
 - blockStart-Tag 45
 - brackets-Tag 46
 - charEnd-Tag 46
 - charEsc-Tag 47
 - charStart-Tag 46
 - commentEnd-Tag 47
 - commentStart-Tag 47
 - CSS-Beispieltext 62
 - cssImport-Tag 47
 - cssMedia-Tag 47
 - cssProperty-Tag 48
 - cssSelector-Tag 48
 - cssValue-Tag 48
 - defaultAttribute-Tag 49
 - defaultTag-Tag 49
 - defaultText-Tag 49
 - endOfLineComment-Tag 49
 - entity-Tag 50
 - Farbeinstellungsdatei 43

- functionKeyword-Tag 50
- idChar1-Tag 50
- idCharRest-Tag 50
- ignoreCase-Tag 51
- ignoreMMTPParams-Tag 51
- ignoreTags-Tag 51
- Informationen 43
- isLocked-Tag 51
- JavaScript 62
- keywords-Tag 52
- keyword-Tag 52
- numbers-Tag 52
- operators-Tag 52
- regex-Tag 53
- sampleText-Tag 53
- Schemaverarbeitung 57
- Schemen bearbeiten 60
- scheme-Tag 44
- searchPattern-Tag 54
- Stil, Colors.xml (Datei) 43
- stringEnd-Tag 54
- stringEsc-Tag 54
- stringStart-Tag 54
- tagGroup-Tag 55
- Codefragmenterweiterungen, Informationen 69
- CodeHints.xml (Datei)
 - Beschreibung 31
 - Komponenten 32, 33
- Codehinweise
 - codehints-Tag 36
 - description-Tag 37
 - Informationen 31, 69
 - JavaScript 33
 - Klassen deklarieren 34
 - menugroup-Tag 36
 - menuitem-Tag 38
 - menu-Tag 37
 - method-Tag 40
 - optionparammenu-Tag 41
 - parammenu-Tag 40
 - property-Tag 42
- Codeüberprüfung 63
- colorpicker-Tag 168
- colorRect-Attribut 171
- Colors.xml (Datei) 43
- combobox-Tag 167
- command-Attribut 100, 173
- commandButtons() 130, 153, 187
- commentEnd-Tag, Codefarben 47
- comment-Objekt 93
- commentStart-Tag, Codefarben 47
- configureSettings() 187

- confirm() 88
- copyServerBehavior() 240
- cssImport-Tag, Codefarben 47
- cssMedia-Tag, Codefarben 47
- cssProperty-Tag, Codefarben 48
- cssSelector-Tag, Codefarben 48
- css-support-Tag, Codeüberprüfung 64
- cssValue-Tag, Codefarben 48
- customText-Wert, blockStart 55

D

- data-Eigenschaft
 - von comment-Objekten 93
 - von text-Objekten 92
- Dateien
 - CodeHints.xml 32
 - insertbar.xml 103
 - menus.xml 133
 - mm_deleted_files.xml (Datei) 12
 - MMDocumentTypes.xml 15
 - SpryCodeHints.xml 33
 - String-XML 73
 - toolbars.xml 157
 - XML 88
- Dateityp, Standard ändern 7
- Datenbank-Steuerelemente 79
- Datenformatierung
 - aufrufende Funktionen 291
 - Informationen 289
- Datenquellen 275
- Datenquellenerweiterungen
 - ColdFusion-Dokumente 277
 - Informationen 68
- Datenübersetzer
 - Arten 314
 - Benutzererfahrung 313
 - Debugging 319
 - für Attribut 314
 - für Tags oder Codeblöcke 316
- Datenübersetzererweiterungen, Informationen 68
- date-Objekt 88
- defaultAttribute-Tag, Codefarben 49
- defaultTag-Tag, Codefarben 49
- defaultText-Tag, Codefarben 49
- Definitionsdatei, Dokumenttyp 15
- Deinstallieren in einer Mehrbenutzerkonfiguration 14
- deleteBehavior() 229
- deleteditems (Tag) 13
- deleteDynamicSource() 283
- deleteFormat() 292
- deleteSB() 244

- deleteServerBehavior() 240
 - delete-Tag 262
 - deleteType-Attribut 262
 - description-Tag 37
 - Dialogfelder, Anzeige anpassen 6
 - Dienstkomponente, hinzufügen 297
 - disabledImage-Attribut 170
 - displayHelp()
 - in API für Eigenschafteninspektor 207
 - in API für schwebende Bedienfelder 215
 - in API für Serververhalten 241
 - in Datenquellen-API 284
 - in Objekt-API 112
 - in Objektdateien 112
 - in Verhaltens-API 229
 - display-Tag 267
 - DOCTYPE-Anweisungen 76
 - Document Object Model
 - Baumstruktur 87
 - DOM Level 1-Spezifikation 88
 - Dreamweaver 88
 - Informationen 87
 - documentEdited() 215
 - documentElement-Eigenschaft 90
 - document-Objekt
 - DOM Level 1, Eigenschaften und Methoden 90
 - Netscape DOM, Eigenschaften und Methoden 88
 - Dokumente öffnen 21
 - Dokumenterweiterungen 19
 - Dokumentknoten 90
 - Dokumenttypen
 - Definitionsdatei 15, 16
 - Definitionsdatei, Regeln 21
 - dynamische Vorlagen 19
 - erweiterbar 15
 - Erweiterungen 19
 - Informationen 69
 - lokalisieren 16, 20
 - Tags in Definitionsdateien 16
 - Verfahren zum Öffnen von Dokumenten 21
 - DOM. *Siehe* Dokumentobjektmodell
 - domRequired-Attribut 171
 - Dreamweaver-DOM 88
 - dreamweaver-Objekt 93
 - Drittanbieter-Tags
 - Darstellung ändern 5
 - Interpretation anpassen 8
 - Markierungsfarbe ändern 11
 - tagspec 9
 - Umschreiben vermeiden 11
 - dropdown-Tag 167
 - dwscripts-Funktionen
 - applySB() 243
 - deleteSB() 244
 - findSBs() 242
 - Dynamische Daten (Dialogfeld) 275
 - dynamische Menüs
 - Beispielcode 147
 - Benutzererfahrung 144
 - dynamische Vorlagen 19
 - Dynamischer Text (Dialogfeld) 275
- E**
- editcontrol-Tag 168
 - editDynamicSource() 284
 - EDML (Extension Data Markup Language) 234
 - EDML-Dateien
 - bearbeiten 244
 - Beschreibung 234
 - EDML-Struktur 245
 - Gruppendatei-Tags 246
 - Informationen 233
 - reguläre Ausdrücke verwenden 244
 - EDML-Datei-Tags
 - Attribute 266
 - closeTag 267
 - delete 262
 - deleteType-Attribut 262
 - display 267
 - group 246
 - groupParticipant 250
 - groupParticipants 249
 - insertText 253
 - isOptional-Attribut 259
 - limitSearch-Attribut 258, 265
 - location-Attribut 253
 - name-Attribut 250
 - nodeName-Attribut 255
 - openTag 266
 - paramName-Attribut 262
 - paramNames-Attribut 258
 - participant 251
 - partType-Attribut 250
 - quickSearch 252
 - searchPatterns 255, 256, 263
 - selectParticipant-Attribut 249
 - subType-Attribut 248
 - title 249
 - translation 264
 - translations 264
 - translationType-Attribut 265
 - translator 263
 - updatePattern 260, 262
 - updatePatterns 260
 - version-Attribut 252
 - whereToSearch-Attribut 264
 - Eigenschafteninspektoren
 - *LOCKED* (Schlüsselwort) 317
 - Beispielcode 203
 - benutzerdefinierte 201
 - Benutzererfahrung 202
 - Blitzsymbol 315
 - Dateistruktur 201
 - gesperrter Inhalt für 317
 - Kommentar am Dateianfang 201
 - übersetzte Attribute 315
 - Übersicht 201
 - eigenständige Berichte 183, 184
 - Einfügeleiste
 - ändern 5
 - Definitionsdatei 96
 - Objektdateien 95
 - Objekte hinzufügen 102
 - eingebettete Werte, lokalisierbare Strings 74
 - Elementknoten 91
 - Elementtags, in Symboleleisten 165
 - enabled-Attribut 100, 171
 - endOfLineComment-Tag, Codefarben 49
 - endReporting() 186
 - entity-Tag, Codefarben 50
 - Ereignisprozeduren
 - in Erweiterungsdateien 71
 - in Verhalten-Dialogfeldern 221
 - Wert aus Ereignisprozeduren zurückgeben 222
 - Ereignisse
 - in Erweiterungsdateien 88
 - Rolle in Verhalten 221
 - Ergebnisfenster-API 183
 - Ermitteln der Browserkompatibilität 117
 - erweiterbare Dokumenttypen 15
 - Erweiterungen
 - Dreamweaver 67
 - Funktionen aktivieren 67
 - Hilfe anzeigen 72
 - Inhalt des Ordners „Shared“ 349
 - installieren 1
 - lokalisieren 73
 - neu laden 71
 - Schaltfläche zur Farbsteuerung 83

Verarbeitung von JavaScript in Dreamweaver 71

Erweiterungen neu laden 71

escape() 88

Extension Manager

Richtlinien 75

verwenden 74

Extensions.txt (Datei) 19

externe JavaScript-Dateien 71

F

Farben von

Schemenblocktrennzeichen 55

file-Attribut 100, 171

file-Objekt (Feld) 88

findDynamicSources() 285

findIssue() 119

findSBs() 242

findServerBehaviors() 241

Flash-SWF-Dateien, in Dreamweaver anzeigen 84

focus() 88

formatDynamicDataRef() 293

Formate 289

Formatliste bearbeiten, Plusmenü (+) des Dialogfelds 290

Formats.xml (Datei) 290

form-Objekt 88

FTP-Zuordnungen, ändern 14

functionKeyword-Tag, Codefarben 50

function-Objekt 88

function-Tag 39

Funktionen der Objekt-API

canInsertObject() 111

displayHelp() 112

insertObject() 113

isDOMRequired() 112

objectTag() 114

windowDimensions() 114

Funktionen, abgelöste 3

G

gemeinsame Speicherverwendung 273

generateDynamicDataRef() 285

generateDynamicSourceBindings() 286

gesperrter Inhalt, untersuchen 317

getAffectedBrowserDisplayNames() 120

getAffectedBrowserProfiles() 120

getAttribute() 91

getCodeViewDropCode() 300

getComponentChildren() 298

getContextMenuId() 299

getCurrentValue() 160, 174

getDockingSide() 216

getDynamicContent() 154, 175

getElementsByAttributeName (Eigenschaft) 91, 92

getElementsByTagName() für document-Objekte 90

von tag-Objekten 91

getFileExtensions() 308

getIssueID() 121

getIssueName() 121

getLanguageSignatures() 308

getMenuID() 176

getServerExtension() 309

getServerInfo() 310

getServerLanguages() 310

getServerModelDelimiters() 311

getServerModelDisplayName() 311
getServerModelExtDataNameUD4() 311

getServerModelFolderName() 312

getServerSupportsCharset() 312

getSetupSteps() 300

getTranslatedAttribute() 91

getTranslatorInfo() 327

getUpdateFrequency() 177

getVersionArray() 312

groupParticipants-Tag 249

groupParticipant-Tag 250

Gruppendateien

Serververhalten 234

Gruppendatei-Tags 246

H

handleDesignViewDrop() 302

handleDoubleClick() 303

hasChildNodes()

für comment-Objekte 93

für document-Objekte 90

für text-Objekte 92

von tag-Objekten 91

hasTranslatedAttributes() 91

havePreviewTarget() 149

hidden-Objekt (Feld) 88

Hilfe anzeigen 72

Hilfsfunktionen, in Verhalten 222

hline 201

HTML

Eigenschaft inner/outer 91

Standardformatierung ändern 66

I

id-Attribut 99, 169

idChar1-Tag, Codefarben 50

idCharRest-Tag, Codefarben 50

identifyBehaviorArguments() 230

ignoreCase-Tag, Codefarben 51

ignoreMMTPParams-Tag, Codefarben 51

ignoreTags-Tag, Codefarben 51

image-Attribut 99, 169

image-Objekt 88

include/-Tag 163

initialPosition() 216

initialTabs() 217

innerHTML-Eigenschaft 91

innerTag-Wert, blockStart 56

innerText-Wert, blockStart 55

insertbar.xml (Datei) 95, 103

insertbar-Tag 97

insertObject() 113

insertText-Tag 253

inspectBehavior() 231

inspectDynamicDataRef() 287

inspectFormatDefinition() 293

inspectServerBehavior() 241

inspectTag() 198

Inspektorerweiterungen, Informationen 68

Installieren von Erweiterungen 1

isATarget() 217

isAvailableInCodeView() 218

isCommandChecked() 154, 177

isdomrequired (Attribut) 136

isDOMRequired() 112, 130, 178

isLocked-Tag, Codefarben 51

isOptional-Attribut 259

isResizable() 218

item (Tag) 13

item() 88

itemref/-Tag 163

itemtype/-Tag 163

J

JavaScript

externe Dateien 71

Steuerelemente 77

URLs 71

JavaScript-Codehinweise 33

JavaScript-Interpreter

C-Level-Erweiterbarkeit 333

Datentypen 333

JS_BooleanToValue() 337

JS_DefineFunction() 334

JS_DoubleToValue() 337

JS_ExecuteScript() 340

JS_GetArrayLength() 339

JS_GetElement() 339

JS_IntegerToValue() 337

JS_NewArrayObject() 338

JS_ObjectToValue() 338

JS_ObjectType() 338

JS_ReportError() 340

JS_SetElement() 339

JS_StringToValue() 336

JS_ValueToBoolean() 336

JS_ValueToDouble() 335

JS_ValueToInteger() 335

JS_ValueToObject() 336

JS_ValueToString() 334

JSBool (Boolescher Wert) 333

JSContext (Datentyp) 333

JSNative 334

JSObject (Datentyp) 333

jsval 333

K

keywords-Tag, Codefarben 52

keyword-Tag, Codefarben 52

Knoten

DOM-Baumstruktur 87

Informationen 88

Struktursteuerelemente 88

Knotenkonstanten 88

Kompatibilitätsprüfung, Browser 117

Komponenten (Bedienfeld)

Dateien 296

erweitern 295

Strukturansicht 298

Komponentenerweiterungen, Informationen 68

Konfiguration bei mehreren Benutzern

anpassen 12

Konfigurationsordner und Erweiterungen 69

Konventionen, in diesem Handbuch 3

L

label-Attribut 170

layer-Objekt 88

limitSearch-Attribut 258, 265

liveDataTranslateMarkup() 330

location-Attribut 253

LOCKED (Schlüsselwort) 317

lokalisieren 73

lokalisierte Strings 16

M

math-Objekt 88

Mehrbenutzerplattformen, Konfigurationsordner 19, 341

Mehrbenutzerumgebung, Konfiguration
Konfigurationsdateien löschen 12
neu installieren und deinstallieren 14

Ordner 70, 341

menu_ID-Attribut 171

menubar (Tag) 134

Menübefehle

Beispielcode 144

Benutzererfahrung 144

Beschreibung 142

Erweiterungen, Informationen 67

Menübefehle, ändern 143

menubutton-Tag 97, 166

menugroup-Tag 36

menuitem-Tag 38, 135

Menüordner, Befehlsdatei ablegen 147

Menüs

ändern 5, 140

Befehle 143

Informationen 69

Popup- und Kontextmenüs
ändern 142

menus.xml (Datei)

action-Tag 138

activate-Tag 139

ändern 140

Beschreibung 133

menubar (Tag) 134

menuitem-Tag 135

menu-Tag 134

override-Tag 139

separator (Tag) 136

shortcut (Tag) 137

shortcutlist (Tag) 137

tool-Tag 138

menu-Tag 37, 134

method-Tag 40

MM:TREECOLUMN-Tag 81

MM:TREENODE-Tag 81

MM_ConfigFileExists() 343

MM_CreateConfigFolder() 346

MM_DeleteConfigFile() 347

mm_deleted_files.xml (Datei)

deleteditems (Tag) 13

Informationen 12

item (Tag) 13

Tag-Syntax 13

MM_GetConfigFileAttributes() 344

MM_GetConfigFolderList() 342

mm_jsapi.h (Datei)

Beispiel 347

einschließen 333

MM_OpenConfigFile() 343

MM_RemoveConfigFolder() 346

MM_returnValue 222

MM_SetConfigFileAttributes() 345

MMDocumentTypes.xml (Datei) 15

N

name-Attribut 101, 250

nameTagScript-Wert, blockStart 57

nameTag-Wert, blockStart 56

navigator-Objekt 88

neu Installieren in einer
Mehrbenutzerkonfiguration 14

neu laden, Erweiterungen 71

nextSibling (Eigenschaft) 88, 90, 91, 92

Node.COMMENT_NODE 88

Node.DOCUMENT_NODE 88

Node.ELEMENT_NODE 88

Node.TEXT_NODE 88

odelist-Objekt 88

nodeName-Attribut 255

nodeType-Eigenschaft

von comment-Objekten 93

von document-Objekten 90

von tag-Objekten 91

von text-Objekten 92

number-Objekt 88

numbers-Tag, Codefarben 52

O

object-Objekt 88

objectTag() 114

Objekte

erstellen 95

Flash-SWF-Dateien hinzufügen 84

Funktion von Dateien 103

Komponenten 95

zur Einfügeleiste hinzufügen 102

Objekte der Einfügeleiste

Beispiel 103

Erweiterungen, Informationen 67

Reihenfolge ändern 102

Öffnen von Dokumenten 21

onBlur-Ereignis 88

onChange-Ereignis 88

- onClick-Ereignis 88
 - onFocus-Ereignis 88
 - onLoad-Ereignis 88
 - onMouseOver-Ereignis 88
 - onResize-Ereignis 88
 - openTag-Attribut 266
 - operators-Tag, Codefarben 52
 - option-Objekt 88
 - optionparammenu-Tag 41
 - Ordner „Shared“
 - Inhalt 349
 - Inhalt des Ordners „Common“ 349
 - zusätzliche Ordner 353
 - outerHTML-Eigenschaft 91
 - outerTag-Wert, blockStart 56
 - overImage-Attribute 170
 - override-Tag 139
- P**
- panelset (Tag) 22
 - parammenu-Tag 40
 - paramName-Attribut 262
 - paramNames-Attribut 258
 - parentNode-Eigenschaft
 - von comment-Objekten 93
 - von document-Objekten 90
 - von tag-Objekten 91
 - von text-Objekten 92
 - parentWindow-Eigenschaft 90
 - participant-Tag 251
 - partType-Attribut 250
 - password-Objekt (Feld) 88
 - pasteServerBehavior() 242
 - previousSibling (Eigenschaft) 88, 90, 91, 92
 - Problem-API 119
 - processFile() 186
 - property-Tag
 - Codeüberprüfung 64
 - Informationen 42
- Q**
- quickSearch-Tag 252, 268
- R**
- radiobutton-Tag 166
 - radio-Objekt 88
 - receiveArguments() 131, 151, 155, 160, 179
 - regex-Objekt 88
 - regex-Tag, Codefarben 53
 - reguläre Ausdrücke in EDML-Dateien 244
 - removeAttribute() 91
 - reset-Objekt 88
 - resizeTo() 88
- S**
- sampleText-Tag, Codefarben 53
 - Schaltfläche zur Farbsteuerung 83
 - Schemaverarbeitung
 - Codefarben 57
 - Escape-Zeichen 58
 - Maximale Stringlänge 59
 - Platzhalterzeichen 58
 - Rangfolge 59
 - schema-Tag, Codefarben 44
 - schwebende Bedienfelder
 - Beispiele 210
 - Benutzererfahrung 209
 - Erweiterungen, Informationen 68
 - Performance-Fragen 219
 - Script Editor (Erweiterung)
 - JavaScript-Code für 212
 - schwebendes Bedienfeld 210
 - searchPatterns-Tag 255, 256, 263
 - searchPattern-Tag, Codefarben 54
 - Seitendesigns 6
 - select() 88
 - selection, exact und within 201
 - selectionChanged() 212, 218
 - select-Objekt 88
 - selectParticipant-Attribut 249
 - separator (Tag) 136
 - separator-Tag 98, 164
 - Servermodelle
 - Beschreibung 307
 - Erweiterungen, Informationen 68
 - Serververhalten
 - aktualisieren 272
 - Beispiel 235
 - dwscripits-Funktionen 242
 - Erweiterung 233
 - Erweiterungen, Informationen 68
 - Gruppendateien 234
 - Instanz 233
 - Laufzeitcode 233
 - löschen 273
 - suchen 268
 - Suchmuster-Auflösung 271
 - Techniken 268
 - Teilnehmer 233
 - Teilnehmerdateien 234
 - Übersicht 233
 - setAttribute() 91
 - setInterval() 88
 - setMenuText() 156
 - setTimeout() 88, 219
 - setupStepsCompleted() 302
 - shortcut (Tag) 137
 - shortcutlist (Tag) 137
 - showIf() 180
 - showIf-Attribut 99, 169
 - Shutdown (Ordner) 70
 - Site-Berichte
 - Informationen 181
 - Workflow 181
 - Site-Objekt, Eigenschaften 93
 - Skriptsymbol 218
 - Sprachinformationen 93
 - Spry-Framework, Codehinweise 31
 - Standard-Dateityp ändern 7
 - Standarddokumente anpassen 6
 - Startbefehle 70
 - Startup (Ordner) 70
 - String-Dateien 73
 - stringEnd-Tag, Codefarben 54
 - stringEsc-Tag, Codefarben 54
 - string-Objekt 88
 - Strings, lokalisierbar 74
 - stringStart-Tag, Codefarben 54
 - Strukturansicht, XML 88
 - Struktursteuerelement bearbeiten 83
 - Struktursteuerelemente
 - erstellen 81
 - hinzufügen 79
 - Informationen 79
 - Inhalt bearbeiten 83
 - submit-Objekt 88
 - subType-Attribut 248
 - Suchmuster-Auflösung 271
 - Symbolleisten
 - andocken 158
 - API für Befehle 174
 - Befehlsverhalten 158
 - button-Tag 165
 - checkboxbutton-Tag 165
 - colorpicker-Tag 168
 - combobox-Tag 167
 - Dateidefinition 160
 - dropdown-Tag 167
 - editcontrol-Tag 168
 - einfache Befehlsdatei 159
 - Elementtags 165
 - erstellen 157
 - include/-Tag 163
 - itemref/-Tag 163
 - itemtype/-Tag 163
 - menubutton-Tag 166

- radiobutton-Tag 166
- separator-Tag 164
- Steuerelemente 157
- tag-Attribut 161, 169
- toolbars.xml (Datei) 157
- toolbar-Tag 161
- Verhalten:Informationen 157
- Symbolleistenenerweiterungen,
Informationen 68
- systemScript-Eigenschaft 93

T

- tag-Attribut 101
- Tag-Auswahl 193
- Tag-Bibliotheken 190
- Tag-Dialogfelderweiterungen,
Informationen 68
- Tag-Editor
 - Beispiele 195
 - erstellen 195, 198
- tagGroup-Tag, Codefarben 55
- tagName-Eigenschaft 91
- tag-Objekt 91
- tagspec (Tag) 9
- Tastaturkurzbefehle
 - ändern 141
 - Zuordnungsdateien 27
- Teilnehmer und Serververhalten 233
- Teilnehmerdateien 234
- textarea-Objekt 88
- TextNode 92
- text-Objekt (Feld) 88
- text-Objekte 92
- title-Tag 249
- toolbarControls() 304
- toolbars.xml (Datei) 157, 160
- toolbar-Tag 161
- tool-Tag 138
- tooltip-Attribut 170
- translateDOM() 328
- translateMarkup() 329
- translations-Tag 264
- translation-Tag 264
- translationType-Attribut 265
- translator-Tag 263
- TREECOLUMN-Tag 81
- TREENODE-Tag 81

U

- Übersetzer
 - Attribut 314
 - Block/Tag 316
 - Debugging 319

- übersetzte Attribute
 - einzelne 314
 - in Tags suchen 91
 - mehrere 315
 - untersuchen 315
- übersetzte Tags, untersuchen 317
- unescape() 88
- update-Attribut 172
- updatePatterns-Tag 260
- updatePattern-Tag 260, 262
- updateScript() 213
- URL-Eigenschaft 90

V

- validateTag() 199
- value-Attribut 172
- value-Tag, Codeüberprüfung 65
- variable Tabellen-Steuerelemente 80
- VBScript 221
- Verhalten
 - API 226
 - Beispielcode 222
 - Benutzererfahrung 221
 - erforderliche Funktionen 226
 - Hilfsfunktionen 222
 - mehrere Funktionen einfügen 222
- Verhaltens-API-Funktionen
 - applyBehavior() 226
 - behaviorFunction() 227
 - canAcceptBehavior() 228
 - deleteBehavior() 229
 - displayHelp() 229
 - eigenständige Berichte 183
 - identifyBehaviorArguments() 230
 - inspectBehavior() 231
 - windowDimensions() 232
- Verhaltensenerweiterungen,
Informationen 68
- version-Attribut 252
- vline 201

W

- W3C 88
- whereToSearch-Attribut 264
- width-Attribut 170
- window.close() 88
- windowDimensions()
 - in Befehls-API 131
 - in Berichts-API 188
 - in Menübefehlen 156
 - in Objekt-API 114
 - in Verhaltensaktionen 232
- window-Objekt 88

X

- XML
 - Strukturansicht 88
- XML-
 - Struktur 245
- XML-Dateien
 - CodeHints.xml 32
 - Formats.xml 290
 - Informationen 88
 - insertbar.xml 103
 - menus.xml 133
 - MMDocumentTypes.xml 15
 - SpryCodeHints.xml 33
 - String 73
 - toolbars.xml 157
- XML-Tag
 - Symbolleiste 161
- XML-Tags
 - codehints 36

Z

- Zielbrowser, Codeüberprüfung 63
- Zuordnungsdateien,
Tastaturkurzbefehle 27