Erweitern von ADOBE® DREAMWEAVER® CS5 & CS5.5

Rechtliche Hinweise

Letzte Aktualisierung 22.6.2011

Inhalt

Kapitel 1: Einführung

Im Handbuch *Erweitern von Dreamweaver CS5* werden das Adobe® Dreamweaver® CS5-Framework und die API-Programmierschnittstelle (Application Programming Interface) beschrieben, mit denen Sie Erweiterungen für Dreamweaver erstellen können. Das Handbuch *Erweitern von Dreamweaver CS5* enthält Informationen zu folgenden Themen:

- Funktionsweise der einzelnen Erweiterungstypen
- API-Funktionen, die Dreamweaver zum Implementieren der verschiedenen Objekte aufruft
- Verschiedene Funktionsmerkmale von Dreamweaver, darunter u. a. Menüs, schwebende Bedienfelder und Serververhalten
- Einfaches Beispiel für jeden Erweiterungstyp
- Anpassen von Dreamweaver durch Bearbeiten von Tags in verschiedenen HTML- und XML-Dateien, um Befehle oder Dokumenttypen hinzuzufügen

Informationen zum Dienstprogramm und zu allgemeinen JavaScript™-APIs, die Sie für verschiedene Hilfsvorgänge in Ihren Dreamweaver-Erweiterungen einsetzen können, finden Sie im *Dreamweaver API-Referenzhandbuch*. Wenn Sie Erweiterungen erstellen möchten, die mit Datenbanken verwendet werden können, finden Sie entsprechende Informationen in den Kapiteln zum Herstellen von Verbindungen mit Datenbanken im Handbuch *Dreamweaver verwenden*.

Erweiterungen

Die meisten Dreamweaver-Erweiterungen sind in HTML und JavaScript programmiert. 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 (Dynamic Link Libraries) in C erstellen und verwenden können. Wenn Sie Erweiterungen zum Erstellen von Webanwendungen programmieren, sollten Sie auch mit serverbasierten Skripts auf mindestens einer Plattform vertraut sein, z. B. Active Server Pages (ASP), ASP.NET, PHP: Hypertext Preprocessor (PHP), Adobe® ColdFusion® oder Java Server Pages (JSP).

Installieren von Erweiterungen

Um sich mit dem Programmieren von Erweiterungen vertraut zu machen, sollten Sie die Erweiterungen und Ressourcen testen, die auf der Adobe Exchange-Website unter (http://www.adobe.com/go/exchange_de) zur Verfügung stehen. Durch die Installation einer vorhandenen Erweiterung lernen Sie zahlreiche Tools kennen, die Sie bei der Verwendung benutzerdefinierter Erweiterungen benötigen.

1. Laden Sie den Adobe® Extension Manager auf der Adobe Download-Website unter http://www.adobe.com/go/downloads_de herunter und installieren Sie ihn.
2. Melden Sie sich auf der Adobe Exchange-Website unter http://www.adobe.com/go/exchange_de an.
3. Wählen Sie eine der verfügbaren Erweiterungen 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“ aus. Wählen Sie in Dreamweaver die Option „Befehle“ > „Erweiterungen verwalten“ aus, um den Extension Manager zu starten.

Der Extension Manager installiert die Erweiterung aus dem Ordner „Downloaded Extensions“ automatisch in Dreamweaver.

In einigen Fällen muss Dreamweaver neu gestartet werden, bevor Sie auf eine Erweiterung zugreifen können. Wenn Dreamweaver während der Installation der Erweiterung ausgeführt wird, werden Sie unter Umständen 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 zur entsprechenden Erweiterung einzusehen.

Erstellen von Erweiterungen

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

- Legen Sie den Typ der zu erstellenden Erweiterung fest. Weitere Informationen zu den Erweiterungstypen finden Sie unter „Typen von Dreamweaver-Erweiterungen“ auf Seite 80.
- 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.
- Definieren Sie gegebenenfalls die Benutzeroberfläche für die Erweiterung.
- Erstellen Sie die erforderlichen 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 „Arbeiten mit dem Extension Manager“ auf Seite 87.

Weitere Ressourcen für die Programmierung von Erweiterungen

Wenn Sie sich mit anderen Entwicklern austauschen möchten, die Erweiterungen erstellen, können Sie der Dreamweaver Extensibility Newsgroup beitreten. Die Adobe-Website dieser Newsgroup finden Sie (in englischer Sprache) unter
http://www.adobe.com/cfusion/webforums/forum/categories.cfm?forumid=12&catid=190&entercat=y.

Neue Funktionen in Dreamweaver CS5

Neue Funktionen

Jede dieser Funktionsgruppen wurde mit neuen Teilfunktionen ausgestattet, die der Dienstprogramm-API und der JavaScript-API hinzugefügt wurden. Informationen zu den neuen Funktionen finden Sie unter Neue Funktionen in Dreamweaver CS5.

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

Veraltete Funktionen
Verschiedene Dreamweaver-Funktionen sind mit dieser Version als veraltet eingestuft. Informationen zu den Funktionen, die aus den Dienstprogramm- und JavaScript-APIs entfernt wurden, finden Sie im *Dreamweaver API-Referenzhandbuch*.

Konventionen in diesem Handbuch

In diesem Handbuch werden die folgenden typografischen Konventionen verwendet:

- `Codeschrift` kennzeichnet Codefragmente und API-Literale, z. B. Klassennamen, Methodennamen, Funktionsnamen, Typnamen, Skripts, SQL-Anweisungen, HTML- und XML-Tag-Namen sowie Attributnamen.
- *`Kursive Codeschrift`* kennzeichnet Platzhalterelemente im Code.
- Das Fortsetzungssymbol (¬) gibt an, dass sich eine lange Codezeile über mehrere Handbuchzeilen erstreckt. 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.` (z. B. `dreamweaver.`*`Funktionsname`*) können beim Programmieren von Code durch `dw.`*`Funktionsname`* abgekürzt werden. In diesem Handbuch wird das vollständige Präfix `dreamweaver.` bei der Definition der Funktion und im Index verwendet. In vielen Beispielen wird jedoch das kürzere Präfix `dw.` verwendet.

  Folgende Benennungskonventionen werden in diesem Handbuch verwendet:

- Sie – Person, die Erweiterungen programmiert, d. h. der Entwickler
- Benutzer – Person, die Dreamweaver verwendet
- Besucher – Person, die die vom Benutzer erstellte Webseite anzeigt

Kapitel 2: Anpassen von Dreamweaver

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

Möglichkeiten zur Anpassung von Dreamweaver

Dreamweaver kann auf verschiedenste Weise angepasst werden. Einige der Vorgehensweisen werden in *Dreamweaver verwenden* erläutert. Sie können für eine Vielzahl von Bereichen Voreinstellungen festlegen. Dazu gehören Eingabehilfen, Codefarben, Schriften, Markierungen und die Vorschau in einem Browser, die Sie über das Bedienfeld „Voreinstellungen“ („Bearbeiten“ > „Voreinstellungen“ oder „Dreamweaver“ > „Voreinstellungen“ [Mac OS X]) definieren können. Außerdem können Sie mithilfe des Tastaturkurzbefehl-Editors („Bearbeiten“ > „Tastaturkurzbefehle“) oder durch Bearbeiten einer Konfigurationsdatei Tastenkombinationen ändern.

Anpassen von Standarddokumenten

Der Ordner „DocumentTypes/NewDocuments“ enthält ein (leeres) Standarddokument für jeden Dokumenttyp, den Sie in Dreamweaver erstellen können. Wenn Sie ein neues, leeres Dokument erstellen, indem Sie „Datei“ > „Neu“ und anschließend eine Option in der Kategorie „Einfache Seite“, „Dynamische Seite“ oder „Andere“ auswählen, erstellt Dreamweaver das neue Dokument basierend auf dem entsprechenden Standarddokument in diesem Ordner. Um die Anzeige eines Standarddokuments für einen bestimmten Dokumenttyp zu ändern, bearbeiten Sie das entsprechende Dokument in diesem Ordner.

Hinweis: *Wenn alle Seiten Ihrer Site gemeinsame Elemente (z. B. einen Urheberrechtshinweis) oder ein gemeinsames Layout aufweisen sollen, empfiehlt es sich, Vorlagen und Bibliothekselemente zu verwenden, anstatt die Standarddokumente zu ändern. Weitere Informationen zu Vorlagen und Bibliothekselementen finden Sie im Handbuch „Verwenden von Dreamweaver“.*

Anpassen von Seitendesigns

Dreamweaver enthält eine Reihe vordefinierter CSS-Vorlagen, Framesets und Seitendesigns. Sie können Seiten erstellen, die auf diesen Designs basieren, indem Sie „Datei“ > „Neu“ auswählen.

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 zu 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 jeweiligen Ordner „_notes“) für die entsprechende Seitendesign-Datei.

Anpassen der Anzeige von Dialogfeldern

Die Layouts der Dialogfelder für Objekte, Befehle und Verhalten sind als HTML-Formulare angegeben. Sie befinden sich in HTML-Dateien im Unterordner „Configuration“ des Dreamweaver-Anwendungsordners. 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 die Kopien der Konfigurationsdateien in Ihrem Benutzer-Konfigurationsordner bearbeiten, anstatt die Dreamweaver-Konfigurationsdateien zu bearbeiten. Weitere Informationen finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 83.*

Ändern der Anzeige von Dialogfeldern

1. Wählen Sie in Dreamweaver „Bearbeiten“ > „Voreinstellungen“ und anschließend die Kategorie „Codeumschreibung“ aus.
2. Deaktivieren Sie die Option „Formularelemente beim Einfügen umbenennen“.

   Durch Deaktivieren dieser Option wird gewährleistet, dass die Formularelemente beim Kopieren und Einfügen ihren ursprünglichen Namen beibehalten.
3. Klicken Sie auf „OK“, um das Dialogfeld „Voreinstellungen“ 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 bei Bedarf später 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“ übernimmt beispielsweise die Texteingabe aus einem Textbereich in einem Dialogfeld 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 in der Datei „Comment.htm“ im Ordner „Configuration/Objects/Invisibles“. 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.

Ändern des Standarddateityps

Dreamweaver zeigt im Dialogfeld „Öffnen“ („Datei“ > „Öffnen“) standardmäßig alle kompatiblen Dateitypen an. Mithilfe eines Popupmenüs 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 im Dialogfeld „Datei“ > „Öffnen“ alle Dateitypen angezeigt werden sollen (auch Dateien, die nicht mit Dreamweaver geöffnet werden können), wählen Sie „Alle Dateien (*.*). Dies unterscheidet sich von der Option „Alle Dokumente“, bei der nur die Dateien anzeigt werden, die mit Dreamweaver geöffnet werden können.*

Ändern des Standarddateityps, 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 einem Texteditor.
3. Schneiden Sie die Zeile mit der gewünschten neuen Standardeinstellung aus. Fügen Sie sie dann am Anfang der Datei ein, sodass diese Zeile an erster Stelle in der Datei steht.
4. Speichern Sie die Datei.
5. Starten Sie Dreamweaver neu.

Um die neue Standardeinstellung zu testen, wählen Sie „Datei“ > „Öffnen“ und überprüfen dann das Popupmenü für die Dateitypen.

Hinzufügen neuer Dateitypen zum Menü im Dialogfeld „Datei“ > „Öffnen“

1. Erstellen Sie eine Sicherungskopie der Datei „Extensions.txt“ im Ordner „Configuration“.
2. Öffnen Sie die Datei „Extensions.txt“ in einem Texteditor.
3. Fügen Sie für jeden neuen Dateityp eine neue Zeile hinzu. Geben Sie, durch Kommas getrennt, die für den neuen Dateityp möglichen Dateierweiterungen in Großbuchstaben ein. Geben Sie dann einen Doppelpunkt sowie eine kurze Beschreibung ein, die im Dialogfeld „Datei“ > „Öffnen“ im Popupmenü für die Dateitypen angezeigt werden soll.

   Geben Sie beispielsweise für JPEG-Dateien `JPG,JPEG,JFIF:JPEG-Bilddateien` ein.
4. Speichern Sie die Datei.
5. Starten Sie Dreamweaver neu.

Um die Änderungen zu überprüfen, wählen Sie „Datei“ > „Öffnen“ aus und klicken auf das Popupmenü für die Dateitypen.

Anpassen der Interpretation von Drittanbieter-Tags

Serverseitige Technologien wie ASP, Adobe ColdFusion, JSP und PHP verwenden speziellen HTML-fremden Code in HTML-Dateien. Webserver erstellen auf Basis dieses Codes die HTML-Inhalte und stellen sie bereit. Wenn Dreamweaver Nicht-HTML-Tags erkennt, werden diese mit den Informationen in den Dateien für Drittanbieter-Tags verglichen, in denen festgelegt ist, wie Dreamweaver Nicht-HTML-Tags auswertet 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 benutzerdefinierter Tag-Datenbankdateien festlegen, wie Dreamweaver Tags auswertet 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 benutzerdefinierter Tags in Dreamweaver definiert wird. Es wird jedoch nicht beschrieben, wie der Inhalt oder die Eigenschaften benutzerdefinierter Tags bearbeitet werden können. Weitere Informationen über das Erstellen eines Eigenschafteninspektors zum Überprüfen und Ändern der Eigenschaften von benutzerdefinierten Tags finden Sie unter „Eigenschafteninspektoren“ auf Seite 228.*

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 auswerten und verarbeiten kann. Tag-Datenbankdateien haben die Dateierweiterung „.xml“.

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

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

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

Mithilfe von `tagspec` können Sie zwei Tag-Typen definieren:

- Normale Tags im HTML-Stil

  Im 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 (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 handeln würde, 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 verfügt über kein schließendes Tag.

Im Folgenden werden die Attribute und gültigen Werte für das Tag `tagspec` beschrieben. 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 gekennzeichnet sind, sind obligatorisch.

<tagspec>

Beschreibung
Enthält Informationen über ein 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 prüfen können, müssen in der ersten Zeile den Eintrag `*ASP*` aufweisen. Informationen über die Eigenschafteninspektor-API finden Sie unter „Eigenschafteninspektoren“ auf Seite 228.
- `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 beim 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 Inhaltstypen das Tag enthalten kann und wo in einer HTML-Datei das Tag 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, beispielsweise `div` und `p`. Dieses Tag kann nur im body-Bereich oder innerhalb anderer body-Tags (z. B. `div`, `layer` oder `td`) auftreten.
  - `head_model` gibt an, dass das Tag Text beinhalten und nur im head-Bereich 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` gekennzeichnet 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 darf. Einfacher Text kann beispielsweise nicht (außerhalb eines gültigen head-Elements) im head-Bereich eines Dokuments auftreten. Sie können ein `marker_model`-Tag mit einfachem Text daher nicht im head-Bereich platzieren. (Um ein benutzerdefiniertes Tag mit einfachem Text im head-Bereich 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 `p` oder `div`, z. B. innerhalb eines Absatzes). Wenn das 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-Kennzeichnungen (z. B. bestimmte ColdFusion-Tags) verwendet, die Dreamweaver nicht analysieren soll.
- `start_string` gibt ein Trennzeichen an, das den Anfang eines zeichenfolgenbegrenzten Tags kennzeichnet. Zeichenfolgenbegrenzte Tags dürfen an jeder Stelle in einem Dokument vorkommen, an der auch ein Kommentar zulässig ist. Tags oder Entities bzw. URLs zwischen den Tags `start_string` und `end_string` werden von Dreamweaver nicht analysiert und nicht dekodiert. 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 ggf. Anführungszeichen ("). Wenn 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"` (Standardwert) gesetzt ist, analysiert Dreamweaver die Attribute. Wenn es auf `"false"` gesetzt ist, ignoriert Dreamweaver alle Elemente, bis die nächste schließende spitze Klammer außerhalb von Anführungszeichen auftritt. Dieses Attribut sollte beispielsweise auf `"false"` gesetzt werden für ein Tag wie `cfif` (z. B. in `<cfif a is 1>`, das von Dreamweaver nicht als Gruppe von Attributname-Wert-Paaren 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 ist `"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 angewendet wird, 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 Dreamweaver auf einer JSP-Seite auf Code mit entsprechenden Trennzeichen trifft, wird daher ein JSP-Symbol angezeigt. Wenn Dreamweaver auf diesen Code auf einer Seite trifft, die keine JSP-Seite ist, wird ein ASP-Symbol angezeigt.

Inhalt

Kein (leeres Tag).

Container

Keine.

Beispiel

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

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 `tagspec` ab. Wenn `tag_type` den Wert `"empty"` aufweist, wird das im Attribut `icon` angegebene Symbol angezeigt. Wenn `tag_type` den Wert `"nonempty"` aufweist, `render_contents` jedoch den Wert `"false"`, wird das Symbol wie für ein leeres Tag angezeigt. 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 dargestellt. Stattdessen werden das Anfangs- und End-Tag und der 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 (z. B. der Text zwischen den Tags in `<mytag>This is the content between the opening and closing tags</mytag>`). Wenn „Ansicht“ > „Unsichtbare Elemente“ aktiviert ist, wird der Inhalt in der Farbe markiert, die in den Markierungsvoreinstellungen für Drittanbieter-Tags festgelegt ist. (Markierungen werden nur auf Tags angewendet, die in Tag-Datenbankdateien definiert sind.)

Ändern der Markierungsfarbe für Drittanbieter-Tags

1. Wählen Sie „Bearbeiten“ > „Voreinstellungen“ und dann 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*.

Vermeiden von automatischen Tag-Korrekturen für Drittanbieter-Tags

Dreamweaver berichtigt bestimmte Fehlertypen im HTML-Code. Ausführliche Informationen hierzu finden Sie in *Dreamweaver verwenden*. Standardmäßig ändert Dreamweaver 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 die Codekorrektur ändern, sodass HTML-Code beim Öffnen der oben genannten Dateien korrigiert wird. Sie können zudem weitere Dateitypen hinzufügen, die Dreamweaver nicht automatisch ändern soll.

Dreamweaver kodiert bestimmte Sonderzeichen, indem diese durch numerische Werte ersetzt werden, 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 verschiedene Plattformen und Browser hinweg korrekt angezeigt werden. Sie können das Dreamweaver-Kodierungsverhalten jedoch ändern, da es bei Verwendung der entsprechenden Dateien unter Umständen Auswirkungen auf Drittanbieter-Tags haben kann.

Zulassen der automatischen Korrektur von HTML-Code in weiteren Dateitypen

1. Wählen Sie „Bearbeiten“ > „Voreinstellungen“ aus 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 in 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.)

Hinzufügen von Dateitypen, die Dreamweaver nicht automatisch korrigieren soll

1 Wählen Sie „Bearbeiten“ > „Voreinstellungen“ aus 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. Ausführliche Informationen finden Sie unter „Ändern des Standarddateityps“ auf Seite 5.

Deaktivieren der Dreamweaver-Kodierungsoptionen

1 Wählen Sie „Bearbeiten“ > „Voreinstellungen“ aus und klicken Sie dann auf die Kategorie „Codeumschreibung“.

2 Deaktivieren Sie eine oder beide Sonderzeichenoptionen.

Informationen zu weiteren Einstellungen für die Codeumschreibung finden Sie in *Dreamweaver verwenden*.

Anpassen von Dreamweaver in einer Mehrbenutzerumgebung

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. Wenn Sie Dreamweaver das erste Mal in einem Mehrbenutzer-Betriebssystem ausführen, werden die Konfigurationsdateien in den Ordner „Configuration“ des Benutzers kopiert. Wenn Sie Dreamweaver mithilfe von Dialog- und Bedienfeldern anpassen, werden Ihre Benutzer-Konfigurationsdateien und nicht die Konfigurationsdateien von Dreamweaver geändert. Um Dreamweaver in einer Mehrbenutzerumgebung anzupassen, bearbeiten Sie anstelle der Konfigurationsdateien von Dreamweaver die Konfigurationsdatei des jeweiligen Benutzers. Bearbeiten Sie eine Dreamweaver-Konfigurationsdatei, um Änderungen vorzunehmen, die Auswirkungen für fast alle Benutzer haben. Benutzern, für die bereits spezifische Konfigurationsdateien vorhanden sind, werden diese Änderungen jedoch nicht angezeigt. Um Änderungen vorzunehmen, die Auswirkungen auf alle Benutzer haben, erstellen Sie eine Erweiterung und installieren Sie sie mit dem Extension Manager.

Hinweis: *In älteren Mehrbenutzer-Betriebssystemen (Windows 98, Windows Me und Mac OS 9.x) teilen sich alle Benutzer dieselben Dreamweaver-Konfigurationsdateien.*

Der Speicherort des Ordners „Configuration“ für einzelne Benutzer richtet sich nach dem Betriebssystem.

Bei Windows XP befindet sich der Ordner unter:

Festplatte:\Dokumente und Einstellungen\`Benutzername`\Anwendungsdaten\Adobe\Dreamweaver CS5\Configuration

Hinweis: *Dieser Ordner befindet sich unter Umständen in einem versteckten Ordner.*

Bei Windows Vista befindet sich der Ordner unter:

Festplatte:\Benutzer\`Benutzername`\AppData\Roaming\Adobe\Dreamweaver CS5\Configuration

Für Mac OS X-Plattformen gilt der folgende Speicherort:

Festplatte:/Benutzer/`Benutzername`/Library/Application Support/Adobe/Dreamweaver CS5/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 Benutzer-Konfigurationsordner kopiert. (Die zu kopierenden Dateien sind im Ordner „Configuration" in der Datei „version.xml" angegeben.) Wenn Sie Dreamweaver innerhalb der Anwendung anpassen, kopiert Dreamweaver die Konfigurationsdateien in Ihren Benutzer-Konfigurationsordner. Beispielsweise werden die Dateien kopiert, wenn Sie eines der vorentwickelten Codefragmente im Bedienfeld „Codefragmente" ändern. 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 anpassen zu können, muss sie sich im Benutzerordner „Configuration" befinden. Wenn die Datei von Dreamweaver nicht bereits kopiert wurde, kopieren Sie sie in den Benutzerordner „Configuration" und bearbeiten Sie sie dort.

Löschen von Konfigurationsdateien in einer Mehrbenutzerumgebung

Wenn Sie unter einem Mehrbenutzer-Betriebssystem in Dreamweaver einen Vorgang durchführen, durch den eine Konfigurationsdatei gelöscht wird (z. B. durch Löschen eines vorentwickelten Codefragments aus dem Bedienfeld „Codefragmente"), erstellt Dreamweaver in Ihrem Benutzer-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 existiert.

Deaktivieren einer 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 neu.

Tag-Syntax in der Datei „mm_deleted_files.xml"

Die Datei „mm_deleted_files.xml" enthält eine strukturierte Liste von Einträgen mit den von Dreamweaver zu ignorierenden Konfigurationsdateien. Diese Elemente werden durch XML-Tags angegeben, die Sie in einem Texteditor bearbeiten können.

In den Syntaxbeschreibungen der im Folgenden aufgeführten in „mm_deleted_files.xml" verwendeten Tags sind optionale Attribute in den Attributlisten durch geschweifte Klammern ({}) markiert. Nicht durch geschweifte Klammern gekennzeichnete Attribute 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
Keine.

Beispiel

```
<deleteditems>
<!-- item tags here -->
</deleteditems>
```

<item>

Beschreibung
Gibt eine Konfigurationsdatei an, die Dreamweaver ignorieren soll.

Attribute
`name`

Das Attribut `name` gibt den Pfad zur Konfigurationsdatei relativ zum Ordner „Configuration“ an. Verwenden Sie unter Windows einen umgekehrten Schrägstrich (\), um die einzelnen Teile des Pfads voneinander zu trennen. Verwenden Sie auf dem Macintosh® einen Doppelpunkt (:).

Inhalt
Kein (leeres Tag).

Container
Dieses Tag muss sich innerhalb eines `deleteditems`-Tags befinden.

Beispiel

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

Neuinstallieren und Deinstallieren von Dreamweaver in einer Mehrbenutzerumgebung

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

Ändern von FTP-Zuordnungen

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 (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 möchten, müssen Sie einen anderen FTP-Client verwenden.*

Im folgenden Beispiel ist eine Zeile (aus der Macintosh-Datei) dargestellt, 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 sind in Großbuchstaben angegeben.

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

Hinzufügen von Informationen über eine neue Dateierweiterung

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 Tabstoppzeichen, den Dateityp und ein weiteres Tabstoppzeichen 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 verschiedene XML-Schemas zum Strukturieren der Informationen über Serververhalten, Tags und Tag-Bibliotheken, Komponenten, Dokumenttypen und Verweise.

Wenn Sie in Dreamweaver Erweiterungen erstellen und verwenden, ergeben sich viele Gelegenheiten, zum Verwalten der in den Erweiterungen 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.

Definitionsdateien für Dokumenttypen

Die zentrale Komponente erweiterbarer Dokumenttypen ist die Definitionsdatei für Dokumenttypen. 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, Codefarbstil, Beschreibungen usw. angegeben.

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

Dreamweaver enthält eine anfängliche Definitionsdatei für Dokumenttypen. 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	Dynamic	aspx, ascx	
ASP.NET VB	ASP.NET-VB	Dynamic	aspx, ascx	
ASP JavaScript	ASP-JS	Dynamic	asp	
ASP VBScript	ASP-VB	Dynamic	asp	
ColdFusion	ColdFusion	Dynamic	cfm, cfml	UltraDev 4 ColdFusion
ColdFusion-Komponente		Dynamic	cfc	
JSP	JSP	Dynamic	jsp	
PHP	PHP	Dynamic	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	
CSS		Text	css	
Java		Text	java	
JavaScript		Text	js	
VB		Text	vb	

Dokumenttyp	Servermodell	Interner Typ	Dateierweiterungen	Bisheriges Servermodell
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 möchten, können Sie entweder der von Adobe bereitgestellten Definitionsdatei („MMDocumentTypes.xml“) Ihren Eintrag oder dem Ordner „Configuration/DocumentTypes“ Ihre benutzerdefinierte Definitionsdatei hinzufügen.

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

Aufbau der Definitionsdateien für Dokumenttypen

Im folgenden Beispiel ist eine typische Definitionsdatei für Dokumenttypen dargestellt:

```
<?xml version="1.0" encoding="utf-8"?>
<documenttypes     xmlns:MMString="http://www.adobe.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 Codefarben für die Dokumenttypen sind in den XML-Dateien im Ordner „Configuration/CodeColoring“ angegeben.*

In diesem Beispiel kennzeichnet das Element `loadstring` die lokalisierten Strings, die Dreamweaver für Titel und Beschreibung der Dokumente vom Typ ASP-JS verwendet. Weitere Informationen über lokalisierte Strings finden Sie unter „Bereitstellen lokalisierter Strings“ auf Seite 22.

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

Tag	Attribut	Erforderlich	Beschreibung
`documenttype` `(root)`		Ja	Übergeordneter Knoten.
	`id`	Ja	Eindeutiger Bezeichner in allen Definitionsdateien für Dokumenttypen.

	`servermodel`	Nein	Gibt das entsprechende Servermodell an (Groß- und Kleinschreibung ist zu beachten); standardmäßig sind die folgenden Werte gültig:
			`ASP.NET C#`
			`ASP.NET VB`
			`ASP VBScript`
			`ASP JavaScript`
			`ColdFusion`
			`JSP`
			`PHP MySQL`
			Beim Aufruf der Funktion `getServerModelDisplayName()` werden diese Namen zurückgegeben. Die Implementierungsdateien für Servermodelle sind im Ordner „Configuration/ServerModels“ gespeichert.
			Entwickler von Erweiterungen können neue Servermodelle erstellen, indem sie diese Liste erweitern.

	`internaltype`	Ja	Eine grundsätzliche Klassifizierung zur Verarbeitung von Dateien in Dreamweaver. `internaltype` gibt an, ob für dieses Dokument die Entwurfsansicht aktiviert ist, und regelt Sonderfälle wie Dreamweaver-Vorlagen oder Dreamweaver-Erweiterungen.
			Folgende Werte sind gültig:
			`Dynamic`
			`DWExtension` (hat bestimmte Anzeigebereiche)
			`DWTemplate` (hat bestimmte Anzeigebereiche)
			`HTML`
			`HTML4`
			`Text` (nur Codeansicht)
			`XHTML1`
			`XML` (nur Codeansicht)
			Alle servermodellbezogenen Dokumenttypen verweisen auf `Dynamic`. HTML verweist auf `HTML`. Skriptdateien (z. B. CSS, JS, VB und CS) verweisen auf `Text`.
			Wenn für `internaltype` der Wert `DWTemplate` angegeben ist, legen Sie einen Wert für `dynamicid` fest. Andernfalls wird im Bedienfeld „Serververhalten“ oder „Bindungen“ die im Dialogfeld „Neues Dokument“ erstellte neue leere Vorlage nicht erkannt. Instanzen dieser Vorlage sind nichts weiter als eine HTML-Vorlage.

Tag	Attribut	Erforderlich	Beschreibung
	`dynamicid`	Nein	Ein Verweis auf den eindeutigen Bezeichner eines dynamischen Dokumenttyps. Dieses Attribut hat nur Bedeutung, wenn `internaltype` den Wert `DWTemplate` hat. Mithilfe dieses Attributs können Sie eine dynamische Vorlage mit einem dynamischen Dokumenttyp verknüpfen.
	`winfileextension`	Ja	Die mit dem Dokumenttyp unter Windows verknüpfte Dateinamenerweiterung. Zur Angabe mehrerer Dateinamenerweiterungen verwenden Sie eine durch Kommas getrennte Liste. Die erste Erweiterung in der Liste wird von Dreamweaver verwendet, wenn der Benutzer ein Dokument des Typs `documenttype` speichert. Wenn zwei nicht auf das Servermodell bezogene Dokumenttypen die gleiche Dateinamenerweiterung verwenden, verwendet Dreamweaver den ersten Eintrag als Dokumenttyp für die Erweiterung.
	`macfileextension`	Ja	Die auf dem Macintosh mit dem Dokumenttyp verknüpfte Dateinamenerweiterung. Zur Angabe mehrerer Dateinamenerweiterungen verwenden Sie eine durch Kommas getrennte Liste. Die erste Erweiterung in der Liste wird von Dreamweaver verwendet, wenn der Benutzer ein Dokument des Typs `documenttype` speichert. Wenn zwei nicht auf das Servermodell bezogene Dokumenttypen die gleiche Dateinamenerweiterung verwenden, verwendet Dreamweaver den ersten Eintrag 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.

Tag	Attribut	Erforderlich	Beschreibung
	`priorversionservermodel`	Nein	Wenn das Servermodell dieses Dokuments eine Entsprechung in Dreamweaver UltraDev 4 hat, wird hier 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 unter „Bereitstellen lokalisierter Strings“ auf Seite 22. Eine Formatierung ist nicht zulässig. Somit können keine HTML-Tags angegeben werden.
`description` (Subtag)		Nein	Der String, der den Dokumenttyp beschreibt. 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 unter „Bereitstellen lokalisierter Strings“ auf Seite 22. Eine Formatierung ist zulässig. Somit können HTML-Tags angegeben werden.

Hinweis: *Wenn der Benutzer ein neues Dokument speichert, überprüft Dreamweaver die Liste der mit dem Dokumenttyp verknüpften Erweiterungen der jeweiligen Plattform. Dies ist beispielsweise* `winfileextension` *und* `macfileextension`*. Dreamweaver wählt den ersten String in der Liste aus und verwendet ihn als Standard-Dateinamenerweiterung. Um diese Standard-Dateinamenerweiterung zu ändern, gruppieren Sie die durch Kommas getrennten Erweiterungen in der Liste um, sodass die neue Standard-Dateinamenerweiterung am Anfang der Liste aufgeführt wird.*

Beim Start liest Dreamweaver alle Definitionsdateien für Dokumenttypen und erstellt eine Liste gültiger Dokumenttypen. Dreamweaver behandelt 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 die Definitionsdateien für Dokumenttypen beschädigt oder nicht im Ordner „Configuration/DocumentTypes“ vorhanden sind, wird Dreamweaver mit einer Fehlermeldung geschlossen.

Definieren dynamischer Vorlagen

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

- Der Wert des Attributs `internaltype` für den neuen Dokumenttyp muss `DWTemplate` lauten.
- Das Attribut `dynamicid` muss angegeben werden und sein Wert muss ein Verweis auf den Bezeichner eines vorhandenen dynamischen Dokumenttyps sein.

  Im folgenden Beispiel wird ein dynamischer Dokumenttyp definiert:

```
<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. Darüber hinaus kann der Benutzer beim Erstellen von Instanzen der neuen Vorlage auch PHP-Serververhalten in der Instanz erstellen.

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

Hinzufügen und Ändern von Dokumenterweiterungen und Dateitypen

Dreamweaver zeigt im Dialogfeld „Öffnen“ („Datei“ > „Öffnen“) standardmäßig alle kompatiblen Dateitypen an. Nach dem Erstellen eines Dokumenttyps muss der Entwickler der Erweiterung die entsprechende Datei „Extensions.txt“ aktualisieren. Unter Umständen arbeitet der Benutzer auf einem Mehrbenutzer-Betriebssystem (z. B. Windows XP, Windows Vista oder Mac OS X). In diesen Fällen befindet sich im Konfigurationsordner des Benutzers eine weitere Datei „Extensions.txt“. Der Benutzer muss diese Datei „Extensions.txt“ aktualisieren, da es sich um die Instanz handelt, die Dreamweaver sucht und analysiert.

Der Speicherort des Ordners „Configuration“ für einzelne Benutzer richtet sich nach dem Betriebssystem.

Bei Windows XP befindet sich der Ordner unter:

Festplatte:\Dokumente und Einstellungen*Benutzername*\Anwendungsdaten\Adobe\Dreamweaver CS5\Configuration

Hinweis: *Dieser Ordner befindet sich unter Umständen in einem versteckten Ordner.*

Bei Windows Vista befindet sich der Ordner unter:

Festplatte:\Benutzer*Benutzername*\AppData\Roaming\Adobe\Dreamweaver CS5\Configuration

Für Mac OS X-Plattformen gilt der folgende Speicherort:

Festplatte:/Benutzer/*Benutzername*/Library/Application Support/Adobe/Dreamweaver CS5/Configuration

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

Hinweis: *Auf Mehrbenutzerplattformen verwendet Dreamweaver die Kopie der Datei „Extensions.txt“ im Konfigurationsordner des jeweiligen Benutzers, nicht die Datei im Dreamweaver-Ordner „Configuration“. Wenn Sie die Version von „Extensions.txt“ im Dreamweaver-Ordner „Configuration“ bearbeiten, haben die Änderungen daher keine Auswirkungen auf Dreamweaver.*

Zum Erstellen einer Dokumenterweiterung können Sie entweder einem vorhandenen Dokumenttyp die neue Erweiterung hinzufügen oder einen Dokumenttyp erstellen.

Hinzufügen einer neuen Erweiterung zu einem vorhandenen Dokumenttyp

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

Hinzufügen eines neuen Dokumenttyps

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

   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“ aus und klicken auf das Popupmenü für die Dateitypen.

Ändern des Standarddateityps, 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 einem Texteditor.
3. Schneiden Sie die Zeile mit der gewünschten neuen Standardeinstellung aus. Fügen Sie sie dann am Anfang der Datei ein, sodass diese Zeile an erster Stelle in der Datei steht.
4. Speichern Sie die Datei „Extensions.txt“.
5. Starten Sie Dreamweaver neu.

Um die Änderungen zu überprüfen, wählen Sie „Datei“ > „Öffnen“ aus und klicken auf das Popupmenü für die Dateitypen.

Verwandte Themen

http://www.adobe.com/go/16410_de

Bereitstellen lokalisierter Strings

In einer Definitionsdatei für Dokumenttypen 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 für die beiden Subtags lokalisierte Strings 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.

Bereitstellen lokalisierter Strings

1 Fügen Sie am Anfang der Definitionsdatei für Dokumenttypen die folgende Anweisung ein:

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

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

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

3 Verwenden Sie an der Stelle in der Definitionsdatei für Dokumenttypen, an der Sie einen lokalisierten String einfügen möchten, 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 den folgenden 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 Page with <em>special</em> features]]>"
        />
    ...
    </strings>
```

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

Regeln bezüglich Definitionsdateien für Dokumenttypen

In Dreamweaver können Dokumenttypen, die mit einem Servermodell verknüpft sind, Dateierweiterungen gemeinsam verwenden. Die Dateierweiterung „.asp“ kann beispielsweise den beiden Dokumenttypen ASP-JS und ASP-VB zugewiesen sein. (Informationen darüber, welches Servermodell Vorrang hat, finden Sie unter „canRecognizeDocument()“ auf Seite 342.)

Dokumenttypen hingegen, die nicht mit einem Servermodell verknüpft sind, können in Dreamweaver Dateierweiterungen nicht gemeinsam verwenden.

Wenn eine Dateierweiterung zwei Dokumenttypen zugewiesen ist, wobei ein Dokumenttyp mit einem Servermodell verknüpft ist und der andere nicht, erhält der letztere Vorrang. Angenommen, Sie haben einen Dokumenttyp mit dem Namen „SAM“ erstellt, 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 und nicht ASP-JS den Dokumenttyp SAM zu.

Definieren von Dokumentdeklarationen

Dreamweaver ermöglicht das Festlegen von DTDs für Dokumente mithilfe der Datei `MMDocumentTypeDeclarations.xml`, die sich im Ordner „Configuration/DocumentTypes“ befindet. Die Liste der verfügbaren DTDs und der Dokumente, für die sie gelten, ist in der Datei `MMDocumentTypeDeclarations.xml` festgelegt.

Öffnen von Dokumenten in Dreamweaver

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

Wenn Dreamweaver einen eindeutigen Dokumenttyp findet, wird dieser Typ verwendet und ggf. das verknüpfte Servermodell für das Dokument geladen, das der Benutzer öffnet. Wenn der Benutzer Serververhalten von Dreamweaver UltraDev 4 verwenden möchte, lädt Dreamweaver das entsprechende Servermodell von UltraDev 4.

Wenn die Dateierweiterung auf mehrere Dokumenttypen verweist, führt Dreamweaver folgende Aktionen aus:

- Wenn 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 342). Dreamweaver erfasst die Rückgabewerte und ermittelt, welches Servermodell die höchste positive Ganzzahl zurückgegeben hat. Der Dokumenttyp, dessen Servermodell die höchste Ganzzahl zurückgegeben hat, wird dem geöffneten Dokument zugewiesen. Wenn mehrere Servermodelle die gleiche Ganzzahl zurückgeben, ermittelt Dreamweaver in der alphabetischen Liste dieser Servermodelle den ersten Eintrag und verwendet diesen Dokumenttyp. Wenn ein .asp-Dokument z. B. sowohl ASP-JS als auch ASP-VB zugeordnet ist und die jeweiligen `canRecognizeDocument()`-Funktionen den gleichen Wert 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.

Anpassen von Arbeitsbereichlayouts

In Dreamweaver kann das Arbeitsbereichlayout angepasst werden. Sie können festlegen, welche Bedienfelder im angegebenen Layout dargestellt werden, ob Bedienfelder ein- oder ausgeblendet werden und ggf. deren Position und Größe sowie die Position und Größe des Anwendungs- und Dokumentfensters ändern.

Das Arbeitsbereichlayout wird in XML-Dateien festgelegt, die im Ordner „Configuration/Workspace layouts“ gespeichert sind. In den folgenden Abschnitten wird die Syntax der XML-Tags beschrieben. Optionale Attribute werden in den Attributlisten durch geschweifte Klammern ({}) markiert. Attribute, die nicht durch geschweifte Klammern gekennzeichnet sind, sind obligatorisch.

<panelset>

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

Attribute
Keine.

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

Container
Keine.

Beispiel

```
<panelset>
    <!-- panelset tags here -->
</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`, wenn das Anwendungsfenster beim Start maximiert sein soll, andernfalls `false`. Der Standardwert ist `true`.

Inhalt
Keine.

Container
Dieses Tag muss sich innerhalb eines `panelset`-Tags befinden.

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. Wenn `maximize` den Wert `true` aufweist, wird der Wert für `rect` ignoriert.
- `maximize` ist ein boolescher Wert: `true`, wenn das Dokumentfenster beim Start maximiert sein soll, andernfalls `false`. Der Standardwert ist `true`.

Inhalt

Keine.

Container

Dieses Tag muss sich innerhalb eines `panelset`-Tags befinden.

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. Wenn sich der Ganzzahlwert nicht auf dem Bildschirm befindet, wird die Bedienfeldgruppe so an der diesem Wert nächsten Bildschirmposition angezeigt, dass sie sichtbar ist. Bei den relativen Werten kann es sich um „left“ oder „right“ handeln. Diese Werte geben an, welcher Rand der Bedienfeldgruppe an welchem Rand des virtuellen 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. Wenn sich der Ganzzahlwert nicht auf dem Bildschirm befindet, wird die Bedienfeldgruppe so an der diesem Wert nächsten Bildschirmposition angezeigt, dass sie sichtbar ist. Bei den relativen Werten kann es sich um „top“ oder „bottom“ handeln. Diese Werte geben an, welcher Rand der Bedienfeldgruppe an welchem Rand des virtuellen Bildschirms ausgerichtet werden soll.

- `width` ist die Breite der Bedienfeldgruppe in Pixel. Dieses Attribut ist optional. Wenn Sie die Breite nicht angeben, wird der für die Bedienfeldgruppe vorgegebene Standardwert verwendet.
- `height` ist die Höhe der Bedienfeldgruppe in Pixel. Dieses Attribut ist optional. Wenn Sie die Höhe nicht angeben, wird der für die Bedienfeldgruppe vorgegebene Standardwert verwendet.
- `dock` ist ein String, der angibt, an welchen Rand des Anwendungsframes die Bedienfeldgruppe angedockt ist. Auf dem Macintosh wird dieses Attribut ignoriert, da Bedienfeldgruppen auf dieser Plattform nicht angedockt sein können.
- `collapse` ist ein boolescher Wert: `true` gibt an, dass die Bedienfeldgruppe reduziert ist, `false` bedeutet, dass die Bedienfeldgruppe erweitert ist. Dieses Attribut wird auf dem Macintosh ignoriert, da diese Plattform schwebende Bedienfelder verwendet.

Inhalt

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

Container

Dieses Tag muss sich innerhalb eines `panelset`-Tags befinden.

Beispiel

```
<panelset>
    <panelframe rect="196 453 661 987" visible="true" dock="floating">
        <!-- panelcontainer tags here -->
    </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, andernfalls `false`.
- `title` ist ein String, der den Titel des Bedienfelds angibt.
- `height` ist eine Ganzzahl, mit der die Höhe des Bedienfelds in Pixel angegeben wird. Dieses Attribut ist optional. Wenn Sie `height` nicht angeben, wird der für jedes Bedienfeld vorgegebene Standardwert verwendet.

  Hinweis: *Die Breite wird vom übergeordneten Element übernommen.*
- `activepanel` ist eine Zahl, die die ID des sich im Vordergrund befindenden Bedienfelds angibt.
- `visible` ist ein boolescher Wert: `true`, wenn das Bedienfeld sichtbar ist, andernfalls `false`.
- `maximize` ist ein boolescher Wert: `true`, wenn das Bedienfeld anfangs maximiert sein soll, andernfalls `false`.
- `maxRestorePanel` ist eine Zahl und die ID 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 unterhalb sondern in der Titelleiste platziert werden sollen, andernfalls `false`.

Inhalt

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

Container

Dieses Tag muss sich innerhalb eines `panelframe`-Tags befinden.

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 here --->
        </panelcontainer>
    </panelframe>
</panelset>
```

<panel>

Beschreibung

Gibt das im Bedienfeldcontainer enthaltene Bedienfeld an.

Attribute

`id, visibleTab`

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

Software	ID	Bedienfeld
Adobe® Flash®	1	Eigenschaften
	2	Aktionen
	3	Ausrichten
	4	Verhalten
	5	Komponenten
	6	Komponenteninspektor
	7	Farbmischer
	8	Farbfelder
	9	Verlauf
	10	Info
	11	Bibliothek
	12	Film-Explorer
	13	Ausgabe
	14	Eigenschaften

Software	ID	Bedienfeld
	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 die Registerkarte und das Bedienfeld sichtbar sind, andernfalls `false`.

Inhalt

Keine.

Container

Dieses Tag muss sich innerhalb eines `panelcontainer`-Tags befinden.

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>
```

Anpassen der Code-Symbolleiste

In der Code-Symbolleiste werden anfänglich 15 Schaltflächen angezeigt. 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.

Ändern der Reihenfolge der Schaltflächen

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

   ```
   <!-- Code view toolbar -->
   ```

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

Entfernen einer Schaltfläche

1 Öffnen Sie die Datei „toolbars.xml“ im Ordner „Configuration/Toolbars“.

2 Suchen Sie nach folgendem Kommentar, der den Abschnitt für die Code-Symbolleiste einleitet:

```
<!-- Code view toolbar -->
```

3 Schließen Sie die zu entfernende Schaltfläche in Kommentarzeichen ein.

Im folgenden Beispiel ist eine Schaltfläche dargestellt, die in einen Kommentar eingeschlossen wurde, damit sie nicht in der Symbolleiste angezeigt wird.

```
<!-- remove button from Coding toolbar
    <button id="DW_ExpandAll"
        image="Toolbars/images/MM/T_ExpandAll_Sm_N.png"
        disabledImage="Toolbars/images/MM/T_ExpandAll_Sm_D.png"
        tooltip="Expand All"
        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.

Ändern der Zuordnungen von Tastaturkurzbefehlen

Dreamweaver umfasst viele Tastaturkurzbefehle zu Programmfunktionen. Die Standard-Tastaturkurzbefehle sind in der Datei „menus.xml“ aufgelistet und gelten für die US-Tastaturbelegung. Aufgrund der großen Anzahl möglicher Tastaturkurzbefehle in Dreamweaver müssen bestimmte nicht alphanumerische Tastaturkurzbefehle (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 ausgeliefert, die die Zuordnungen von Tastaturkurzbefehlen bei internationalen Tastaturen definieren. Diese Dateien befinden sich im Ordner „Configuration/Menus/Adaptive Sets“. Wenn Dreamweaver feststellt, dass an den Computer eine internationale Tastatur angeschlossen ist, werden die Tastaturkurzbefehle auf die Zuordnungsdatei der betreffenden Tastatur gesetzt. Wenn für eine Tastatur keine Zuordnungsdatei verfügbar ist, werden alle Tastaturkurzbefehle entfernt, die auf der betreffenden Tastatur nicht ausgeführt werden können.

Die Namen der Zuordnungsdateien für Tastaturkurzbefehle bestehen meist aus einem Sprachcode von zwei Buchstaben für die entsprechende Tastaturbelegung. Die Datei für die deutsche Tastaturbelegung hat beispielsweise den Namen de.xml. Wenn es in einer Sprache verschiedene Tastaturbelegungen für verschiedene Länder gibt, bestehen die Namen der Zuordnungsdateien aus den zwei Buchstaben für den Sprachcode, einem Bindestrich („-“) und dann zwei Buchstaben für den Ländercode. So lautet beispielsweise der Dateiname für die schweizerdeutsche Tastaturbelegung de-ch.xml. 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).

Wenn sich die aktive Tastaturspracheinstellung auf Ihrem Computer ändert, überprüft Dreamweaver, ob für die betreffende Sprache und das Land eine geeignete Zuordnungsdatei für Tastaturkurzbefehle vorhanden ist. Dreamweaver sucht zunächst nach einer länderspezifischen Zuordnungsdatei und, wenn keine vorhanden ist, nach einer Datei für die betreffende Sprache. Wenn Sie an Ihrem Computer beispielsweise eine schweizerdeutsche Tastatur angeschlossen haben, wird zuerst nach der Datei „de-ch.xml“ mit der entsprechenden Tastaturbelegung gesucht. Wenn diese Datei nicht vorhanden ist, sucht Dreamweaver nach de.xml. In der folgenden Tabelle sind alle mit Dreamweaver bereitgestellten Zuordnungsdateien aufgeführt.

Dateiname	Windows-Plattform	Macintosh-Plattform
ca.xml	Katalanisch	Katalanisches Spanisch
de.xml	Deutsch (Deutschland, Österreich)	Österreichisch, Deutsch
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	Niederländisch (Belgien), Französisch (Belgien)	Belgisch
zh-cn.xml	Chinesisch (VR China), Chinesisch (Singapur)	Vereinfachtes Chinesisch

Wenn Sie eine Tastaturbelegung verwenden, die nicht mit Dreamweaver ausgeliefert wurde, können Sie eine eigene Zuordnungsdatei für Ihre Tastatur erstellen und im Ordner „Configuration/Menus/Adaptive Sets“ speichern.

Erstellen einer Tastaturzuordnungsdatei

1. Legen Sie eine Kopie einer der Tastaturzuordnungsdateien im Ordner „Configuration/Menus/Adaptive Sets“ an und ändern Sie den Dateinamen entsprechend den 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 z. B. 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 neu 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 zu ändern sind, können Sie die für die US-Tastatur festgelegten Kurzbefehle mit denen für Ihre Sprache vergleichen. (Im Anschluss wird erläutert, wie Sie die Kurzbefehle aus zwei verschiedenen Tastaturbelegungen vergleichen können.)

5 Nachdem Sie Ihre Änderungen an den Tastaturkurzbefehlen vorgenommen haben, speichern Sie die Datei und verschieben Sie sie wieder in den Ordner „Configuration/Menus/Adaptive Sets“.

Ermitteln der zu bearbeitenden Tastaturbefehl-Tags

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

2 Starten Sie den Tastaturkurzbefehl-Editor von Dreamweaver, indem Sie „Bearbeiten“ > „Tastaturbefehle“ auswählen.

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“ wird angezeigt. Geben Sie einen Namen und Pfad für die Zusammenfassungsdatei mit den Kurzbefehlen für Ihre aktuelle Tastaturbelegung an.

4 Schließen Sie das Dialogfeld „Tastaturkurzbefehle“, nachdem Sie die Zusammenfassungsdatei gespeichert haben.

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

6 Starten Sie den Tastaturkurzbefehl-Editor von Dreamweaver, indem Sie „Bearbeiten“ > „Tastaturbefehle“ auswählen.

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

8 Schließen Sie das Dialogfeld „Tastaturkurzbefehle“, nachdem Sie die Zusammenfassungsdatei gespeichert haben.

9 Sie können nun die beiden Zusammenfassungsdateien für Tastaturkurzbefehle ausdrucken und vergleichen, welche Tastaturkurzbefehle für die Tastaturbelegung Ihrer Sprache entfernt wurden. Dies sind die Tastaturkurzbefehle, denen Sie neue Tastenkombinationen zuweisen müssen, die nur in Tastaturbelegung Ihrer Sprache verfügbar sind.

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

XML-Dateien für die Tastaturzuordnung

Im folgenden Beispiel ist das Format der französischen Zuordnungsdatei für die Tastaturbelegung („fr.xml“) angegeben:

```
<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>
```

Erläuterung:

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

Kapitel 3: Anpassen der Codeansicht

Adobe Dreamweaver enthält in der Codeansicht zwei Hilfsmittel, mit denen die Eingabe von Code vereinfacht sowie Code gut dokumentiert und klar formatiert dargestellt wird. Dabei handelt es sich um Codehinweise und Codefarben. Zusätzlich überprüft Dreamweaver den eingegebenen Code für die angegebenen Zielbrowser und ermöglicht Änderungen der HTML-Standardformatierung.

Sie können die Codehinweise und die Codefarben durch Ändern der XML-Dateien anpassen, die diese Funktionen implementieren. Sie können dem Menü für Codehinweise Einträge hinzufügen, indem Sie sie in die Datei „CodeHints.xml“ oder „SpryCodeHints.xml“ einfügen. Ändern Sie die Codefarbstildatei „Colors.xml“, um Farbschemas anzupassen, oder ändern Sie eine der Syntaxdateien für Codefarben (z. B. „CodeColoring.xml“) um Codefarbschemas anzupassen oder neu hinzuzufügen. Sie können auch die CSS-Profildatei (Cascading Style Sheet) für den Zielbrowser bearbeiten, um so die Überprüfung von CSS-Eigenschaften und CSS-Werten in Dreamweaver zu beeinflussen. Darüber hinaus können Sie die HTML-Standardformatierung von Dreamweaver im Dialogfeld „Voreinstellungen“ ändern. In den folgenden Abschnitten wird die Anpassung dieser Funktionen beschrieben.

Codehinweise

Codehinweise sind Menüs, die in Dreamweaver geöffnet werden, wenn Sie in der Codeansicht bestimmte Zeichenmuster eingeben. Codehinweise ermöglichen eine schnellere Eingabe, indem eine Liste der Strings angezeigt wird, mit denen der eingegebene String vervollständigt werden kann. Wenn der eingegebene String im Menü angezeigt wird, können Sie ihn in der Liste auswählen und die Eingabe durch Drücken der Eingabetaste bzw. des Zeilenschalters automatisch vervollständigen. Wenn Sie beispielsweise < eingeben, wird im Popupmenü eine Liste mit Tag-Namen angezeigt. Anstatt den Rest des Tag-Namens einzugeben, können Sie das Tag im Menü auswählen, um es in den Text einzufügen. Dreamweaver enthält auch Codehinweise für das Spry-Framework.

In Dreamweaver werden Menüs für Codehinweise aus der Datei „CodeHints.xml“ und allen anderen XML-Dateien im Ordner „Configuration/CodeHints“ geladen. Sie können Menüs für Codehinweise in Dreamweaver einfügen, indem Sie sie im XML-Schemaformat (siehe Beschreibung in diesem Abschnitt) in Ihren eigenen XML-Dateien definieren und die Dateien dann im Ordner „Configuration/CodeHints“ ablegen.

Nachdem der Inhalt einer Codehinweisdatei in Dreamweaver geladen wurde, können Sie über JavaScript auch dynamisch neue Menüs für Codehinweise hinzufügen. Die Liste der Sitzungsvariablen im Bedienfeld „Bindungen“ wird beispielsweise durch JavaScript-Code gefüllt. Mit demselben Code können Sie auch ein Menü für Codehinweise hinzufügen. Wenn ein Benutzer in der Codeansicht **Session.** eingibt, wird ein Menü mit Sitzungsvariablen angezeigt. Informationen zum Hinzufügen oder Ändern eines Menüs für Codehinweise mit JavaScript finden Sie im *Dreamweaver API-Referenzhandbuch* unter „Codefunktionen“.

Einige Typen von Menüs für Codehinweise können in Dreamweaver nicht über die XML-Datei oder die JavaScript-API angegeben werden. Die Dateien „CodeHints.xml“ und „SpryCodeHints.xml“ sowie die JavaScript-API stellen eine beträchtliche Teilmenge des Moduls für Codehinweise bereit. Einige Dreamweaver-Funktionen sind jedoch nicht verfügbar. Beispielsweise gibt es keine JavaScript-Funktion zum Öffnen der Farbauswahl. Somit kann das Menü „Attributwerte“ nicht über JavaScript dargestellt werden. Sie können lediglich ein Menü mit Textelementen öffnen, aus dem Sie Text einfügen können.

Hinweis: *Wenn Sie Text einfügen, wird die Einfügemarke hinter dem eingefügten String positioniert.*

Datei „CodeHints.xml“

Die Datei „CodeHints.xml“ enthält die folgenden Elemente:

- Liste aller Menügruppen

  Dreamweaver zeigt die Liste der Menügruppen an, wenn Sie im Dialogfeld „Voreinstellungen“ die Kategorie für Codehinweise auswählen. Das Dialogfeld „Voreinstellungen“ können Sie durch Auswählen von „Bearbeiten“ > „Voreinstellungen“ öffnen. Dreamweaver enthält die folgenden Menügruppen oder Typen von Menüs für Codehinweise: „Tag-Namen“, „Attributnamen“, „Attributwerte“, „Funktionsargumente“, „Objektmethoden und Variablen“ und „HTML-Entities“.

- Beschreibungen der einzelnen Menügruppen

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

- Menüs für Codehinweise

  Ein Menü besteht aus einem Muster, das das Menü für Codehinweise aufruft, und aus einer Liste mit Befehlen. Beispielsweise kann ein Muster wie `&` ein Menü wie `&`, `>`, `<` aufrufen.

  Im folgenden Beispiel ist das Format der Datei „CodeHints.xml“ angegeben. (Die fett hervorgehobenen Tags werden in „Tags für Codehinweise“ auf Seite 39 beschrieben.)

```
<codehints>
<menugroup name="HTML Entities" enabled="true" id="CodeHints_HTML_Entities">
    <description>
    <![CDATA[ When you type a '&', a pop-up menu shows
        a list of HTML entities. The list of HTML entities
        is stored in Configuration/CodeHints.xml. ]]>
    </description>
    <menu pattern="&amp;amp;">
            <menuitem value="&amp;amp;amp;" texticon="&amp;amp;"/>
            <menuitem value="&amp;amp;lt;" icon="lessThan.gif"/>
    </menu>
</menugroup>

<menugroup name="Tag Names" enabled="true" id="CodeHints_Tag_Names">
    <description>
            <![CDATA[ When you type '<', a pop-up menu shows
            all possible tag names.0You 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 Codehinweise für das Spry-Framework. Die Datei für Spry-Codehinweise („SpryCodeHints.xml“) hat dasselbe Grundformat wie die Datei „CodeHints.xml“. Sie verwendet bestimmte neue Schlüsselwörter wie `method` und enthält das neue Attribut `classpattern`, mit dem die Klassenelementliste mit der Klasse verknüpft werden kann (z. B. Spry.Data.XMLDataSet). Die Klassenelementliste für die Klassen ist innerhalb des Menüs verschachtelt (Methoden, Eigenschaften und Ereignisse).

Das Tag `<method>` und die zugehörigen Attribute ähneln dem Tag `function` und den zugehörigen Attributen, dem übergeordneten Tag `menu` muss jedoch das `classpattern`-Attribut zugewiesen sein, damit die Verknüpfung funktioniert. Außerdem gibt es ein `property`-Tag für Eigenschaften und ein `event`-Tag für Ereignisse. Diese Tags werden im Popupmenü für Codehinweise durch die entsprechenden Symbole dargestellt. Es gibt zudem die Tags `parammenu` und `parammenuitem` für die Unterstützung von Parameterhinweisen.

Im folgenden Beispiel ist das Format einer „SpryCodeHints.xml“-Datei dargestellt. (Die fett hervorgehobenen Tags werden in „Tags für Codehinweise“ auf Seite 39 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>
```

Deklarieren von Klassen

Mit dem folgenden Format wird eine Klasse deklariert, 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 Farbstatus-Engine erkennt, dass es sich um die Instanz einer Klasse handelt, den auf der linken Seite der Deklaration definierten Variablennamen akzeptiert und ihn mit den entsprechenden Klassentypen für diese Seite (z. B. mit der Variable `fooAccordion` oder dem Klassentyp `Spry.Widget.Accordion` aus dem vorherigen Beispiel) in der Liste der Variablen speichert.

Syntax für die erneute Deklaration des Klassennamens in der Datei „CodeColoring.xml“:

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

Erläuterung:

- `classlibrary` ist ein neues Tag zum Gruppieren der Klassen nach Farb-ID `"CodeColor_JavascriptSpry"`.

  `class` dient zur Auflistung aller in der Klassenbibliothek verfügbaren Klassen. Die Liste der Klassen kann um andere Spry-Klassen aus verschiedenen Spry-Paketen (z. B. Debug, Data, Utils, Widgets und Effects) oder anderen asynchronen JavaScript- und XML-Toolkits (Ajax) oder JavaScript-Bibliotheken erweitert werden.

Codehinweise für Tag-übergreifende Attribute

Dreamweaver enthält Codehinweise für die Attributnamen und Attributwerte von Spry. Diese Attribute werden in den verschiedenen Tags immer wieder verwendet. Damit nicht jede Datei des Typs „*tag*.vtm“ geöffnet und die Spry-Attributliste hinzugefügt werden muss, verfügt Dreamweaver über ein neues XML-Format für Attributgruppen (z. B. „spry:region“, „spry:repeat“) und Tag-Gruppen, das in einer einzelnen VTM-Datei mit dem Namen „Spry.vtm“ im Verzeichnis „Configuration/TagLibraries“ angewendet werden kann.

Spry-Attributgruppenformat

Der folgende Code gibt das Format der VTM-Datei an. In diesem Format können Sie die Attribute angeben, die für bestimmte Tags gelten.

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

```
<crosstag_attributes>
    <attributegroup id="group_id_1" name="group_name_1">
        <attrib name = "fooAttr1">
        <attrib name = "barAttr1">
        ...
        <taggroup>
            <tag name = "fooTag1">
            <tag name = "barTag1">
            ...
        </taggroup>
    </attribgroup>
    <attributegroup 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>
```

Erläuterung:

- `attributegroup` listet die Attribute der nachfolgenden Tag-Gruppe auf.
- `taggroup` listet die Tags auf, für die die vorangegangenen Attribute gelten.

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-Dateien für Codehinweise enthalten die folgenden Tags, mit denen Menüs für Codehinweise definiert werden. Mit diesen Tags können Sie weitere Menüs für Codehinweise definieren.

<codehints>

Beschreibung

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

Attribute

Keine.

Inhalt

Mindestens ein `menugroup`-Tag.

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 für 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,version`

- Das Attribut·`name` ist der lokalisierte Name, der im Dialogfeld „Voreinstellungen“ in der Liste der Menügruppen in der Kategorie für Codehinweise angezeigt wird.
- `enabled` gibt an, ob die Menügruppe derzeit ausgewählt oder aktiviert ist. Eine aktivierte Menügruppe wird im Dialogfeld „Voreinstellungen“ in der Kategorie für Codehinweise durch ein Häkchen gekennzeichnet. Weisen Sie den Wert `true` zu, um die Menügruppe zu aktivieren, oder den Wert `false`, um die Menügruppe zu deaktivieren.
- `id` ist ein nicht lokalisierter Bezeichner, der auf die Menügruppe verweist.

Inhalt

Die Tags `description`, `menu` und `function`.

Container

Das `codehints`-Tag.

Beispiel

```
<menugroup name="Session Variables" enabled="true" id="Session_Code_Hints" version="1.4.2">
```

<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 von Dreamweaver aufgerufen wird, wenn der Benutzer auf den Hyperlink klickt. Sonderzeichen oder unzulässige Zeichen im String müssen Sie in das Konstrukt „XML CDATA“ einschließen, damit sie in Dreamweaver als Text erkannt werden.

Attribute

Keine.

Inhalt

Beschreibungstext.

Container

Das `menugroup`-Tag.

Beispiel

```
<description>
<![CDATA[ To add or remove tags and attributes, use the
    <a href="javascript:dw.tagLibrary.showTagLibraryEditor()">Tag Library Editor</a>.]]>
</description>
```

<menu>

Beschreibung

Dieses Tag gibt ein einzelnes Popupmenü an. Dreamweaver öffnet das Menü, wenn der Benutzer das letzte Zeichen des Strings des Musterattributs eingegeben hat. Beispielsweise kann ein Menü, das den Inhalt einer Sitzungsvariablen anzeigt, das Musterattribut "`Session.`" aufweisen.

Attribute

`pattern`, `doctypes`, `casesensitive`, `classpattern`, `displayrestriction, alias`

- Das Attribut `pattern` gibt das Muster der einzugebenden Zeichen an, die das Öffnen des Menüs für Codehinweise in Dreamweaver bewirken. Wenn das erste Zeichen des Musters ein Buchstabe, eine Ziffer oder ein Unterstrich ist, wird das Menü nur angezeigt, wenn das dem Muster vorausgehende Zeichen weder ein Buchstabe noch eine Ziffer oder ein Unterstrich ist. Wenn beispielsweise als Muster "`Session.`" angegeben ist und der Benutzer "`my_Session.`" eingibt, wird das Menü in Dreamweaver nicht angezeigt.
- `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), Adobe 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“.
- `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 bei bestimmten Dokumenttypen zwischen Groß- und Kleinschreibung des Musters unterschieden werden soll, bei anderen jedoch nicht. Wenn Sie dieses Attribut nicht angeben, wird der Standardwert `false` verwendet. Wenn `casesensitive` den Wert `true` hat, wird das Menü für Codehinweise nur geöffnet, wenn der vom Benutzer eingegebene Text genau dem mit dem pattern-Attribut angegebenen Muster entspricht. Wenn `casesensitive` den Wert `false` hat, wird das Menü auch angezeigt, wenn sich die Groß- und Kleinschreibung von Muster und Text unterscheidet.
- Das Attribut `classpattern` verknüpft die Klassenelementliste mit der Klasse.
- Mit dem Attribut `displayrestriction` wird das Menü für Codehinweise anhand der in der Datei „CodeColoring.xml“ definierten Codefarbschemas auf einen bestimmten Syntaxblock einer Programmiersprache beschränkt. Wenn beispielsweise `displayrestriction = "JavaScript"` angegeben ist, ist das Menü für Codehinweise auf JavaScript-Syntaxblocks beschränkt.
- Das Argument `alias` dient dazu, Codehinweise mit einem anderen Muster aufzurufen als dem in dem Argument „pattern“ bzw. „classpattern“ aufgeführten. Dieses Argument ist optional.

Inhalt

Das `menuitem`-Tag.

Container
Das `menugroup`-Tag.

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

<menuitem>

Beschreibung
Dieses Tag gibt den Text für ein Menüelement in einem Popupmenü für Codehinweise an. Das Tag `menuitem` gibt auch den Wert an, den Sie in den Text einfügen müssen, wenn Sie das Element auswählen.

Attribute
`label`, `value`, {`icon`}, {`texticon`}, `object`, `source`

- `label` ist der String, der in Dreamweaver im Popupmenü angezeigt wird.
- `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 drückt, ersetzt Dreamweaver den gesamten Text, den der Benutzer seit dem Öffnen des Menüs eingegeben hat. Da der Benutzer die mit dem Muster übereinstimmenden Zeichen bereits vor dem Öffnen des Menüs eingegeben hat, fügt Dreamweaver diese nicht erneut ein. Wenn Sie beispielsweise `&` (die HTML-Entität für das Et-Zeichen (&)) eingeben möchten, können Sie die folgenden `menu`- und `menuitem`-Tags definieren:

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

  Das value-Attribut enthält das Et-Zeichen (&) nicht, da es der Benutzer bereits vor dem Öffnen des Menüs eingegeben hat.
- Das optionale Attribut `icon` gibt den Pfad zu einer Bilddatei an, die Dreamweaver links neben dem Menütext als Symbol anzeigt. Der Pfad wird als URL relativ zum Ordner „Configuration“ angegeben.
- Das optionale Attribut `texticon` gibt einen Textstring an, der im Symbolbereich anstelle einer Bilddatei angezeigt wird. Dieses Attribut wird für das Menü „HTML-Entities“ verwendet.
- Das Attribut `object` verweist auf den Typ des Menüelements. Beispiel: Datentyp „Built-In“: String oder benutzerdefinierte JavaScript-Datei mit benutzerdefiniertem Datentyp.
- Das Attribut `source` verweist auf den Speicherort, in dem es definiert ist oder aus dem es stammt. Beispiel: „DOM/Javascript/custom file.js“.

Inhalt
Keine.

Container
Das `menu`-Tag.

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

<function>

Beschreibung

Wird in der Datei „CodeHints.xml“ verwendet. Dieses Tag ersetzt das `menu`-Tag zum Festlegen von Funktionsargumenten und Objektmethoden für Popupmenüs für Codehinweise. Wenn Sie in der Codeansicht einen Funktions- oder Methodennamen eingeben, zeigt Dreamweaver ein Menü der Funktionsprototypen an, wobei das aktuelle Argument fett hervorgehoben ist. Immer wenn Sie ein Komma eingeben, aktualisiert Dreamweaver das Menü und formatiert das jeweils nächste Argument fett. Wenn Sie beispielsweise in einem ColdFusion-Dokument den Funktionsnamen `ArrayAppend` eingegeben haben, wird `ArrayAppend(array, value)` im Menü für Codehinweise angezeigt. Nachdem Sie das Komma nach `array` eingegeben haben, wird im aktualisierten Menü `ArrayAppend(array, value)` angezeigt.

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

Die erkannten Funktionen werden in den XML-Dateien im Ordner „Configuration/CodeHints“ gespeichert.

Attribute

`pattern, doctypes, casesensitive`

- Das Attribut `pattern` gibt den Namen der Funktion und die zugehörige Argumentliste an. Bei Methoden beschreibt das Attribut „pattern“ den Namen des Objekts sowie den Namen und die Argumente der Methode. Für einen Funktionsnamen wird das Menü für Codehinweise angezeigt, wenn der Benutzer `functionname()` eingibt. Im Menü wird die Liste der Argumente für die Funktion angezeigt. Für eine Objektmethode wird das Menü für Codehinweise angezeigt, wenn der Benutzer Folgendes eingibt: `objectname.` (einschließlich des Punkts) eingibt. In diesem Menü werden die für das Objekt festgelegten Methoden angezeigt. Anschließend wird wie bei einer Funktion das Menü für Codehinweise mit einer Liste der Argumente für die Methode geöffnet.
- Das `doctypes`-Attribut 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 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 `doctypes` angeben. Über die Liste der Dokumenttypen können Sie festlegen, dass bei bestimmten Dokumenttypen zwischen Groß- und Kleinschreibung des Musters unterschieden werden soll, bei anderen jedoch nicht. Wenn Sie dieses Attribut nicht angeben, wird der Standardwert `false` verwendet. Wenn das `casesensitive`-Attribut den Wert `true` hat, wird das Menü für Codehinweise nur angezeigt, wenn der vom Benutzer eingegebene Text genau dem mit dem pattern-Attribut angegebenen Muster entspricht. Wenn `casesensitive` den Wert `false` hat, wird das Menü auch angezeigt, wenn sich die Groß- und Kleinschreibung von Muster und Text unterscheidet.

Inhalt

Keine.

Container

Das `menugroup`-Tag.

Beispiel

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

<method>

Beschreibung

Wird für das Spry-Framework verwendet. Dieses Tag ersetzt das `menu`-Tag zum Festlegen von Methoden für Popupmenüs für Codehinweise. Wenn Sie in der Codeansicht einen Methodennamen eingeben, öffnet Dreamweaver ein Menü der Methodenprototypen mit einer Liste der Parameter für die Methode. Anschließend wird die Abfolge der Parameter protokolliert, während sie eingefügt werden. Bei Methoden ohne Parameter schließt Dreamweaver den Methodenaufruf durch Hinzufügen von Klammern „()“.

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

Attribute

`pattern`, `icon`, `object`, `source`, `constructor`, `static`, `retType`

- Das Attribut `pattern` gibt den Namen der Methode und der zugehörigen Argumentliste an. Es beschreibt zudem den Namen des Objekts und der Methode sowie die Argumente der Methode. Das Menü zeigt die Liste der Argumente für die Methode an. Das Menü für Codehinweise wird angezeigt, wenn der Benutzer `objectname.` (einschließlich des Punkts) eingibt. In diesem Menü werden die für das Objekt festgelegten Methoden angezeigt. Anschließend wird wie bei einer Funktion das Menü für Codehinweise mit einer Liste der Argumente für die Methode geöffnet.
- Mit dem Attribut `icon` wird das Symbol angegeben, das verwendet werden soll.
- Das Attribut `object` gibt den Typ des Befehls an. Beispiel: Datentyp „Built-In“: String oder benutzerdefinierte JavaScript-Datei mit benutzerdefiniertem Datentyp.
- Das Attribut `source` verweist auf den Speicherort, in dem es definiert ist oder aus dem es stammt. Beispiel: „DOM/Javascript/custom file.js“.
- Das Attribut `constructor` ist ein boolescher Wert. `constructor` = `true` verweist auf die Methode, die die Objektinstanz erstellt und die im Vergleich zu anderen Objektmethoden gesondert gekennzeichnet ist.
- Das Attribut `static` ist ein boolescher Wert. `static` = `true` gibt an, dass die Methode nicht für eine bestimmte Objektinstanz, sondern für den Objekttyp verwendet wird. Beispiel:

  ```
  Date.parse(dateString)
  ```

- Das Attribut `retType` verweist auf den Rückgabetyp der Methode, der wiederum ein Objekttyp zur Unterstützung einer Hierarchie von Codehinweisen sein kann.

Inhalt

Keine.

Container

Das `menu`-Tag.

<parammenu>

Beschreibung
Wird für ein beliebiges Objekt (JavaScript) verwendet, um Parameterhinweise für die Parameter anzugeben, die die Methode oder Funktion akzeptiert.

Attribute
`pattern, name, index, type`

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

Inhalt
Keine.

Container
Das Tag `method` oder `function`.

<parammenuitem>

Beschreibung
Kann für ein beliebiges Objekt (JavaScript) verwendet werden, um Parameterhinweise für die Parameter anzugeben, die die Methode oder Funktion akzeptiert.

Attribute
`label`, `value`, `icon`, `{datatype}`, `object`, `source`

- Das Attribut `label` gibt den in Dreamweaver anzuzeigenden Namen an. Dieses Argument ist erforderlich.
- Das Attribut `value` gibt den Wert an, den Dreamweaver einfügen soll, wenn das Element im Menü für Codehinweise ausgewählt wird. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das in Dreamweaver im Menü für Codehinweise verwendet werden muss. Dieses Argument ist erforderlich.
- Mit dem Attribut `datatype` können Sie den Typ `string` festlegen, der angibt, dass schließende Anführungszeichen hinzugefügt werden müssen, wenn der Benutzer einen Wert im Menü für Codehinweise auswählt. Dieses Argument ist optional.

- Das Attribut `object` gibt den Typ des Befehls an. Beispiel: Datentyp „Built-In“: String oder benutzerdefinierte JavaScript-Datei mit benutzerdefiniertem Datentyp.
- Das Attribut `source` verweist auf den Speicherort, in dem es definiert ist oder aus dem es stammt. Beispiel: „DOM/Javascript/custom file.js“.

Inhalt
Keine.

Container
Das `parammenu`-Tag.

<optionparammenu>

Beschreibung
Wird für ein beliebiges Objekt (JavaScript) verwendet, um Optionsarray-Hinweise für die Argumente anzugeben, die die Methode oder Funktion akzeptiert. Ein Optionsarray ist ein Argument, das Unterargumente im Format `option:value` haben kann. Die meisten Spry-Objekte verwenden ein Optionsarray-Argument, sodass Benutzer das Verhalten eines Objekts (z. B. eines Datensatzes, Widgets oder Effekts) konfigurieren können. Optionsarrays werden in der Regel im Format `{option1: value1, option2: value2, option3: value3, ...}` angegeben.

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

- Das Attribut `pattern` gibt die Zeichen an, mit denen das Menü für Codehinweise aufgerufen wird. Dieses Argument ist erforderlich.
- Das Attribut `label` gibt den Namen des Parameters an. Dieses Argument ist erforderlich.
- Das Attribut `value` gibt den Wert des Parameters an, der eingefügt werden soll, wenn der Benutzer das Element im Codehinweismenü auswählt. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol angegeben, das verwendet werden soll. Dieses Argument ist erforderlich.
- Das Argument `type` gibt den Datentyp an. Folgende Datentypen werden unterstützt:
  - `enumerated` (Standard) gibt eine Liste der anzuzeigenden verschachtelten `optionparammenuitem`-Elemente an.
  - `spryDataReferences` gibt eine Liste von Spry-Datensatzspalten an.
  - `cssStyle` gibt eine Liste der für die Seite verfügbaren CSS-Klassen an.
  - `cssId` gibt eine Liste der für die Seite verfügbaren CSS-Selektor-ID-Regeln an.

Inhalt
Keine.

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

<optionparammenuitem>

Beschreibung
Wird für ein beliebiges Objekt (JavaScript) verwendet, um Parameterhinweise für die Parameter anzugeben, die die Methode oder Funktion akzeptiert.

Attribute
`label, value, icon, {datatype}`

- Das Attribut `label` gibt den anzuzeigenden Namen an. Dieses Argument ist erforderlich.
- Das Attribut `value` gibt den Wert an, der eingefügt werden soll, wenn das Element im Menü für Codehinweise ausgewählt wird. Dieses Argument ist erforderlich.
- Mit dem Attribut `icon` wird das Symbol festgelegt, das im Menü für Codehinweise verwendet werden soll. Dieses Argument ist erforderlich.
- Mit dem Attribut `datatype` können Sie den Typ `string` festlegen, der angibt, dass schließende Anführungszeichen hinzugefügt werden müssen, wenn der Benutzer einen Wert im Menü für Codehinweise auswählt. Dieses Argument ist optional.

Inhalt
Keine.

Container
Das `<optionparammenu>`-Tag.

<property>

Beschreibung
Dieses Tag beschreibt Eigenschaften oder Felder eines Objekts und hat die folgenden Standardattribute.

Attribute
`label`, `value`, `icon`, `object`, `source`, `static`, `propType`, `item`

- `label` ist der String, der in Dreamweaver im Popupmenü angezeigt wird.
- `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 drückt, ersetzt Dreamweaver den gesamten Text, den der Benutzer seit dem Öffnen des Menüs eingegeben hat. Da der Benutzer die mit dem Muster übereinstimmenden Zeichen bereits vor dem Öffnen des Menüs eingegeben hat, fügt Dreamweaver diese nicht erneut ein.
- Das optionale Attribut `icon` gibt den Pfad zu einer Bilddatei an, die Dreamweaver links neben dem Menütext als Symbol anzeigt. Der Pfad wird als URL relativ zum Ordner „Configuration“ angegeben.
- Das Attribut `object` verweist auf den Typ des Menüelements. Beispiel: Datentyp „Built-In“: String oder benutzerdefinierte JavaScript-Datei mit benutzerdefiniertem Datentyp.
- Das Attribut `source` verweist auf den Speicherort, in dem es definiert ist oder aus dem es stammt. Beispiel: „DOM/Javascript/custom file.js“.
- Das Attribut `static` ist ein boolescher Wert. `static = true` gibt an, dass die Methode nicht für eine bestimmte Objektinstanz, sondern für den Objekttyp verwendet wird. Beispiel:

```
Number.MAX_VALUE
```

- Das Attribut `propType` verweist auf den Typ der Eigenschaft, der wiederum ein Objekttyp zur Unterstützung einer Hierarchie von Eigenschaftenhinweisen sein kann. Beispiel:

  ```
  domElement.innerHTML.<code hints for String type>
  ```

- Das Attribut `item` verweist auf den Elementtyp, wenn es sich bei dem Attribut `propType` um den collection-Containertyp handelt. Mit `item` wird der Typ der einzelnen Elemente im Container festgelegt (vorausgesetzt, es handelt sich um eine homogene Gruppe mit Elementen des gleichen Typs).

Inhalt
Keine.

Container
Das `menu`-Tag.

<event>

Beschreibung
Dieses Tag beschreibt Ereignisse eines Objekts und hat die folgenden Standardattribute.

Attribute
`label`, `icon`, `source`, `object`

- Das Attribut `label` ist der Name des Ereignisses.
- Das Attribut `icon` gibt den Pfad zu einer Bilddatei an, die Dreamweaver neben dem Menütext als Symbol anzeigt.
- Das Attribut `source` verweist auf den Speicherort, in dem es definiert ist oder aus dem es stammt.
- Das Attribut `object` gibt den Typ des Befehls an.

Container
Das `menu`-Tag.

Beispiel

```
<event label="onblur" source="DOM 1&amp;amp;2" icon="shared/mm/images/codeHintEvent.gif"/>
```

Codefarben

In Dreamweaver können Sie die in der Codeansicht angezeigten Codefarbschemas anpassen oder erweitern. So können Sie einem Schema neue Schlüsselwörter hinzufügen oder Codefarbschemas für neue Dokumenttypen einfügen. Wenn Sie JavaScript-Funktionen zur Verwendung in Ihrem clientseitigen Skript entwickeln, können Sie die Namen dieser Funktionen im Bereich für die Schlüsselwörter hinzufügen, damit sie in der im Dialogfeld „Voreinstellungen“ angegebenen Farbe angezeigt werden. Außerdem haben Sie die Möglichkeit, ein Codefarbschema 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 möglicherweise manuell bearbeitet wurden, neu laden können. Weitere Informationen zu dieser Funktion finden Sie im *Dreamweaver API-Referenzhandbuch*.

Um ein Codefarbschema zu aktualisieren oder ein neues Schema hinzuzufügen, bearbeiten Sie die Definitionsdateien für Codefarben.

Dateien für Codefarben

Dreamweaver definiert Codefarbstile und -schemas in XML-Dateien, die sich im Ordner „Configuration/CodeColoring“ befinden. Eine Datei für Codefarbstile legt die Stile für Felder fest, die in Syntaxdefinitionen definiert sind. Der Stammknoten ist `<codeColors>`. Eine Datei für Codefarbschemas definiert die Codefarbsyntax und hat den Stammknoten `<codeColoring>`.

Die in Dreamweaver bereitgestellte Datei für Codefarbstile ist die Datei „Colors.xml“. Die Dateien für die Codefarbsyntax 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 Tag-Hierarchie in einer Datei für Codefarbstile:

```
<codeColors>
    <colorGroup>
        <syntaxColor id="CodeColor_HTMLEntity" bold="true" />
        <syntaxColor id="CodeColor_JavascriptNative" text="#009999" />
        <syntaxColor id="CodeColor_JavascriptNumber" text="#FF0000" />
...
        <tagColor id="CodeColor_HTMLStyle" text="#990099" />
        <tagColor id="CodeColor_HTMLTable" text="#009999" />
        <syntaxColor id="CodeColor_SpryAttributes" text="#FF6208" />
...
    </colorGroup>
</codeColors>
```

Farben werden als RGB-Hexadezimalwerte (Rot-Grün-Blau) angegeben. Die Anweisung `text="009999"` im oben angegebenen XML-Code weist der ID `"CodeColor_JavascriptNative"` beispielsweise eine blaugrüne Farbe zu.

Der folgende Auszug aus der Datei „CodeColoring.xml“ veranschaulicht die Tag-Hierarchie in einer Datei für Codefarbschemas sowie die Beziehung zwischen der Datei für Codefarbstile und der Datei für Codefarbschemas:

```
<codeColoring>
    <scheme name="Text" id="Text" doctypes="Text" priority="1">
        <ignoreTags>Yes</ignoreTags>
        <defaultText name="Text" id="CodeColor_TextText" />
        <sampleText doctypes="Text">
<![CDATA[Default file syntax highlighting.
The quick brown fox
jumped over the lazy dog.
]]>
        </sampleText>
    </scheme>
    <scheme name="HTML" id="HTML" doctypes=
"ASP.NET_VB,ASP.NET_CSharp,ASP-JS,ASP-VB,ColdFusion,CFC,HTML,JSP,PHP_MySQL,LibraryItem,
    WML,XSLT" priority="50">
        <ignoreCase>Yes</ignoreCase>
        <ignoreTags>No</ignoreTags>
        <defaultText name="Text" id="CodeColor_HTMLText" />
        <defaultTag name="Other Tags" id="CodeColor_HTMLTag" />
        <defaultAttribute />
        <commentStart name="Comment" id="CodeColor_HTMLComment"><![CDATA[<!--]]>
            </commentStart>
    ...
    <tagGroup name="HTML Anchor Tags" id="CodeColor_HTMLAnchor" taglibrary="DWTagLibrary_html"
         tags="a" />
    <tagGroup name="HTML Form Tags" id="CodeColor_HTMLForm" taglibrary="DWTagLibrary_html" tags
="select,form,input,option,textarea" />
```

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

Dreamweaver enthält die folgenden Codefarbschemas: Standard, HTML, JavaScript, ASP_JavaScript, ASP_VBScript, JSP und ColdFusion. Das Standardschema hat den `id`-Wert `"Text"`. Das Standardschema wird für Dokumenttypen verwendet, für die kein Codefarbschema definiert wurde.

Eine Codefarbdatei enthält die im Folgenden 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, ignoreMMTParams, ignoreTags, isLocked, keyword, keywords, numbers/, operators, regexp, sampleText, searchPattern, stringStart, stringEnd, stringEsc, tagGroup`

<scheme>

Beschreibung

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

Ab Dreamweaver CS4 werden jetzt <scheme>-Tags mit derselben ID mithilfe des Codefarben-Parsers verbunden. Alle Tags, bei denen keine Konflikte aufgetreten sind, werden dem vorhandenen <scheme>-Tag hinzugefügt. Das Schema mit dem aktuellsten Dateidatum hat Vorrang vor allen anderen Tags.

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 Schemaname und Feldname an, z. B. `HTML Comment`. Wenn Sie keinen Namen eingeben, werden die Felder des Schemas nicht im Dialogfeld „Farbschema bearbeiten“ aufgeführt. Weitere Informationen zum Dialogfeld „Farbschema bearbeiten“ finden Sie unter „Bearbeiten von Schemas“ auf Seite 69.
- *`id="id_string"`* Erforderlich. Ein Bezeichnerstring, der diesem Syntaxelement Farbe und Stil zuordnet.
- *`priority="string"`* Die Werte liegen zwischen „1“ und „99“. Die höchste Priorität ist „1“. Gibt die Priorität 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 erforderlich, 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, ignoreMMTParam, 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 angegeben werden 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 zu `blockEnd`-Strings finden Sie unter „Platzhalterzeichen“ auf Seite 66.

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 angegeben werden 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 zu `blockStart`-Strings finden Sie unter „Platzhalterzeichen“ auf Seite 66. Weitere Informationen zum Attribut `blockStart.scheme` finden Sie unter „Farben von Schemablocktrennzeichen“ auf Seite 63.

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. Dokumenttypen sind in der Dreamweaver-Datei „Configuration/DocumentTypes/MMDocumentTypes.xml“ definiert.
- `id="`*`id_string`*`"` Erforderlich, wenn `scheme="customText"`. Ein Bezeichnerstring, 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. Definiert die Farbe der Strings `blockStart` und `blockEnd`. Weitere Informationen zu möglichen Werten für das scheme-Attribut finden Sie unter „Farben von Schemablocktrennzeichen“ auf Seite 63.

Beispiel

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

<brackets>

Beschreibung

Eine Liste von Zeichen, die Klammern angeben.

Attribute

`name`, `id`

- `name="`*`bracket_name`*`"` Ein String, der der Liste der Klammern einen Namen zuweist.
- `id="`*`id_string`*`"` Erforderlich. Ein Bezeichnerstring, 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 Codefarbregel für die `@import`-Funktion des `style`-Elements in einem CSS-Stil angibt.

Attribute
`name`, `id`

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

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

<cssMedia/>

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

Attribute
`name`, `id`

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

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

<cssProperty/>

Beschreibung
Ein leeres Tag, das CSS-Regeln angibt und Codefarbattribute enthält.

Attribute

`name`, `id`

- `name="cssProperty_name"` Ein String, der der CSS-Eigenschaft einen Namen zuweist.
- `id="id_string"` Erforderlich. Ein Bezeichnerstring, 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 CSS-Regeln angibt und Codefarbattribute enthält.

Attribute

`name`, `id`

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

Beispiel

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

<cssValue/>

Beschreibung

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

Attribute

`name`, `id`

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

Beispiel

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

<defaultAttribute>

Beschreibung

Optional. Dieses Tag gilt nur für Tag-basierte Syntax (d. h. `ignoreTags="No"`). Wenn dieses Tag vorhanden ist, erhalten alle Tag-Attribute die Farbe des Stils, der diesem Tag zugewiesen ist. Wenn dieses Tag nicht angegeben ist, 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 zum Angeben der Standardfarbe und des Standardstils für Tags in einem Schema.

Attribute
`name`, `id`

- `name="`*`display_name`*`"` Ein String, der in Dreamweaver im Codefarben-Editor angezeigt wird.
- `id="`*`id_string`*`"` Erforderlich. Ein Bezeichnerstring, 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 Tag-Definition die Farbe des Stils, der diesem Tag zugewiesen wurde. Wenn dieses Tag nicht angegeben ist, 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 Bezeichnerstring, 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 und Farbattribute enthalten.

Attribute
`name`, `id`

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

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

<functionKeyword>

Beschreibung
Gibt Schlüsselwörter an, 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 Bezeichnerstring, 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
`includeAlpha`, `includeDecimal`, `includeHiAscii`

Es handelt sich durchweg um logische Attribute (Boolean), die den Wert „true“ oder „false“ annehmen können.

Beispiel
```
<identifier name="Identifier" id="CodeColor_JavascriptIdentifier">
<idChar1>_$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ</idChar1> </identifier>
```

<idCharRest>

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

Attribute

`includeAlpha`, `includeDecimal`, `includeHiAscii`

Es handelt sich durchweg um logische Attribute (Boolean), die den Wert „true“ oder „false“ annehmen können.

Beispiel

```
<identifier name="Identifier" id="CodeColor_JavascriptIdentifier"> <idCharRest>
    _$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</idCharRest>
</identifier>
```

<identifier>

Beschreibung

Die Knoten `<idChar1>` und `<idCharRest>` müssen von einem `<identifier>`-Knoten eingeschlossen sein. Die Attribute `name` und `id` müssen von `<idCharRest>`/`<idChar1>` in den umgebenden `<identifier>`-Knoten verlagert werden.

Wenn Sie mehrere Bezeichnertypen definieren, müssen Sie bei allen diesen Bezeichnern einen eindeutigen Wert für das Attribut `idChar1` angeben. Wird diese Regel nicht eingehalten, wird für Bezeichner, die mit einem Zeichen beginnen, das in zwei überlappenden Stilen vorkommt, eine zufällige Farbe verwendet. Beispiel:

```
 <identifier name="N1" id="I1">
       <idchar1>a</idchar1>
       <idcharrest includeAlpha="true" includeDecimal="true"
includeHiAscii="true">_</idCharRest>
 </identifier>
 <identifier name="N2" id="I2">
       <idchar1  includeAlpha="true" >$_</idchar1>  //  ERROR!!!  text that starts with "a"
can be either N1 or N2, the coloring will be random for it!
       <idcharrest includeAlpha="true" includeDecimal="true"
includeHiAscii="true">_</idCharRest>
 </identifier>
```

Die vor Dreamweaver CS5 geltende Syntax (ohne den `<identifier>`-Knoten) wird weiterhin unterstützt. Allerdings dürfen Sie innerhalb einer bestimmten Datei die alte und die neue Syntax nicht vermischen. Wenn Sie Stile mischen, werden Stile mit der alten Syntax ignoriert. Wenn das Syntaxschema also einen `<identifier>`-Knoten enthält, werden alle `idChar1`/`idCharRest`-Knoten, die nicht in einen `<identifier>`-Knoten eingeschlossen sind, ignoriert.

Attribute

`name`, `id`

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

Beispiel

```
<identifier name="Identifier" id="CodeColor_JavascriptIdentifier"> <idCharRest>

_$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789</idCharRest></identifier
```

<ignoreCase>

Beschreibung
Gibt an, ob die Groß- und Kleinschreibung beim Vergleichen von Token und Schlüsselwörtern ignoriert werden soll. Zulässige Werte sind `Yes` und `No`. Der Standardwert 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`. Der Standardwert ist `Yes`. Dies dient zur korrekten Farbzuweisung 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`. Der Standardwert ist `Yes`. Legen Sie `No` fest, wenn die Syntax für eine Markup-Sprache mit den Trennzeichen `<` und `>` verwendet 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 ist. Zulässige Werte sind `Yes` und `No`. Der Standardwert ist `No`.

Attribute
Keine.

Beispiel

```
<isLocked>Yes</isLocked>
```

<keyword>

Beschreibung

Ein Textstring, der ein Schlüsselwort definiert. 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 den im Kategorieattribut angegebenen Typ. 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 Bezeichnerstring, 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 Ziffern 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 Bezeichnerstring, 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 Bezeichnerstring, 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 Bezeichnerstring, der diesem Syntaxelement Farbe und Stil zuordnet.
- `delimiter` Das Zeichen oder der String, das bzw. der den Anfang und das Ende eines regulären Ausdrucks kennzeichnet.
- `escape` Das Zeichen oder der String, das bzw. der eine besondere Zeichenverarbeitung angibt, das sogenannte 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

Beispieltext, der im Vorschaufenster des Dialogfelds „Farbschema bearbeiten“ angezeigt wird. Weitere Informationen zum Dialogfeld „Farbschema bearbeiten“ finden Sie unter „Bearbeiten von Schemas“ auf Seite 69.

Attribute

`doctypes`

- `doctypes="doc_type1, doc_type2,..."` Die Dokumenttypen, für die der Beispieltext angezeigt wird.

Beispiel

```
<sampleText doctypes="JavaScript"><![CDATA[/* JavaScript */
function displayWords(arrayWords) {
    for (i=0; i < arrayWords.length(); i++) {
        // inline comment
        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

Dieses Tag 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 Bezeichnerstring, der diesem Syntaxelement Farbe und Stil zuordnet.
- `wrap="true"` oder `"false"`. Gibt an, ob bei Codefarben Textstrings erkannt werden, die auf die nächste Zeile umgebrochen wurden. Der Standardwert 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>
```

<tagGroup>

Beschreibung
Dieses Tag gruppiert ein oder mehrere Tags, denen Sie eine individuelle Farbe und einen individuellen Stil zuweisen können.

Attribute
`id`, `name`, `taglibrary`, `tags`

- `id="`*`id_string`*`"` Erforderlich. Ein Bezeichnerstring, der diesem Syntaxelement Farbe und Stil zuordnet.
- `name="`*`display_name`*`"` Ein String, der in Dreamweaver im Codefarben-Editor angezeigt wird.
- `taglibrary="`*`tag_library_id`*`"` Der Bezeichner der Tag-Bibliothek, zu der diese Gruppe von Tags gehört.
- `tags="`*tag_list*" Eine durch Tags oder Kommas getrennte Liste von Tags, die die Tag-Gruppe bilden.

Beispiel
```
<tagGroup name="HTML Table Tags" id="CodeColor_HTMLTable" taglibrary="DWTagLibrary_html"
    tags="table,tbody,td,tfoot,th,thead,tr,vspec,colw,hspec" />
```

Farben von Schemablocktrennzeichen

Das Schemaattribut `blockStart` definiert die Farbe der Anfangs- und Endstrings von Blöcken oder Blocktrennzeichen. Für das Attribut `blockStart` sind die folgenden Werte zulässig.

Hinweis: *Verwechseln Sie das Attribut* `blockStart.scheme` *nicht mit dem* `scheme`*-Tag.*

innerText

Dieser Wert weist Dreamweaver an, die Blocktrennzeichen mit der gleichen Farbe wie den Standardtext des Schemas innerhalb der Blocktrennzeichen zu kennzeichnen.

Das Template-Schema ist ein Beispiel für die Darstellung dieses Schemas. Im Template-Schema werden Blöcke mit schreibgeschütztem Code grau dargestellt, da sie nicht bearbeitet werden können. Die Blocktrennzeichen, d. h. die Strings `<!#EndEditable -->`-- und `<!-- #BeginEditable "..." -->`, werden ebenfalls grau angezeigt, da sie auch nicht bearbeitet werden können.

Beispielcode

```
<!-- #EndEditable -->
    <p><b><font size="+2">header</font></b></p>
    <!-- #BeginEditable "test" -->
    <p>Here's some editable text </p>
    <p> </p>
    <!-- #EndEditable -->
```

Beispiel

```
<blockStart doctypes="ASP-JS,ASP-VB, ASP.NET_CSharp, ASP.NET_VB, ColdFusion,CFC, HTML,
    JSP,LibraryItem,PHP_MySQL" scheme="innerText"><![CDATA[<!--\s*#BeginTemplate]]>
    </blockStart>
```

customText

Dieser Wert weist Dreamweaver an, benutzerdefinierte Farben für die Blocktrennzeichen zu verwenden.

Beispielcode

Ein Beispiel für die Darstellung des `customText`-Werts sind die Trennzeichen für PHP-Skriptblöcke, die rot angezeigt werden:

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

Beispiel

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

outerTag

Der `outerTag`-Wert gibt an, dass die Tags `blockStart` und `blockEnd` vollständige Tags sind. Der Wert legt fest, dass die Tags mit der gleichen Farbe wie die Tags des Schemas außerhalb der Tags gekennzeichnet werden sollen.

Ein Beispiel für diesen Wert ist das JavaScript-Schema, in dem die Strings `<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 Schema außerhalb der Trennzeichen farblich gekennzeichnet werden.

Beispielcode

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

Beispiel

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

innerTag

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

nameTag

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

Mit diesem Schematyp werden Tags angezeigt, die in andere Tags eingebettet werden können, z. B. das `cfoutput`-Tag.

Beispielcode

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

Beispiel

```
<blockStart doctypes="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, beispielsweise Zuordnungsanweisungen oder -ausdrücken.

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

Beispielcode

Siehe Beispieltext für „nameTag“ auf Seite 65.

Beispiel

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

Schemaverarbeitung

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

In jedem Modus werden Codefarben nur bestimmten Feldern zugewiesen. In der folgenden Tabelle sind die Felder aufgeführt, denen im entsprechenden Modus jeweils Codefarben zugewiesen werden.

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 Definition von Schemas flexibler zu gestalten, können Sie in Dreamweaver bestimmte Platzhalter- und Codewechselzeichen (Escape-Zeichen) angeben.

Platzhalterzeichen

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

Platzhalter	Escape-String	Beschreibung
Platzhalter	*	Alle Zeichen in der Regel werden übersprungen, bis das Zeichen nach dem Platzhalter gefunden wurde. Mit `<MMTInstance:Editable name="\*">` werden beispielsweise 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 entspricht dem Platzhalter, jedoch kann zusätzlich ein Codewechselzeichen (Escape-Zeichen) angegeben 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. `/\e*\\/` wird beispielsweise zur Erkennung 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 XML-Code für die Codefarbe angeben.
Platzhalter für optionale Leerräume	\s*	Entspricht einer beliebigen Anzahl von Zeichen für Leerräume oder Zeilenumbrüche, einschließlich der Anzahl 0. Mit `<!--\s*#include` werden beispielsweise ASP-include-Anweisungen gefunden, unabhängig davon, ob vor dem `#include`-Token ein Leerraum angegeben ist, da beide Fälle gültig sind. Die Platzhalter für Leerräume gelten für alle Kombinationen aus Zeichen für Leerräume und Zeilenumbrüche.
Platzhalter für erforderliche Leerräume	\s+	Entspricht einem oder mehreren Zeichen für Leerräume oder Zeilenumbrüche. Mit `<!--#include\s+virtual` werden beispielsweise ASP-include-Anweisungen gefunden, unabhängig davon, in welcher Kombination Leerräume zwischen `#include` und `virtual` angegeben sind. Es muss ein Leerraum zwischen diesen Token angegeben werden, dieser kann jedoch aus einer beliebigen Kombination gültiger Zeichen für Leerräume bestehen. Die Platzhalter für Leerräume gelten für alle Kombinationen aus Zeichen für Leerräume und Zeilenumbrüchen.

Escape-Zeichen

Die folgende Aufstellung enthält die von Dreamweaver unterstützten Codewechselzeichen (Escape-Zeichen) sowie die spezifischen Strings und eine Beschreibung ihrer Verwendung.

Escape-Zeichen	Escape-String	Beschreibung
Umgekehrter Schrägstrich	\\	Der umgekehrte Schrägstrich (\) ist das Escape-Zeichen für Codefarben. Daher muss er bei Verwendung in einer Codefarbregel angegeben werden.
Leerraum	\s	Dieses Escape-Zeichen entspricht den nicht sichtbaren Zeichen, mit Ausnahme der als Zeichen für Zeilenumbrüche aufgelisteten Escape-Zeichen, z. B. Leerzeichen und Tabulatorzeichen. Mit den Platzhaltern für optionale und erforderliche Leerräume werden die Zeichen für Leerräume und Zeilenumbrüche gefunden.
Zeilenumbruch	\n	Dieses Escape-Zeichen entspricht den Zeichen für Zeilenumbruch (auch als Zeilenvorschub bezeichnet) und Wagenrücklauf.

Maximale Stringlänge

In Datenstrings können maximal 100 Zeichen angegeben werden. Das folgende `blockEnd`-Tag enthält z. B. ein Platzhalterzeichen.

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

Wenn davon ausgegangen wird, dass die optionalen Platzhalter für Leerräume (`\s*`) einstellige Zeichen sind, die Dreamweaver automatisch generiert, besteht der Datenstring aus 26 Zeichen plus einem Platzhalterstring (`\*`) für den Namen.

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

Der bearbeitbare Bereichsname kann daher bis zu 74 Zeichen lang sein. Dies entspricht der maximalen Länge von 100 Zeichen minus 26.

Schemapriorität

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

1. Anhand des Dokumenttyps der aktuellen Datei wird das anfängliche Syntaxschema ermittelt. Der Dokumenttyp der Datei wird mit dem Attribut `scheme.documentType` verglichen. Wenn keine Übereinstimmung gefunden wurde, wird das Schema `scheme.documentType = "Text"` verwendet.
2. Schemas können verschachtelt werden, wenn sie in `blockStart` … `blockEnd`-Paaren angegeben sind. Alle verschachtelbaren Schemas, für die in einem der `blockStart.doctypes`-Attribute die aktuelle Dateierweiterung aufgeführt ist, sind für die aktuelle Datei aktiviert. Alle anderen Schemas sind deaktiviert.

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

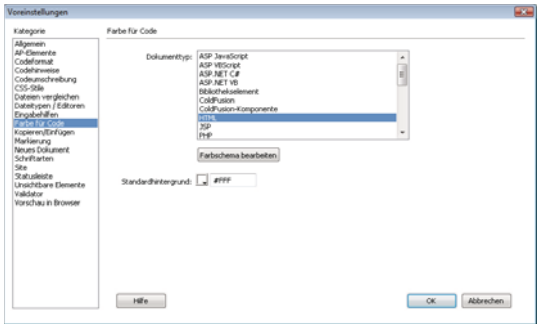
   Ein Schema kann nur in einem anderen Schema verschachtelt werden, wenn die entsprechende Priorität (`scheme.priority`) größer als oder gleich der Priorität des äußeren Schemas ist. Wenn die Priorität gleich ist, kann das Schema nur im body-Bereich des äußeren Schemas verschachtelt werden. Der Block `<script>...</script>` kann beispielsweise im Block `<html>...</html>` nur an einer Position verschachtelt werden, an der Tags zulässig sind, d. h. nicht innerhalb eines Tags, Attributs, Strings, Kommentars usw.

   Schemas mit einer höheren Priorität als das äußere Schema können fast an jeder Stelle im äußeren Schema verschachtelt werden. Zusätzlich zur Verschachtelung im body-Bereich des `<html>...</html>`-Blocks kann der Block `<%...%>` beispielsweise auch innerhalb eines Tags, Attributs, Strings, Kommentars usw. verschachtelt werden.

   Es sind maximal 4 Verschachtelungsebenen zulässig.
3. Beim Vergleichen von `blockStart`-Strings verwendet Dreamweaver immer den längsten übereinstimmenden String.
4. Nach dem `blockEnd`-String für das aktuelle Schema wird die Syntax wieder auf die Farbe zurückgesetzt, die ihr vor dem Erkennen des `blockStart`-Strings zugewiesen war. Wenn ein `<%...%>`-Block beispielsweise in einem HTML-String gefunden wurde, wird anschließend wieder die Farbe des HTML-Strings verwendet.

Bearbeiten von Schemas

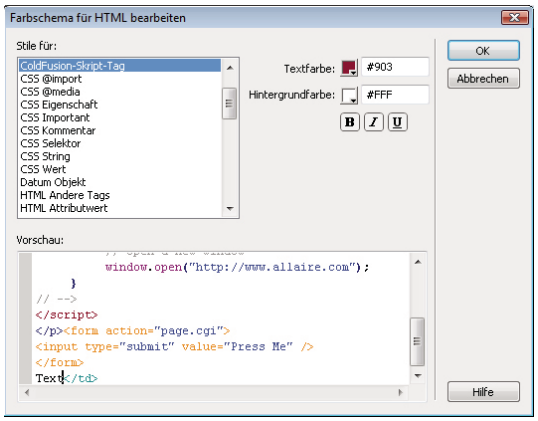
Sie können die Stile für ein Codefarbschema entweder durch Bearbeiten der Codefarbdatei oder durch Auswählen der Kategorie „Farbe für Code“ im Dialogfeld „Voreinstellungen“ von Dreamweaver (siehe folgende Abbildung) anpassen.



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: *Es wird empfohlen, dass Sie eine Sicherungskopie aller XML-Dateien erstellen, bevor Sie Änderungen vornehmen. Überprüfen Sie alle manuellen Änderungen, bevor Sie Farb- und Stileinstellungen im Dialogfeld „Voreinstellungen“ bearbeiten. Wenn Sie im Dialogfeld „Voreinstellungen“ eine ungültige XML-Datei bearbeiten, gehen Daten verloren.*

Um die Stile eines Schemas in der Kategorie „Farben für Code“ des Dialogfelds „Voreinstellungen“ zu bearbeiten, doppelklicken Sie auf einen Dokumenttyp oder klicken Sie auf die Schaltfläche „Farbschema bearbeiten“, um das Dialogfeld „Farbschema für HTML bearbeiten“ zu öffnen.



Wenn Sie den Stil eines bestimmten Elements bearbeiten möchten, wählen Sie das Element in der Liste „Stile für“ aus. Zu den im Fenster „Stile für“ aufgelisteten Elementen gehören auch die Felder für das bearbeitete Schema und die Schemas, die als Blöcke in diesem Schema vorhanden sind. Wenn Sie beispielsweise das HTML-Schema bearbeiten, sind 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“ ist der Wert für das Attribut `scheme.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 ein Beispieltext mit den aktuellen Einstellungen angezeigt. Der Beispieltext wird aus der `sampleText`-Einstellung des Schemas übernommen.

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

Wenn Sie die Einstellung für ein Element eines Schemas ändern, wird der Wert in der Codefarbdatei gespeichert und die ursprüngliche Einstellung überschrieben. Wenn Sie auf „OK“ klicken, lädt Dreamweaver alle Änderungen an den Codefarben automatisch neu.

Beispiele für Codefarben

Mit den folgenden Beispielen für Codefarben werden die Codefarbschemas für ein CSS-Dokument und ein JavaScript-Dokument angegeben. 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

Im folgenden Beispieltext für das CSS-Schema ist das CSS-Codefarbschema angegeben:

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

Die folgenden Zeilen aus der Datei „Colors.xml“ geben die Farb- und Stilwerte an, die im Beispieltext angezeigt werden und mit dem 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" />
        <regexp name="RegExp" id="CodeColor_JavascriptRegexp" delimiter="/" escape="\\">
            <searchPattern><![CDATA[(\s*/\e*\\/]]></searchPattern>
            <searchPattern><![CDATA[=\s*/\e*\\/]]></searchPattern>
        </regexp>
        <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 comment
        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 comment
        alert("Word " + i + " is " + arrayWords[i]);
    }
}
var tokens = new Array("Hello", "world");
displayWords(tokens);
```

Die folgenden Zeilen aus der Datei „Colors.xml“ geben die Farb- und Stilwerte an, die im Beispieltext angezeigt werden und mit dem 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. Fehler werden mit einer roten Wellenlinie unterstrichen.

Dreamweaver verfügt auch über eine neue Funktion zur Browserkompatibilitätsprüfung, mit der Kombinationen von HTML- und CSS-Code gesucht werden, die zu Problemen bei der Browserdarstellung führen können.

Die Browserprofile werden im Dreamweaver-Ordner „Configuration/BrowserProfiles“ gespeichert. Jedes Browserprofil ist als Textdatei definiert, die nach dem jeweiligen Browser benannt ist. Das Browserprofil für Internet Explorer 6.0 trägt beispielsweise den Namen „Internet_Explorer_6.0.txt“. Um die Prüfung von Zielbrowsern für CSS zu ermöglichen, speichert Dreamweaver Informationen zu CSS-Profilen für Browser in einer XML-Datei, deren Name dem Browserprofil entspricht, jedoch mit dem Suffix „_CSS.xml“. Das CSS-Profil für Internet Explorer 6.0 ist beispielsweise „Internet_Explorer_6.0_CSS.xml“. Wenn in Dreamweaver Fehler ausgegeben werden, können Sie die entsprechenden Änderungen an der CSS-Profildatei vornehmen.

Die CSS-Profildatei enthält drei XML-Tags: `css-support`, `property` und `value`. Diese Tags werden im Folgenden 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
Keine.

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 die Unterstützung angeben.
- `names="`*`property_name`*`,`*`property_name,...`*`"` Eine durch Kommas getrennte Liste mit Eigenschaftennamen, für die Sie die Unterstützung angeben.

  Das Attribut `names` ist eine Art Kurzschrift. Das folgende `names`-Attribut ist beispielsweise 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 den Grad der Unterstützung für die Eigenschaft an. Wenn kein Wert angegeben ist, wird von `supported` ausgegangen. Wenn Sie einen anderen Unterstützungsgrad als "`supported`" ohne das message-Attribut angeben, verwendet Dreamweaver die Standardmeldung „Die CSS-Eigenschaft *Eigenschaftenname* wird nicht unterstützt“.
- `message="message_string"` Das Argument `message` definiert eine Meldung, die in Dreamweaver angezeigt wird, wenn die Eigenschaft in einem Dokument gefunden wird. Diese Meldung beschreibt mögliche Einschränkungen oder Problemvermeidungsmaßnahmen für den Eigenschaftenwert.

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"` oder `"color"` verwenden, müssen über das Attribut `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 angegebenen Wert für die Einheit.
- `name="value_name"` Ein Bezeichner für CSS-Werte. Mit Ausnahme von Bindestrichen (-) sind keine Leerzeichen oder Satzzeichen zulässig. Der Name eines der Werte, die für die CSS-Eigenschaft gültig sind, ist im übergeordneten Knoten der Eigenschaft angegeben. Dabei kann es sich um einen bestimmten Wert oder eine Maßeinheit handeln.
- `names="name1, name2, ...`.Gibt eine durch Kommas getrennte Liste mit Wert-`IDs` an.
- `supportlevel="error", "warning", "info"`oder `"supported"` Gibt den Unterstützungsgrad für diesen Wert im Browser an. Wenn kein Wert angegeben ist, wird vom Wert `"supported"` ausgegangen.

- `message="message_string"` Das Attribut `message` definiert eine Meldung, die angezeigt wird, wenn der Eigenschaftenwert in einem Dokument gefunden wird. Wenn das Attribut `message` nicht angegeben ist, wird die Meldung „*Name des Werts* wird nicht unterstützt“ angezeigt.

Inhalt

Keine.

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>
```

Ändern der HTML-Standardformatierung

Die allgemeinen Einstellungen für die Codeformatierung können Sie in der Kategorie „Codeformat“ des Dialogfelds „Voreinstellungen“ ändern. Das Format bestimmter Tags und Attribute können Sie im Tag-Bibliothek-Editor („Bearbeiten“ > „Tag-Bibliotheken“) ändern. 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 die Datei „TagLibraries.vtm“ bearbeiten. Dreamweaver ignoriert alle VTM-Dateien, die nicht in der Datei „TagLibraries.vtm“ aufgeführt sind.

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

Vertikal geteilte Ansicht

Mit der Funktion für die vertikal geteilte Ansicht können entweder die Code- und die Entwurfsansicht oder die Codeansicht und der Layoutmodus nebeneinander angezeigt werden. Benutzer mit Dual-Screen-Arbeitsstationen können mithilfe dieser Funktion auf einem Monitor die Codeansicht anzeigen und auf dem zweiten Monitor in der Entwurfsansicht arbeiten.

Mit der Funktion für die vertikal geteilte Ansicht können Benutzer folgende Aktionen ausführen:

- Auswählen der Ausrichtung der Code- und Entwurfsansicht (horizontal oder vertikal)
- Wechseln zwischen horizontaler und vertikaler Ausrichtung der Code- und Entwurfsansicht und der geteilten Codeansicht

Beim Neustarten von Dreamweaver und Öffnen oder Erstellen eines Dokuments wird das Dokument in der Code- und Entwurfsansicht mit der zuletzt verwendeten Größe und Ausrichtung angezeigt. Die Funktion `dreamweaver.setSplitViewOrientation()` legt dabei die Ausrichtung fest und die Funktion `dreamweaver.setPrimaryView()` die primäre Ansicht. Informationen zur Verwendung dieser Funktionen finden Sie unter „Funktionen für die vertikal geteilte Ansicht“ im *Dreamweaver API-Referenzhandbuch*.

Zugehörige Dateien

Mit der Funktion für zugehörige Dateien können Benutzer auf die unterstützenden und zugehörigen Dateien zugreifen, die mit der bearbeiteten Datei verknüpft sind. Zugehörige Dateien können CSS-, Skript-, SSI- (Server-Side Include) oder XML-Dateien sein.

Wenn beispielsweise eine CSS-Datei mit der Hauptdatei verknüpft ist, kann diese CSS-Datei mithilfe dieser Funktion angezeigt und bearbeitet werden. Der Benutzer kann zudem die Hauptdatei anzeigen und gleichzeitig die zugehörige Datei bearbeiten.

Funktionsweise von zugehörigen Dateien

Zugehörige Dateien tragen zur einfacheren Bearbeitung von Dateien bei, indem Benutzern das Ausführen folgender Aufgaben erleichtert wird:

- Benutzer können die Hauptdatei anzeigen und gleichzeitig die zugehörigen Dateien anzeigen und öffnen. Beim Anzeigen einer Seite mit zugehörigen Dateien (z. B. eine CSS-Datei) kann Folgendes angezeigt werden:
  - Entwurf der Webseite auf einer Seite
  - Zugehörige Datei auf der anderen Seite
- Die Symbolleiste für zugehörige Dateien enthält die Dokumente, die sich auf das Generieren von übergeordnetem HTML-Code auswirken. Benutzer können den HTML-Quellcode, den generierten HTML-Code und die untergeordneten Dokumente der ersten Ebene anzeigen.
- Durch Auswählen einer zugehörigen Datei in der Symbolleiste für zugehörige Dateien können Benutzer die folgenden Aktionen ausführen:
  - Anzeigen und Bearbeiten der zugehörigen Datei in der Codeansicht
  - Anzeigen der übergeordneten Seite in der Entwurfsansicht
- Durch Auswählen von Inhalten in der Entwurfsansicht und durch Ändern der zugehörigen Datei wird die Auswahl nicht aufgehoben, wenn der Benutzer die Entwurfsansicht aktualisiert.
- Wenn Sie den Code der zugehörigen Datei ändern, werden die Änderungen in der Entwurfsansicht dargestellt.

Kann eine Datei nicht gefunden werden, wird eine entsprechende Meldung in einer Leiste am oberen Rand des leeren Fensterrahmens angezeigt.

Begriffserklärungen im Zusammenhang mit zugehörigen Dateien

Im Zusammenhang mit zugehörigen Dateien werden häufig die folgenden Begriffe verwendet:

Begriff	Beschreibung	Beispiel
Dokument der obersten Ebene	Jedes vom Benutzer geöffnete Dokument.	
Übergeordnetes Dokument	Jedes Dokument der obersten Ebene, das in der Entwurfsansicht dargestellt wird.	• HTML, einschließlich .lbi, .dwt • CFML • PHP
Untergeordnetes Dokument der ersten Ebene	Jedes Dokument, das sich eine Ebene unter dem übergeordneten Dokument befindet. Diese Dokumente wirken sich auf das Generieren von HTML-Code aus, mit der Ausnahme von CSS-Dateien. CSS-Dateien können andere CSS-Dateien beinhalten. Alle CSS-Dateien zusammen legen jedoch die endgültigen Stile einer Seite fest.	• Skriptdatei, angegeben durch <SCRIPT src="file.js"> • Server-Side Include • Externe CSS-Datei • Spry XML- und HTML-Datensätze • Bibliothekselement • <iframe> – Remote-Quelle • object-Tag
Untergeordnetes Dokument auf unterer Ebene	Jedes Dokument, das sich mehr als zwei Ebenen unter dem übergeordneten Dokument befindet. Diese Dokumente wirken sich auf das Generieren von HTML-Code aus.	• PHP innerhalb von PHP • DTDs • Vorlagen
Nicht zugehörige Datei	Jedes Dokument, das sich nicht auf das Generieren von HTML-Code oder auf Dateien auswirkt, die ein Benutzer nicht aktiv bearbeitet.	• Bilddateien • Mediendateien • Extern durch ein <a>-Tag verknüpfte Dateien

Die folgenden zugehörigen Dateien werden unterstützt:

Typ	Beschreibung	Verschachtelungsebene
Client-Skript	Alle Sprachen	1 (Skriptverschachtelung nicht möglich)
Server-Side Includes	Wenn alle der folgenden erweiterbaren Bedingungen erfüllt sind: • Definiertes Servermodell • Definierte SSI-Anweisung (d. h. Muster) • Definierter DW-Dokumenttyp Ausnahme: Include-Anweisungen im Apache-Format (`<!--#include ... -->`) in HTML-Dokumenten werden erkannt.	1
Spry-Datensatz		1 (Skriptverschachtelung nicht möglich)
CSS	• Alle externen CSS für alle Medientypen • DTSS	Unendlich

APIs für zugehörige Dateien

Sie können das Menü für zugehörige Dateien anpassen, um Folgendes anzuzeigen:

- Dateinamen der zugehörigen Datei
- HTML-Quellcode und generierter Quellcode

Die Funktion `dreamweaver.openRelatedFile()` zeigt die zugehörige Datei in der Codeansicht an und die Funktion `dreamweaver.getActiveRelatedFilePath()` zeigt den Pfad der geöffneten zugehörigen Datei an. Weitere Informationen zur Verwendung dieser APIs finden Sie unter „Funktionen für zugehörige Dateien“ im *Dreamweaver API-Referenzhandbuch*.

Live-Ansicht

Mit der Funktion für die Live-Ansicht können Sie Ihre Webseiten in der Vorschau anzeigen, wie sie in einem Browser dargestellt werden, ohne Dreamweaver zu beenden. Benutzer können weiterhin direkt auf den Code zugreifen und diesen bearbeiten. Alle Änderungen am Code werden unmittelbar dargestellt. Mit dieser Funktion können Benutzer die geänderte Webseite sofort anzeigen. Wenn der Benutzer CSS-Dateien bearbeitet, wird der aktuelle Status der Datei beibehalten, die CSS-Änderungen werden jedoch angewendet. Benutzer können zudem mit der Seite interagieren und JavaScript-Effekte wie z. B. Rollover-Effekte anzeigen, ohne von Dreamweaver in einen Webbrowser wechseln zu müssen.

Die Funktion für die Live-Ansicht verwendet das Flash-System-Plug-In (%SYSTEM%/Macromed/Flash, /Library/Internet Plug-Ins/). In manchen Fällen ist es auf die Firefox-Version zurückzuführen, wenn die Systemversion des Plug-Ins nicht verfügbar ist.

In der Statusleiste wird eine Meldung angezeigt, wenn das Plug-In nicht gefunden wird. Im Bedienfeld „CSS“ wird immer der aktuelle relevante CSS-Stil für das in der Live-Ansicht angezeigte Element angezeigt, auch wenn die Quelle über einen anderen Speicherort generiert wird. Benutzer können Stylesheets hinzufügen oder entfernen. Andere Bearbeitungen von Inline-CSS oder CSS im <head>-Tag sind jedoch gesperrt. Die Regeln im Bedienfeld „CSS“, die nicht bearbeitet werden können, sind als schreibgeschützt gekennzeichnet.

Sie können die Dreamweaver-API zu folgenden Zwecken verwenden:

- Abrufen und Festlegen der Entwurfsansicht
- Abrufen und Festlegen der Live-Ansicht unter Verwendung des Servers
- Abrufen von Standardwerten für die Live-Ansicht
- Abrufen und Festlegen abhängiger Dateien für die Live-Ansicht
- Anzeigen der Parameter für die Live-Ansicht

Weitere Informationen zu diesen APIs finden Sie unter „Live-Ansichtsfunktionen“ im *Dreamweaver API-Referenzhandbuch*.

Einfaches Beispiel für die Live-Ansicht

In diesem Beispiel erstellen Sie mithilfe von Dreamweaver einen Befehl, mit dem ein einfacher Browser erstellt wird, wenn der Benutzer auf den entsprechenden Befehl im Menü „Befehle“ klickt. Wenn Sie vor dem Testen dieses Beispiels weitere Informationen zum Erstellen von Befehlen benötigen, finden Sie diese unter „Befehle“ auf Seite 141 auf Seite 142. Öffnen Sie in Dreamweaver eine neue einfache HTML-Datei (dies ist Ihre Befehlsdefinitionsdatei) und speichern Sie die Datei unter dem Namen „liveviewexample.htm“. Die Befehlsdefinitionsdatei sieht wie folgt aus:

```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0// dialog"> <html
xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Browser Test</title>
<script type="text/javascript">
var browserLoad = function(e)
{
    var uri = document.getElementById("uri");
    uri.value = e.currentBrowserLocation;
}
var promptToNavigate = function (e)
{
    if( ! confirm(" Is it ok to go from \n" + e.currentBrowserLocation + " to page \n " +
e.requestedBrowserLocation ) )
    {
        e.preventDefault();
    }
}
function initUI()
{
    var browser = document.getElementById("browser");
    browser.addEventListener("BrowserControlBeforeNavigation",
promptToNavigate, true);
    browser.addEventListener("BrowserControlLoad", browserLoad, true);
    browser.openURL("http://www.adobe.com");
}
function loadUri()
{
    var uri = document.getElementById("uri");
    var browser = document.getElementById("browser");
    browser.openURL(uri.value);
}
function showSource(){
    var browser = document.getElementById("browser");
    alert(browser.getWindowObj().document.documentElement.outerHTML);
}
function commandButtons() {
                            return new Array( "Close", "window.close()",
                            "Load URI", "loadUri()",
                            "Show Source", "showSource()"
);
}
</script>
</head>
<body onLoad="initUI()">
<form>
<p>
<label>
<input id="uri" type="text" style="width:500px">
</label>
</p> <mm:browsercontrol id="browser" style="width:500px; height:300px;"/> </form> </body>
</html>
```

Kapitel 4: Erweitern von Dreamweaver

Dreamweaver bietet 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 „Erstellen von Erweiterungen“ auf Seite 2 vor.

Mithilfe der folgenden Funktionen von Dreamweaver können Sie Erweiterungen erstellen:

- Ein HTML-Parser (auch als *Renderer* bezeichnet) ermöglicht das Entwerfen von Benutzeroberflächen für Erweiterungen. Der Parser verwendet dazu Formularfelder, absolut positionierte Elemente, Bilder und andere HTML-Elemente. Dreamweaver verfügt über einen eigenen HTML-Parser.
- Eine Ordnerstruktur, in der die Dateien verwaltet und gespeichert werden, die Elemente und Erweiterungen von Dreamweaver implementieren und konfigurieren.
- Eine Reihe von APIs (Anwendungsprogrammierschnittstellen), die den Zugriff auf die Dreamweaver-Funktionalität über JavaScript ermöglichen.
- Ein JavaScript-Interpreter, der den JavaScript-Code in Erweiterungsdateien ausführt. Dreamweaver verwendet den JavaScript-Interpreter von Netscape in der Version 1.5. Weitere Informationen zu den Unterschieden dieser Interpreter-Version gegenüber früheren Versionen finden Sie unter „Verarbeitung von JavaScript in Erweiterungen durch Dreamweaver“ auf Seite 85.

Typen von Dreamweaver-Erweiterungen

In der folgenden Aufstellung werden die Dreamweaver-Erweiterungen beschrieben:

Einfügeleistenobjekt Diese Erweiterungen erstellen Änderungen in der Einfügeleiste. Ein Objekt wird in der Regel dazu verwendet, das Einfügen von Code in ein Dokument zu automatisieren. Es kann darüber hinaus ein Formular für Benutzereingaben und JavaScript-Code zum Verarbeiten der Eingabe enthalten. Objektdateien werden im Ordner „Configuration/Objects“ gespeichert.

Befehl Diese Erweiterungen können fast alle Arten von Aufgaben ausführen – mit oder ohne Benutzereingabe. Befehlsdateien werden meistens über das Menü „Befehle“ gestartet, können jedoch auch von anderen Erweiterungen aufgerufen werden. Befehlsdateien werden im Ordner „Configuration/Commands“ gespeichert.

Menübefehl Diese Erweiterungen erweitern die API für Befehle um Funktionen im Zusammenhang mit dem Aufrufen von Befehlen aus Menüs. Mithilfe der API für Menübefehle können Sie außerdem ein dynamisches Untermenü erstellen.

Symbolleiste Diese Erweiterungen dienen dazu, in der Benutzeroberfläche von Dreamweaver den Symbolleisten neue Elemente hinzuzufügen oder neue Symbolleisten zu erstellen. Neue Symbolleisten werden unterhalb der Standardsymbolleiste angezeigt. Symbolleistendateien werden im Ordner „Configuration/Toolbars“ gespeichert.

Bericht Diese Erweiterungen dienen zum Hinzufügen benutzerdefinierter Site-Berichte oder zum Ändern der mit Dreamweaver gelieferten vordefinierten Berichte. Mit der API für das Ergebnisfenster können auch eigenständige Berichte erstellt werden.

Tag-Bibliothek und Tag-Editor Diese Erweiterungen werden mit den entsprechenden Tag-Bibliotheksdateien verwendet. Erweiterungen für Tag-Bibliotheken und den Tag-Editor 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.

Eigenschafteninspektor Diese Erweiterungen werden im Bedienfeld des Eigenschafteninspektors angezeigt. Die meisten Inspektoren in Dreamweaver sind Bestandteil des Kernprogramms der Anwendung. Sie können nicht geändert werden. Durch benutzerdefinierte Dateien für die Eigenschafteninspektoren können Sie diese Benutzeroberflächen jedoch ersetzen oder neue Oberflächen zum Prüfen benutzerdefinierter Tags erstellen. Inspektoren werden im Ordner „Configuration/Inspectors“ gespeichert.

Schwebendes Bedienfeld Diese Erweiterungen dienen dazu, der Dreamweaver-Benutzeroberfläche schwebende Bedienfelder hinzuzufügen. Schwebende Bedienfelder können die Auswahl, das Dokument oder die Aufgabe beeinflussen. Außerdem können sie nützliche Informationen anzeigen. Dateien für schwebende Bedienfelder werden im Ordner „Configuration/Floaters“ gespeichert.

Verhalten Mit diesen Erweiterungen können Benutzer ihren Dokumenten JavaScript-Code hinzufügen. Der JavaScript-Code führt als Reaktion auf ein Ereignis eine bestimmte Aktion aus, wenn das Dokument in einem Browser angezeigt wird. Verhaltenserweiterungen werden im Menü mit dem Pluszeichen (+) des Dreamweaver-Bedienfelds „Verhalten“ angezeigt. Die Verhaltendateien werden im Ordner „Configuration/Behaviors/Actions“ gespeichert.

Serververhalten Diese Erweiterungen fügen dem Dokument Blöcke mit serverbasiertem Code (ASP oder ColdFusion) hinzu. Serverbasierter Code führt Aufgaben auf dem Server aus, wenn das Dokument in einem Browser angezeigt wird. Serververhalten werden im Menü mit dem Pluszeichen (+) des Dreamweaver-Bedienfelds „Serververhalten“ angezeigt. Serververhaltendateien werden im Ordner „Configuration/Server Behaviors“ gespeichert.

Datenquelle Diese Erweiterungen 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/Data Sources“ gespeichert.

Serverformat Mit diesen Erweiterungen können Sie die Formatierung von dynamischen Daten festlegen.

Komponente Mit diesen Erweiterungen können Sie dem Bedienfeld „Komponenten“ neue Komponententypen hinzufügen. In Dreamweaver werden mit dem Begriff *Komponenten* einige der häufiger verwendeten und modernen Kapselungsstrategien bezeichnet, z. B. ColdFusion-Komponenten (CFCs).

Servermodell Diese Erweiterungen ermöglichen das Hinzufügen von Unterstützung für neue Servermodelle. Dreamweaver unterstützt die gängigsten Servermodelle (ASP, JSP, ColdFusion, PHP und ASP.NET). Servermodellerweiterungen werden somit nur für benutzerdefinierte Serverlösungen, andere Sprachen oder einen benutzerdefinierten Server benötigt. Servermodelldateien werden im Ordner „Configuration/ServerModels“ gespeichert.

Datenübersetzer Diese Erweiterungen 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 zum Erweitern von Dreamweaver

Sie können zudem die folgenden Elemente von Dreamweaver erweitern, um die Möglichkeiten der Anwendung zu erweitern oder sie an Ihre Anforderungen anzupassen.

Dokumenttypen Diese definieren, wie Dreamweaver mit unterschiedlichen Servermodellen verwendet werden kann. Informationen zu Dokumenttypen für Servermodelle werden im Ordner „Configuration/DocumentTypes“ gespeichert. Weitere Informationen finden Sie unter „Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 14.

Codefragmente Dies sind wiederverwendbare 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 Ordner „Snippets“ ablegen, um sie zur Verfügung zu stellen.

Codehinweise Dies sind Menüs, die eine schnellere Eingabe ermöglichen, dadurch dass eine Liste mit Strings angezeigt wird, mit denen der eingegebene String vervollständigt werden kann. Wenn einer der Strings im Menü mit den ersten von Ihnen eingegebenen Zeichen übereinstimmt, können Sie ihn durch Auswählen einfügen. Menüs für Codehinweise sind in der Datei „codehints.xml“ im Ordner „Configuration/CodeHints“ definiert. Sie können dieser Datei neue Menüs für Codehinweise für neu definierte Tags oder Funktionen hinzufügen.

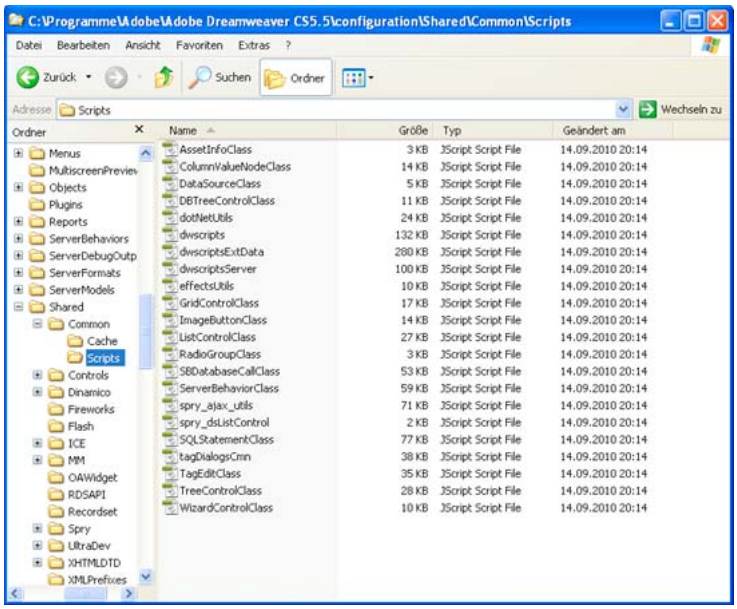
Menüs Diese sind in der Datei „menus.xml“ im Ordner „Configuration/Menus“ definiert. Sie können neue Dreamweaver-Menüs für die Erweiterungen hinzufügen, indem Sie die Menü-Tags für diese Erweiterungen zur Datei „menus.xml“ hinzufügen. Weitere Informationen finden Sie unter „Menüs und Menübefehle“ auf Seite 153.

Konfigurationsordner und Erweiterungen

Die im Konfigurationsordner von Dreamweaver gespeicherten Ordner und Dateien enthalten die mit Dreamweaver gelieferten Erweiterungen. Wenn Sie eine Erweiterung programmieren, müssen Sie die Dateien im richtigen Ordner speichern, damit Dreamweaver sie erkennt. So müssen Sie beispielsweise die Dateien einer neu erstellten Erweiterung für einen Eigenschafteninspektor im Ordner „Configuration/Inspectors“ 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 zum Inhalt der einzelnen Unterordner im Konfigurationsordner finden Sie in der Datei „Configuration_ReadMe.htm“.

Der Ordner „Configuration/Shared“ entspricht keinem bestimmten Erweiterungstyp. Er ist der zentrale Speicherplatz für Dienstprogrammfunktionen, Klassen und Bilder, die von mehreren Erweiterungen 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 hilfreich. Wenn Sie Funktionen suchen, die bestimmte Aufgaben ausführen (z. B. Erstellen eines gültigen DOM-Verweises auf ein Objekt, Ignorieren von Sonderzeichen in Strings oder Überprüfen, ob sich die aktuelle Auswahl in einem bestimmten Tag befindet), sollten Sie zunächst hier nachsehen. Wenn Sie gemeinsam verwendete 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.



Struktur des Ordners „Configuration/Shared/Common/Scripts“

Weitere Informationen zum Ordner „Shared“ finden Sie unter „Ordner „Shared““ auf Seite 391.

Konfigurationsordner bei mehreren Benutzern

Bei den Mehrbenutzer-Betriebssystemen Windows XP, Windows 2000, Windows NT und Mac 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 „Anpassen von Dreamweaver in einer Mehrbenutzerumgebung“ auf Seite 11 und unter „API für Dateizugriff- und Mehrbenutzerkonfiguration“ im *Dreamweaver API-Referenzhandbuch*.

Ausführen von Skripts beim Starten oder Beenden

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 unter „Dokumentobjektmodell von Dreamweaver“ auf Seite 102.

Wenn Sie eine Befehlsdatei im Ordner „Configuration/Shutdown“ speichern, wird diese entsprechend 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 abgebrochen werden.

Weitere Informationen zu Befehlen finden Sie unter „Befehle“ auf Seite 141. Weitere Informationen zur Funktion `dreamweaver.runCommand()` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Erneutes Laden von Erweiterungen

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

Erneutes Laden von Erweiterungen

1 Klicken Sie bei gedrückter Strg-Taste (Windows) bzw. Wahltaste (Macintosh) in der Titelleiste des Bedienfelds „Einfügen“ auf das Menü „Kategorien“.

Hinweis: *Diese Option ist im Registerkartenmodus nicht sichtbar. Klicken Sie im Registerkartenmodus mit der rechten Maustaste auf das Bedienfeldmenü (links oben). Wählen Sie „Als Menü anzeigen“ aus und klicken Sie dann bei gedrückter Strg-Taste auf „Allgemein“, um das Menü mit dem Befehl „Erweiterungen neu laden“ anzuzeigen.*

2 Klicken Sie auf „Erweiterungen neu laden“.

Hinweis: *Bei Mehrbenutzer-Betriebssystemen sollten Sie die Kopien der Konfigurationsdateien in Ihrem Benutzer-Konfigurationsordner bearbeiten, anstatt die Dreamweaver-Konfigurationsdateien zu bearbeiten. Weitere Informationen finden Sie unter „Konfigurationsordner und Erweiterungen“ auf Seite 82.*

APIs für Erweiterungen

Die APIs für Erweiterungen stellen Ihnen die Funktionen zur Verfügung, die Dreamweaver zur Implementierung der einzelnen Erweiterungstypen aufruft. Sie müssen die Hauptteile dieser Funktionen wie für jeden Erweiterungstyp beschrieben verfassen und die erforderlichen, von Dreamweaver erwarteten Rückgabewerte festlegen.

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

In der Dokumentation über APIs für Erweiterungen wird erläutert, welche Aktionen die einzelnen Funktionen beim Aufruf durch Dreamweaver jeweils ausführen und welche Rückgabewerte erwartet werden.

Informationen zur Dienstprogramm-API und zur JavaScript-API, die Funktionen zum Ausführen bestimmter Aufgaben in Erweiterungen enthalten, finden Sie im *Dreamweaver API-Referenzhandbuch*.

Verarbeitung von JavaScript in Erweiterungen durch Dreamweaver

Dreamweaver überprüft beim Start den Ordner „Configuration/*Erweiterungstyp*“. Wenn Dreamweaver in diesem Ordner eine Erweiterungsdatei erkennt, wird der JavaScript-Code 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: *Dieser Ablauf muss beim Start erfolgen, da für einige Erweiterungen möglicherweise globale Variablen initialisiert werden müssen.*

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

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

Hinweis: *Wenn JavaScript-Code in der Erweiterungsdatei das Zeichen " enthält, wird dies vom JavaScript-Interpreter als schließendes `script`-Tag interpretiert und es wird eine Fehlermeldung über einen nicht terminierten String ausgegeben. Sie können dieses Problem vermeiden, indem Sie den String wie folgt unterteilen und die einzelnen Teile miteinander verketten: `"<' + '/SCRIPT>"`.*

Dreamweaver führt den Code in der `onLoad`-Ereignisprozedur aus (sofern diese im `body`-Tag vorhanden ist), wenn der Benutzer den Befehl oder die Aktion in einem Menü für Befehls- oder Verhaltenserweiterungen aufruft.

Dreamweaver führt den Code in der `onLoad`-Ereignisprozedur aus (sofern diese im `body`-Tag vorhanden ist), wenn der body-Bereich des Dokuments ein Formular für Objekterweiterungen enthält.

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

- Datenübersetzer
- Eigenschafteninspektor
- Schwebendes Bedienfeld

Dreamweaver führt bei allen Erweiterungen Code in anderen Ereignisprozeduren aus (z. B. `onBlur="alert('Dieses Feld ist erforderlich.')"`), wenn der Benutzer die Formularfelder bearbeitet, denen sie zugeordnet sind.

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

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

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

Anzeigen von Hilfeinformationen

Wenn Sie die Funktion `displayHelp()`, die Teil verschiedener APIs für Erweiterungen ist, in Ihre Erweiterung einfügen, führt Dreamweaver die beiden folgenden Aktionen aus:

- Hinzufügen einer Hilfeschaltfläche zur Benutzeroberfläche
- Aufrufen der Funktion `displayHelp()`, wenn der Benutzer auf die Hilfeschaltfläche klickt

Sie müssen den Hauptteil der Funktion `displayHelp()` programmieren, damit Hilfeinformationen angezeigt werden. Die Programmierung der Funktion `displayHelp()` legt fest, wie die Erweiterung Hilfeinformationen 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, z. B. die Anzeige von Meldungen in einem anderen absolut positionierten Element in Warnfeldern.

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

```
// The following instance of displayHelp() opens a browser to display a file
// that explains how to use the extension.
function displayHelp() {

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

Lokalisieren von Erweiterungen

Verwenden Sie die folgenden Techniken, um die Übersetzung von 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 Warnmeldungen und den Benutzerschnittstellencode). 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 so die Strings in einer einzigen XML-Datei gemeinsam verwenden. Gegebenenfalls können Sie auch aus C++- und JavaScript-Erweiterungen auf den jeweiligen String verweisen.

Sie können beispielsweise eine Datei mit dem Namen „myExtensionStrings.xml“ erstellen. Das Format der Datei ist im folgenden Beispiel dargestellt:

```
<strings>
    <!-- errors for feature 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 in den String-IDs Schrägstriche (/) verwenden, jedoch keine Leerzeichen. Durch die Verwendung von Schrägstrichen können Sie die gewünschte Hierarchie erstellen und alle Strings in einer einzigen XML-Datei einfügen.

Hinweis: *Dateien im Ordner „Configuration/Strings“, deren Name mit „cc“ beginnt, sind Contribute-Dateien. So handelt es sich z. B. bei der Datei „ccSiteStrings.xml“ 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, an welcher Stelle im String Werte auftreten, und übergeben Sie dann die String- und Variablennamen 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 einzubettenden Variablen an die Funktion `alert()` übergeben wird.

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

Arbeiten mit dem Extension Manager

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

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

Kapitel 5: Benutzeroberflächen für Erweiterungen

Die meisten Erweiterungen sind so konfiguriert, dass Benutzer Informationen über eine Benutzeroberfläche eingeben. Wenn Sie beispielsweise eine Eigenschafteninspektor-Erweiterung für das Tag `marquee` erstellen, müssen Sie Benutzern die Möglichkeit geben, Attribute wie die Richtung oder Höhe festzulegen. Wenn Sie Ihre Erweiterungen zur Zertifizierung an Adobe senden möchten, müssen Sie die Richtlinien befolgen, die in den Extension Manager-Dateien auf der Adobe Exchange-Website (www.adobe.com/go/exchange_de) verfügbar sind. Der Zweck dieser Richtlinien ist es nicht, Ihre Kreativität einzuschränken. Sie sollen nur gewährleisten, dass zertifizierte Erweiterungen in der Benutzeroberfläche von Adobe Dreamweaver effektiv funktionieren und die Funktionalität der Benutzeroberfläche für Erweiterungen nicht beeinträchtigt wird.

Entwurfsrichtlinien für Benutzeroberflächen von Erweiterungen

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

So können 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, die Abläufe zum Abrufen dieser Werte und zum Formatieren dieser Informationen für die Anzeige auf der Website bleiben jedoch 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.

Die Benutzeroberfläche für die Erweiterung dient somit zum Empfangen der Informationen, die von den Benutzern eingegeben werden. Diese Informationen sind zum Verarbeiten der variablen Werte eines wiederholten Arbeitsschritts erforderlich, der mithilfe der Erweiterung ausgeführt wird. 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 zweispaltige Tabelle mit Textbeschreibungen und Formulareingabefeldern enthält.

Beim Entwurf einer Erweiterung sollten Sie ermitteln, welche Variablen erforderlich sind und welche Formularelemente für diese am besten eingesetzt werden.

Beim Entwerfen der Benutzeroberfläche für eine Erweiterung sollten Sie die folgenden grundlegenden 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 der Benutzeroberfläche rechtsbündig an, die Textfelder auf der rechten Seite hingegen linksbündig. Dadurch kann der Benutzer den Anfang der einzelnen Textfelder leichter erkennen. Hinter dem Textfeld kann noch etwas Text folgen, z. B. eine Erklärung oder eine Maßeinheit.
- Richten Sie Bezeichnungen von Kontrollkästchen und Optionsschaltern auf der rechten Seite linksbündig aus.
- Für gut dokumentierten Code sollten Sie den Textfeldern logische Namen zuweisen. Wenn Sie die Benutzeroberfläche Ihrer Erweiterung mit Dreamweaver 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 die folgenden Aufgaben für die Benutzeroberfläche wie vorgesehen ausgeführt werden:

- Abrufen der Werte aus den Textfeldern
- Festlegen von Standardwerten 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 Formularsteuerelemente mit mehr umgebendem Freiraum dargestellt als in Microsoft Internet Explorer und Netscape Navigator. Die Formularsteuerelemente in den Benutzeroberflächen von Erweiterungen werden 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 besser mit der Browseranzeige überein. Um die verbesserte Darstellung zu nutzen, verwenden Sie in Ihren Erweiterungsdateien wie folgt eine der drei neuen DOCTYPE-Anweisungen:

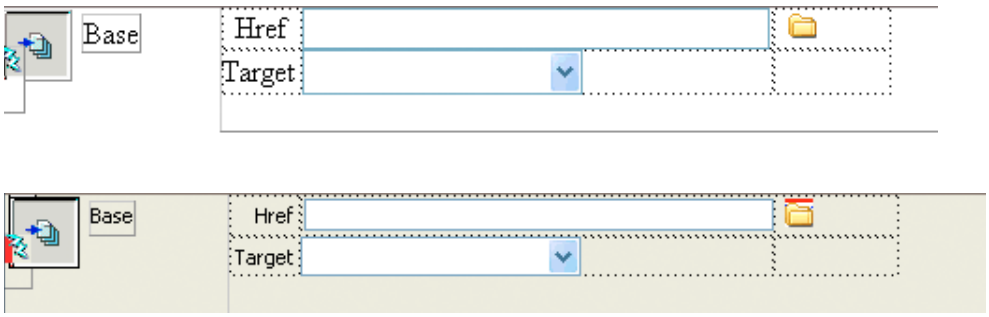
```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//dialog">
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//floater">
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//pi">
```

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

Mithilfe der neuen DOCTYPE-Anweisungen können Sie Ihre Erweiterungen in der Dreamweaver-Entwurfsansicht anzeigen. Sie sehen sie genau so, wie sie den Benutzern angezeigt werden.

Sie finden drei Beispiele, in denen die verbesserte Dialogfelddarstellung verwendet wird, 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 für verbesserte Darstellung der Formularsteuerelemente und mit der `DOCTYPE`-Anweisung aufgeführt.



Benutzerdefinierte Steuerelemente für Benutzeroberflächen in Erweiterungen

Neben den HTML-Standardformularelementen unterstützt Dreamweaver benutzerdefinierte Steuerelemente, um das Erstellen flexibler, professionell wirkender Benutzeroberflächen zu erleichtern.

Bearbeitbare Auswahllisten

Bearbeitbare Auswahllisten (auch als Kombinationsfelder bezeichnet) ermöglichen Ihnen, die Funktionalität von Auswahllisten mit der von Textfeldern zu kombinieren.

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

Im folgenden Beispiel sind 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 das editable-Attribut vorhanden ist und welchen Wert es hat. Wenn kein Wert vorhanden ist, gibt die Auswahlliste den Standardwert `false` zurück. Dies gibt an, dass die Auswahlliste nicht bearbeitbar ist.

Wie herkömmliche nicht bearbeitbare Auswahllisten verfügen auch bearbeitbare Auswahllisten über eine `selectedIndex`-Eigenschaft (siehe „Objekte, Eigenschaften und Methoden des Dreamweaver-DOM“ auf Seite 103). Diese Eigenschaft gibt den Wert -1 zurück, wenn das Textfeld ausgewählt ist.

Um den Wert eines aktiven bearbeitbaren Textfelds in eine Erweiterung zu übernehmen, muss der Wert der `editText`-Eigenschaft gelesen werden. Die `editText`-Eigenschaft gibt den vom Benutzer im bearbeitbaren Textfeld eingegebenen String oder den Wert des `editText`-Attributs zurück. Wenn kein Text eingegeben und kein Wert für die Eigenschaft `editText` angegeben wurde, wird ein leerer String zurückgegeben.

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

Attributname	Beschreibung	Akzeptierte Werte
`editable`	Deklariert, dass das Popupmenü über einen bearbeitbaren Textbereich verfügt.	Ein boolescher Wert (`true` oder `false`).
`editText`	Enthält den 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:

Erstellen des Beispiels

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“.

Testen des Beispiels

1 Starten Sie Dreamweaver neu.

2 Wählen Sie „Befehle“ > „EditableSelectTest“ aus.

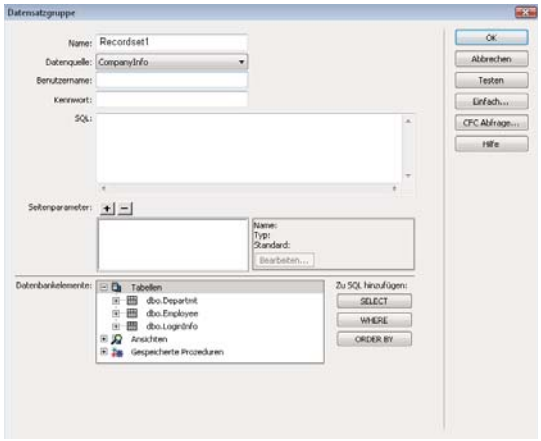
Wenn Sie einen Wert in der Liste auswählen, wird eine Meldung mit dem Index des Werts und dem Text angezeigt. Wenn Sie einen Wert eingeben, wird in einer Meldung angezeigt, dass keine Auswahl getroffen wurde.

Datenbank-Steuerelemente

Datenbank-Steuerelemente ermöglichen die Anzeige von Datenhierarchien und Datenfeldern.

In Dreamweaver können Sie das HTML-Tag `select` erweitern, um Datenbankstruktur-Steuerelemente zu erstellen. Sie können auch ein Steuerelement für variable Tabellen hinzufügen. Das Datenbankstruktur-Steuerelement zeigt Datenbankschemas an, das Steuerelement für variable Tabellen hingegen tabellarische Informationen.

In der folgenden Abbildung sehen Sie ein erweitertes Dialogfeld „Datensatzgruppe“ mit einem Datenbankstruktur-Steuerelement und einem Steuerelement für variable Tabellen:



Hinzufügen von Datenbankstruktur-Steuerelementen

Das Datenbankstruktur-Steuerelement hat folgende Attribute:

Attributname	Beschreibung
`name`	Name des Datenbankstruktur-Steuerelements
`control.style`	Breite und Höhe in Pixel
`type`	Steuerelementtyp
`connection`	Name der im Connection Manager definierten Datenbankverbindung. Wenn das Attribut leer ist, ist auch das Steuerelement leer.
`noexpandbuttons`	Wenn dieses Attribut angegeben ist, enthält das Struktursteuerelement nicht die Plus- (+) und Minuszeichen (–) bzw. auf dem Macintosh die entsprechenden Pfeile als Symbole für das Erweitern oder Reduzieren. 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.

Alle option-Tags innerhalb des Tags `select` werden ignoriert.

Um einem Dialogfeld ein Datenbankstruktur-Steuerelement hinzuzufügen, können Sie den folgenden Beispielcode mit den entsprechenden Ersetzungen für in Anführungszeichen gesetzte Variablen verwenden:

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

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

Hinzufügen von Steuerelementen für variable Tabellen

Das Steuerelement für variable Tabellen hat folgende Attribute:

Attributname	Beschreibung
`name`	Name des Steuerelements für variable Tabellen
`style`	Breite des Steuerelements in Pixel
`type`	Steuerelementtyp
`columns`	Jeder Spalte muss ein Name zugewiesen sein. Die Namen der Spalten müssen durch Kommas getrennt sein.
`columnWidth`	Breite der einzelnen Spalten, getrennt durch Kommas. Wenn die Breite nicht angegeben wird, haben alle Spalten die gleiche Breite.

Im folgenden Beispiel wird einem Dialogfeld ein einfaches Steuerelement für variable Tabellen 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 Steuerelement für variable Tabellen erstellt, das 500 Pixel breit ist und fünf Spalten unterschiedlicher Breite hat:

```
<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>
```

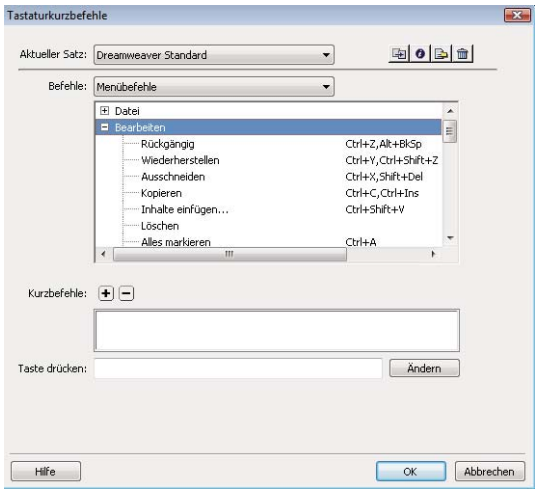
In diesem Beispiel werden zwei leere Spalten mit einer Breite von jeweils 182 Pixel erstellt. (Die angegebenen Spalten ergeben zusammen 136 Pixel. Die Gesamtbreite des Steuerelements für variable Tabellen beträgt 500 Pixel. Der verbleibende Raum nach dem Positionieren der ersten drei Spalten beträgt 364 Pixel. Es bleiben zwei Spalten übrig: 364 geteilt durch 2 ergibt 182.)

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

Struktursteuerelemente

Struktursteuerelemente strukturieren Informationen als erweiterbare und reduzierbare Knoten.

Struktursteuerelemente zeigen Daten als Hierarchie an und ermöglichen dem Benutzer, Knoten in der Struktur zu erweitern und zu reduzieren. Mithilfe des `MM:TREECONTROL`-Tags können Sie Struktursteuerelemente für alle Arten von Informationen erstellen. Im Gegensatz zu dem im Abschnitt „Hinzufügen von Datenbankstruktur-Steuerelementen“ auf Seite 93 beschriebenen Datenbankstruktur-Steuerelement ist keine Verknüpfung mit einer Datenbank erforderlich. In Dreamweaver verwendet der Tastaturkurzbefehl-Editor das Struktursteuerelement wie in der folgenden Abbildung dargestellt:



Erstellen von Struktursteuerelementen

Das `MM:TREECONTROL`-Tag erstellt ein Struktursteuerelement und kann mithilfe zusätzlicher, in der folgenden Aufstellung beschriebenen Tags Strukturen hinzufügen:

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

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

Attributname	Beschreibung
`name`	Name des Struktursteuerelements
`size`	Optional. Anzahl der im Steuerelement angezeigten Zeilen. Der Standardwert ist 5 Zeilen.
`theControl`	Optional. Wenn die Anzahl der Knoten im `theControl`-Attribut größer als der Wert des size-Attributs ist, werden Bildlaufleisten angezeigt.
`multiple`	Optional. Ermöglicht die Mehrfachauswahl. Die Standardeinstellung ist die Einfachauswahl.
`style`	Optional. Stildefinition für Höhe und Breite des Struktursteuerelements. Wenn dieses Attribut angegeben ist, haben die Werte Vorrang vor dem `size`-Attribut.
`noheaders`	Optional. Gibt an, dass die Kopfzeilen der Spalten nicht angezeigt werden.

`MM:TREECOLUMN`-Tags 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 (Prozentangaben werden nicht unterstützt). Der Standardwert ist 100.
`align`	Optional. Gibt an, ob der Text in der Spalte linksbündig, rechtsbündig oder zentriert ausgerichtet werden soll. Die Standardeinstellung ist linksbündig.
`state`	Gibt an, ob die Spalte ein- oder ausgeblendet ist.

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

```
<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
`value`	Enthält den Inhalt des angegebenen Knotens. Wenn mehrere Spalten enthalten sind, wird dieser String durch senkrechte Striche unterteilt. Platzieren Sie einen einzelnen Leerschritt vor dem senkrechten Strich (|), wenn Sie eine leere Spalte angeben möchten.
`state`	Legt mithilfe der Strings `"expanded"` oder `"collapsed"` fest, dass der Knoten erweitert oder reduziert ist.
`selected`	Wenn die Struktur ein MULTIPLE-Attribut 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 z. B. 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"0state="expanded">
<mm:treenode value="node2" state="expanded"></mm:treenode>
<mm:treenode value="node4" state="expanded"></mm:treenode>
</mm:treenode>

</mm:treecontrol>
```

Bearbeiten von Inhalten eines Struktursteuerelements

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 unter „Dokumentobjektmodell von Dreamweaver“ auf Seite 102.

Hinzufügen eines Knotens Um einem vorhandenen Struktursteuerelement im Programmcode einen Knoten hinzuzufügen, legen Sie die Eigenschaft `innerHTML` des `MM:TREECONTROL`-Tags 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;
//add a top-level node to the bottom of the tree
tree.innerHTML = tree.innerHTML + '<mm:treenode name="node3" value="node3">';
```

Hinzufügen eines untergeordneten Knotens Um dem derzeit ausgewählten Knoten einen untergeordneten Knoten hinzuzufügen, legen Sie einen Wert für die Eigenschaft `innerHTML` des ausgewählten Knotens fest.

Im folgenden Beispiel wird dem ausgewählten Knoten ein untergeordneter Knoten hinzugefügt:

```
var tree = document.myTreeControl;
var selNode = tree.selectedNodes[0];
//deselect the node, so we can select the new one
selnode.removeAttribute("selected");
//add the new node to the top of the selected node's children
selNode.innerHTML = '<mm:treenode name="item10" value="New item11" expanded selected>' + ¬
selNode.innerHTML;
```

Löschen eines Knotens Um den 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 = "";
```

Hinzufügen von Steuerelementen zur Farbauswahl

Mit Steuerelementen zur Farbauswahl können Sie Ihren Erweiterungen Benutzeroberflächen zur Farbauswahl hinzufügen.

Neben den Standardeingabetypen wie „text“, „check box“ und „button“ unterstützt Dreamweaver in Erweiterungen den zusätzlichen Eingabetyp `mmcolorbutton`.

Durch Angeben von `<input type="mmcolorbutton">` im Code wird in der Benutzeroberfläche eine Farbauswahl angezeigt. Sie können die Standardfarbe für die Farbauswahl festlegen, indem Sie im input-Tag ein value-Attribut angeben. Wenn kein Wert angegeben ist, wird die Farbauswahl standardmäßig grau angezeigt und die value-Eigenschaft des Eingabeobjekts gibt einen leeren String zurück.

Im folgenden Beispiel ist ein gültiges `mmcolorbutton`-Tag dargestellt:

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

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

Synchronisieren Sie ein Textfeld mit der Farbauswahl. Im folgenden Beispiel wird ein Textfeld erstellt, das die Farbe des Textfelds mit der Farbe der Farbauswahl synchronisiert:

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

In diesem Beispiel ändert der Benutzer den Wert des Textfelds und wechselt mit dem Mauszeiger oder der Tabulatortaste an eine andere Stelle. Daraufhin wird die Farbauswahl aktualisiert und zeigt die im Textfeld angegebene Farbe an. Immer wenn der Benutzer mit der Farbauswahl eine neue Farbe auswählt, wird das Textfeld aktualisiert und zeigt den Hexadezimalwert dieser Farbe an.

Hinzufügen von Flash-Inhalten zu Dreamweaver

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 andere Flash-Inhalte verwenden.

Im Grunde nutzen Sie die Möglichkeit, Dialogfelder für Dreamweaver-Objekte und -Befehle anzuzeigen (weitere Informationen zum Erstellen von Objekten finden Sie unter „Objekte der Einfügeleiste“ auf Seite 112 und weitere Informationen zu Befehlen unter „Befehle“ auf Seite 141). Dazu verwenden Sie das `form`-Tag mit dem `object`-Tag, um Flash-Inhalte in ein Dreamweaver-Dialogfeld einzubetten.

Beispiel für ein einfaches Flash-Dialogfeld

In diesem Beispiel erstellen Sie mithilfe von Dreamweaver einen Befehl. Mit dem erstellten Befehl wird eine SWF-Datei mit dem Namen „myFlash.swf“ angezeigt, wenn der Benutzer auf den entsprechenden Befehl im Menü „Befehle“ klickt. Wenn Sie vor dem Testen dieses Beispiels weitere Informationen zum Erstellen von Befehlen benötigen, finden Sie diese unter „Befehle“ auf Seite 141.

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

Öffnen Sie in Dreamweaver eine neue einfache HTML-Datei (dies ist Ihre Befehlsdefinitionsdatei). Geben Sie zwischen dem öffnenden und schließenden `title`-Tag die Zeichenfolge **My Flash Movie** ein, sodass der Anfang der Seite wie folgt aussieht:

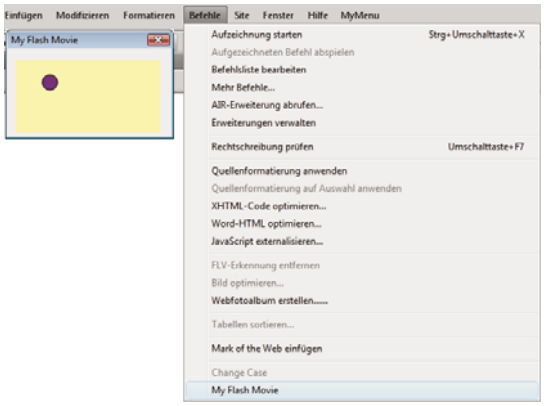
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>My Flash Movie</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
```

Speichern Sie nun die Datei unter dem Namen „My Flash Movie.htm“ im Anwendungsordner „Configuration/Commands“. Schließen Sie die Datei jedoch noch nicht. Speichern Sie anschließend die Datei, um die SWF-Datei mit einer relativen Pfadangabe einbetten zu können. Andernfalls verwendet Dreamweaver einen absoluten Pfad.

Fügen Sie im HTML-Dokument zwischen dem öffnenden und schließenden `body`-Tag ein öffnendes und schließendes `form`-Tag hinzu. Fügen Sie anschließend innerhalb der `form`-Tags mithilfe der Option „Einfügen“ > „Medien“ > „Flash“ Ihre SWF-Datei in die Befehlsdefinitionsdatei ein. Wählen Sie bei entsprechender Aufforderung im Ordner „Commands“ die SWF-Datei aus und klicken Sie auf „OK“. Ihre Befehlsdefinitionsdatei sieht nun wie das folgende Beispiel aus (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 id="FlashID" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="700"
height="150">
    <param name="movie" value="myFlash.swf">
    <!--[if !IE]>-->
    <object type="application/x-shockwave-flash" data="myFlash.swf" width="700"
height="150">
    <!--<![endif]-->
    <param name="quality" value="high"/>
    <param name="wmode" value="opaque" />
    <param name="swfversion" value="8.0.35.0" />
    <!-- This param tag prompts users with Flash Player 6.0 r65 and higher to download the
latest version of Flash Player. Delete it if you don't want users to see the prompt. -->
    <param name="expressinstall"
value="../../../../../ColdFusion8/wwwroot/lr/Scripts/expressInstall.swf" />
    <!-- The browser displays the following alternative content for users with Flash Player
6.0 and older. -->
    <div>
    <h4>Content on this page requires a newer version of Adobe Flash Player.</h4>
    <p><a href="http://www.adobe.com/go/getflashplayer"><img
src="http://www.adobe.com/images/shared/download_buttons/get_flash_player.gif" alt="Get
Adobe Flash player" width="112" height="33" /></a></p>
    </div>
    <!--[if !IE]>-->
</object>
<!--<![endif]-->
</object>
</form>
</body>
</html>
```

Speichern Sie die Datei erneut. Beenden Sie anschließend Dreamweaver und starten Sie die Anwendung neu. Wählen Sie „Befehl“ > „My Flash Movie“ aus. Ihre SWF-Datei wird im Dreamweaver-Dialogfeld angezeigt (siehe folgende Abbildung).



In diesem Beispiel ist eine einfache Implementierung der Dreamweaver-Unterstützung für SWF-Dateien dargestellt. Wenn Sie mit dem Erstellen von Objekten und Befehlen sowie mit komplexeren Formularen vertraut sind, können Sie SWF-Dateien in Dreamweaver-Erweiterungen einfügen, um die Anwendung für Benutzer dynamischer zu gestalten. Weitere Informationen finden Sie unter „Befehle“ auf Seite 141, mit Angaben zum Erstellen einer `commandButtons()`-Funktion zum Hinzufügen von Schaltflächen zu einem Dialogfeld, in dem die SWF-Dateien angezeigt werden.

Photoshop-Integration und Smart Objekte

Adobe® Dreamweaver CS5® importiert und verarbeitet Photoshop-Dateien als Smart Objekte. Jede mit Photoshop am Originalbild vorgenommene Veränderung wird sofort in Dreamweaver angezeigt. Weitere Informationen zu APIs für die Photoshop-Integration in Dreamweaver finden Sie unter „Photoshop-Integration“ im *Dreamweaver API-Referenzhandbuch*.

Beispiel für ein Smart Objekt

In diesem Beispiel erstellen Sie mithilfe von Dreamweaver einen Befehl, mit dem eine Photoshop-Datei (PSD-Datei) aktualisiert wird, wenn der Benutzer auf den entsprechenden Befehl im Menü „Befehle“ klickt. Um die Funktionalität dieses Befehls zu gewährleisten, müssen Sie sicherstellen, dass sich auf der HTML-Seite ein Smart Objekt eines Webbilds befindet. Wenn Sie vor dem Testen dieses Beispiels weitere Informationen zum Erstellen von Befehlen benötigen, finden Sie diese unter „Befehle“ auf Seite 141.

Öffnen Sie in Dreamweaver eine neue einfache HTML-Datei (dies ist Ihre Befehlsdefinitionsdatei). Die Befehlsdefinitionsdatei sieht wie folgt aus:

```
<html xmlns:MMStr ing="http://www.adobe.com /schemes/data/   string/">
<head>
    <title> Smart Objects API</ title >
    <SC RIPT SRC="../Shared/Common/Scripts /dwscripts.js"></SC RIPT>
    <SCRIPT LANGUAGE="Javascript">
function invokeSmartObjectJavaScriptAPICall() {
    var selection = dw.getSelection();
    if (!selection) {
    alert("Err: No selection!");
    return;
    }
    var node = dw.offsetsToNode(selection[0], selection[1]);
    if (!node) {
                alert("Err: No Node!");
                return;
                }
    var imageSrc = node.getAttribute("src");
    if (!imageSrc) {
                    alert("Err: No src attribute!");
                    return;
                    }
    var fullPath = getFullPath(imageSrc);
    if (!fullPath) {
                    alert("Err: No path!");
                    return;
                    }
    //alert(fullPath);
    alert("updateSmartObjectFromOriginal");
    dw.updateSmartObjectFromOriginal(fullPath);
}
    <script>
</head>
<body onload="invokeSmartObjectJavaScriptAPICall()">
</body>
</html>
```

Speichern Sie die Datei unter dem Namen „smartobjects.htm“ im Dreamweaver-Ordner „Configuration/Commands“. Wählen Sie bei entsprechender Aufforderung die SWF-Datei im Ordner „Commands“ aus und klicken Sie auf „OK“.

Kapitel 6: Dokumentobjektmodell von Dreamweaver

Das Dokumentobjektmodell (DOM) in Adobe Dreamweaver ist für das Erstellen von Erweiterungen von entscheidender Bedeutung.

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.

Allgemeines zum Dreamweaver-DOM

Die Struktur eines HTML-Dokuments entspricht der eines Baums. Der Stamm ist das `html`-Tag und die beiden größten Äste sind die `head`- und `body`-Tags. Zu den Zweigen des `head`-Tags gehören die Tags `title`, `style`, `script`, `isindex`, `base`, `meta` und `link`. Zweige des `body`-Tags sind beispielsweise:

- Überschriften (`h1`, `h2` usw.)
- Elemente auf Blockebene (`p`, `div`, `form` usw.)
- Inline-Elemente (`br`, `img` usw.)
- Andere Elementtypen

Zu den Blättern an diesen Zweigen zählen Attribute wie `width`, `height`, `alt` und andere.

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 ein HTML-Element von JavaScript als Knoten verwendet werden. Über diese Struktur können Sie auf das Dokument oder jedes beliebige Element im Dokument zugreifen.

In JavaScript können Sie über den Namen oder Index auf jedes Objekt im Dokument verweisen. Angenommen, eine Schaltfläche „Senden“ mit dem Namen oder der ID „`myButton`“ ist das zweite Element im ersten Formular des Dokuments. In diesem Fall sind beide der folgenden Verweise auf diese Schaltfläche gültig:

- 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 Optionen) werden zu einem Array zusammengefasst. Sie können auf ein bestimmtes Objekt im Array zugreifen, indem Sie den Index (von Null ausgehend) erhöhen. Beispielsweise wird auf die erste Option mit dem Namen „`myRadioGroup`“ in einem Formular mit dem Namen „`myForm`“ als `document.myForm.myRadioGroup[0]` verwiesen.

Unterscheiden zwischen dem DOM des Benutzerdokuments und dem DOM der Erweiterung

Es ist wichtig, zwischen dem DOM des Benutzerdokuments und dem DOM der Erweiterung zu unterscheiden. Die Informationen in diesem Abschnitt 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 das erste Bild im aktiven Dokument zu referenzieren, können Sie `dw.getDocumentDOM().images[0]` verwenden. Sie können das Dokumentobjekt auch in einer Variable speichern und diese Variable für zukünftige Verweise verwenden, wie im folgenden Beispiel dargestellt:

```
var dom = dw.getDocumentDOM();  //get the dom of the current document
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*.

Dreamweaver-DOM

Das Dreamweaver-DOM enthält eine Teilmenge von Objekten, Eigenschaften und Methoden der DOM-Level-1-Spezifikation des World Wide Web Consortium (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. Einige 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•	Keine	Keine
document	forms• (Array von Formularobjekten) images• (Array von Bildobjekten) layers • (Array mit LAYER- und ILAYER-Elementen sowie mit absolut positionierten Elementen) child-Objekte nach Name• nodeType• parentNode• childNodes• previousSibling• nextSibling• documentElement• body• URL• parentWindow•	getElementsByTagName() getElementsByAttributeName() getElementById() hasChildNodes()	onLoad
Alle Tags/Elemente	nodeType• parentNode • childNodes• tagName• previousSibling• nextSibling• attributes nach Name innerHTML outerHTML	getAttribute() setAttribute() removeAttribute() getElementsByTagName() getElementsByAttributeName() hasChildNodes()	

Objekt	Eigenschaften	Methoden	Ereignisse
`form`	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: `tags:elements` • (Array mit 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.	Keine

`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 von option-Objekten)`
	`selectedIndex`
		Zusätzlich zu den für alle Tags verfügbaren Methoden:
		`blur()` (nur Windows)
		`focus()` (nur Windows)
			`onBlur` (nur Windows) `onChange`
			`onFocus` (nur Windows)

Objekt	Eigenschaften	Methoden	Ereignisse
option	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: text	Nur die für alle Tags verfügbaren Methoden.	Keine
mmcolorbutton	Zusätzlich zu den für alle Tags verfügbaren Eigenschaften: name value	Keine	onChange
array boolean date function number object string regexp	Entspricht Netscape Navigator 4.0	Entspricht Netscape Navigator 4.0	Keine
text	nodeType• parentNode• childNodes• previousSibling• nextSibling• data	hasChildNodes()	Keine
comment	nodeType• parentNode• childNodes• previousSibling• nextSibling• data	hasChildNodes()	Keine
NodeList	length•	item()	Keine
NamedNodeMap	length•	item()	Keine

Eigenschaften und Methoden von document-Objekten

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 jedoch nicht zu DOM Level 1 oder 2.)
`childNodes` •	Eine Knotenliste (`NodeList`), die alle unmittelbar untergeordneten Elemente des Objekts `document` enthält. In der Regel verfügt ein Dokument über ein einziges untergeordnetes Element, das `HTML`-Objekt.
`previousSibling` •	null
`nextSibling` •	null
`documentElement` •	Das JavaScript-Objekt, das dem `HTML`-Tag entspricht. Mit dieser Eigenschaft wird der Wert von `document.childNodes` direkt abgefragt und das `HTML`-Tag aus der `NodeList` extrahiert.
`body` •	Das JavaScript-Objekt, das dem `body`-Tag entspricht. Mit dieser Eigenschaft wird der Wert von `document.documentElement.childNodes` direkt abgefragt und das `body`-Tag aus der `NodeList` extrahiert. Bei Frameset-Dokumenten gibt diese Eigenschaft den Knoten für das äußerste Frameset zurück.
`URL` •	Die Pfadangabe für das Dokument im URL-Format ***`file`***`://` oder ein leerer String, wenn das Dokument nicht gespeichert wurde.
`getElementsByTagName(tagName)`	Eine `NodeList` zum schrittweisen Durchlaufen von Tags des Typs ***`tagName`*** (z. B. `img`, `div` usw.). Wenn der Wert des Arguments ***`tagName`*** „LAYER“ ist, gibt die Funktion alle Tags des Typs `LAYER` und `ILAYER` sowie alle absolut positionierten Elemente zurück. Wenn der Wert des Arguments ***`tagName`*** „INPUT“ ist, gibt die Funktion alle Formularelemente zurück. (Wenn ein name-Attribut für ein oder mehrere ***`tagName`***-Objekte angegeben ist, muss es – wie in der HTML 4.01-Spezifikation vorgegeben – mit einem Buchstaben beginnen, andernfalls ist die Länge des von dieser Funktion zurückgegebenen Arrays falsch.)
`getElementsById(Id)`	Ruft den Elementknoten mit der angegebenen ***`id`*** ab. Dabei ist ***`id`*** ein String mit der ID des abzurufenden Elements. `var dom = dw.getDocumentDOM(); var contObj = dom.getElementById('content'); alert("The element with the id 'content' is a " + contObj.tagName);`
`getElementsByAttributeName(attrName)`	Eine `NodeList` zum schrittweisen Durchlaufen von Elementen mit einem ***`attrName`***-Attribut (z. B. alle Elemente mit dem Attribut „for“). Gehört nicht zu DOM Level 1 oder 2.
`getElementById(id)`	Das HTML-Element mit der angegebenen ID.
`hasChildNodes()`	`true`

Eigenschaften und Methoden von HTML-Elementen

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

Eigenschaft oder Methode	**Rückgabewert**
`nodeType` •	`Node.ELEMENT_NODE`
`parentNode` •	Das übergeordnete Tag. Wenn es sich dabei um das `HTML`-Tag handelt, wird das document-Objekt zurückgegeben.
`childNodes` •	Eine `NodeList` mit allen unmittelbar untergeordneten Elementen des Tags.
`previousSibling` •	Der Parallelknoten unmittelbar vor diesem Knoten. In einem HTML-Dokument ist der `previousSibling` für das `body`-Element beispielsweise das Element `head`.
`nextSibling` •	Der Parallelknoten unmittelbar nach diesem Knoten. In einem HTML-Dokument ist der `nextSibling` für das `head`-Element beispielsweise das Element `body`. (Alle `script`-, `style`- und `meta`-Tags im head-Bereich sind untergeordnete Knoten des `head`-Elements.)
`tagName` •	Der HTML-Tag-Name für das Element, z. B. `IMG`, `A` oder `DIV`. Dieser Wert wird immer in Großbuchstaben zurückgegeben.
`attrName`	Ein String, der den Wert des angegebenen Tag-Attributs enthält. `tag.`***`attrName`*** kann nicht verwendet werden, wenn das Attribut ***`attrName`*** ein reserviertes Wort der JavaScript-Programmiersprache ist (z. B. `class`). Verwenden Sie in diesem Fall `getAttribute()` und `setAttribute()`.
`innerHTML`	Der Quellcode zwischen dem öffnenden und dem schließenden Tag. Für den Code `<p><b>Hello</b>, World!</p>` gibt `p.innerHTML` beispielsweise Folgendes zurück: `<b>Hello</b>, World!`. Wenn Sie in diese Eigenschaft schreiben, wird die DOM-Struktur sofort aktualisiert, um die neue Dokumentstruktur wiederzugeben. (Diese Eigenschaft ist im Microsoft Internet Explorer 4.0-DOM definiert, gehört jedoch nicht zu DOM Level 1 oder 2.)
`outerHTML`	Der Quellcode für dieses Tag, einschließlich des Tags selbst. Im vorherigen Beispielcode wurde von `p.outerHTML` folgender String zurückgegeben: `<p><b>Hello</b>, World!</p>`. Wenn Sie in diese Eigenschaft schreiben, wird die DOM-Struktur sofort aktualisiert, um die neue Dokumentstruktur wiederzugeben. (Diese Eigenschaft ist im Microsoft Internet Explorer 4.0-DOM definiert, gehört jedoch nicht zu DOM Level 1 oder 2.)
`getAttribute(`***`attrName`***`)`	Der Wert des angegebenen Attributs, sofern ausdrücklich angegeben, andernfalls `null`.
`getTranslatedAttribute(`***`attrName`***`)`	Der übersetzte Wert des angegebenen Attributs oder der gleiche Wert, der von `getAttribute()` 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.)
`setAttribute(`***`attrName`***`, `***`attrValue`***`)`	Gibt keinen Wert zurück. Setzt das betreffende Attribut auf das angegebene Wertbeispiel, d. h. `img.setAttribute("src", "image/roses.gif")`.
`removeAttribute(`***`attrName`***`)`	Gibt keinen Wert zurück. Entfernt das angegebene Attribut und seinen Wert aus dem HTML-Code für dieses Tag.

Eigenschaft oder Methode	Rückgabewert
`getElementsByTagName(`***`tagName`***`)`	Eine `NodeList` zum schrittweisen Durchlaufen von untergeordneten Tags des Typs ***`tagName`*** (z. B. `IMG`, `DIV` usw.). Wenn der Wert des Arguments ***`tagName`*** „LAYER" ist, gibt die Funktion alle Tags des Typs `LAYER` und `ILAYER` sowie alle absolut positionierten Elemente zurück. Wenn der Wert des Arguments ***`tagName`*** „INPUT" ist, gibt die Funktion alle Formularelemente zurück. (Wenn ein name-Attribut für ein oder mehrere `tagName`-Objekte angegeben ist, muss es – wie in der HTML 4.01-Spezifikation vorgegeben – mit einem Buchstaben beginnen, andernfalls ist die Länge des von dieser Funktion zurückgegebenen Arrays falsch.)
`getElementsByAttributeName(`***`attrName`***`)`	Eine `NodeList` zum schrittweisen Durchlaufen von Elementen mit einem ***`attrName`***-Attribut (z. B. alle Elemente mit dem Attribut „for"). Gehört nicht zu DOM Level 1 oder 2.
`hasChildNodes()`	Ein boolescher Wert, der angibt, ob das Tag untergeordnete Elemente hat.
`hasTranslatedAttributes()`	Ein boolescher Wert, der angibt, ob das Tag übersetzte Attribute hat. (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 text-Objekten

Jeder zusammenhängende Textblock in einem HTML-Dokument (z. B. der Text innerhalb eines `p`-Tags) wird durch ein JavaScript-Objekt dargestellt. text-Objekte 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 von text-Objekten aufgeführt. Ein Punkt (•) kennzeichnet schreibgeschützte Eigenschaften.

Eigenschaft oder Methode	Rückgabewert
`nodeType` •	`Node.TEXT_NODE`
`parentNode` •	Das übergeordnete Tag.
`child Nodes` •	Ein leeres NodeList-Array mit untergeordneten Knoten.
`previousSibling` •	Der Parallelknoten unmittelbar vor diesem Knoten. Im Code `<p>blah blah</p>` hat das Tag `<p>` z. B. drei untergeordnete Knoten (Textknoten, Elementknoten, Textknoten). Der `previousSibling` des dritten untergeordneten Elements ist das ` `-Tag und der `previousSibling` des ersten untergeordneten Elements ist `null`.
`nextSibling` •	Der Parallelknoten unmittelbar nach diesem Knoten. Beispielsweise ist im Code `<p>blah blah</p>` der `nextSibling` des ersten untergeordneten Elements des `p`-Tags das ` `-Tag und der `nextSibling` des dritten untergeordneten Elements ist `null`.
`data`	Der eigentliche Textstring. Entitäten im Text werden als einzelne Zeichen dargestellt (der Text `Johann &amp; ich` wird z. B. als `Johann & ich` zurückgegeben).
`hasChildNodes()`	`false`

Eigenschaften und Methoden von comment-Objekten

Jeder HTML-Kommentar wird durch ein JavaScript-Objekt dargestellt. In der folgenden Tabelle sind Einzelheiten zu den vom DOM Level 1 übernommenen und in Dreamweaver verwendeten Eigenschaften und Methoden von comment-Objekten 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 Knoten.
`nextSibling` •	Der Parallelknoten unmittelbar nach diesem Knoten.
`data`	Der Textstring zwischen den Kommentarmarkierungen (`<!-- und -->`)
`hasChildNodes()`	`false`

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 häufig innerhalb von APIs und beim Programmieren von Erweiterungen verwendet. Weitere Informationen zu den Methoden der Objekte `dreamweaver` und `site` finden Sie im *Dreamweaver API-Referenzhandbuch*.

Eigenschaften des dreamweaver-Objekts

Das `dreamweaver`-Objekt hat zwei schreibgeschützte Eigenschaften, die im Folgenden beschrieben sind:

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

  Der Wert der Eigenschaft `appVersion` für die schwedische Windows-Version von Dreamweaver lautet beispielsweise `"10.0.XXXX [se] (Win32)"` und der Wert für die englische Macintosh-Version lautet `"10.0.XXXX [en] (MacPPC)"`.

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

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

Um die genaue Version von Dreamweaver zu ermitteln, überprüfen Sie zunächst das Vorhandensein von `appVersion` und dann die Versionsnummer, wie im folgenden Beispiel dargestellt:

```
if (dreamweaver.appVersion && dreamweaver.appVersion.indexOf('3.01') != -1){¬
    // execute code
}
```

Das `dreamweaver`-Objekt hat eine Eigenschaft mit dem Namen `systemScript`, mit der Sie die Sprache des verwendeten Betriebssystems ermitteln können. Verwenden Sie diese Eigenschaft wie folgt, um in Ihrem Erweiterungscode Sonderfälle 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:

Sprache	Wert
Japanisch	`ja`
Koreanisch	`ko`
Traditionelles Chinesisch	`zh_tw`
Vereinfachtes Chinesisch	`zh_cn`

Die Betriebssysteme aller europäischen Sprachen geben `'en'` zurück.

Objekt „site“

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

Kapitel 7: Objekte der Einfügeleiste

Sie können der Einfügeleiste Elemente hinzufügen, um häufig wiederkehrende Aufgaben für Benutzer zu automatisieren. Zudem können Sie Dialogfelder erstellen, in denen 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 Einfügeleiste zusammengefasst. Sie können diese Dateien öffnen, um den Aufbau bereits vorhandener Objekte nachzuvollziehen. Sie können beispielsweise die Datei „Configuration/Objects/Common/Hyperlink.htm“ öffnen, um den Code zu überprüfen, der der Schaltfläche für ein Hypertextverknüpfungsobjekt auf der Einfügeleiste entspricht.

In der folgenden Tabelle sind die Dateien zum Erstellen von Objekten aufgeführt:

Pfad	**Datei**	**Beschreibung**
Configuration/Objects/*Objekttyp*/	*Objektname*.htm	Gibt an, welches Objekt in das Dokument eingefügt werden soll.
Configuration/Objects/*Objekttyp*/	*Objektname*.js	Enthält die auszuführenden Funktionen.
Configuration/Objects/*Objekttyp*/	*Objektname*.gif	Enthält das Bild, das in der Einfügeleiste angezeigt wird.
Configuration/Objects	insertbar.xml	Gibt die in der Einfügeleiste angezeigten Objekte und deren Reihenfolge an.

Funktionsweise von Objektdateien

Mit Objekten können bestimmte Codestrings in ein Benutzerdokument eingefügt werden. Mithilfe von Objekten können Benutzer Inhalte wie Bilder, absolut positionierte Elemente (AP-Elemente) und Tabellen hinzufügen, indem sie auf Symbole oder Optionen im Menü klicken.

Objekte umfassen die folgenden Komponenten:

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

  Der head-Bereich einer Objektdatei enthält JavaScript-Funktionen (oder Verweise auf externe JavaScript-Dateien), die Formulareingaben im body-Bereich verarbeiten und steuern, welche Inhalte dem Dokument eines Benutzers hinzugefügt werden. Der body-Bereich einer Objektdatei kann ein HTML-Formular enthalten, dem Parameter für das Objekt übergeben werden können (z. B. die Anzahl der in eine Tabelle einzufügenden Zeilen und Spalten) und das ein Dialogfeld zum Eingeben von Attributen öffnet.

  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 unter „Dreamweaver anpassen“.*

- Das auf der Einfügeleiste angezeigte Bild im Format 18 x 18 Pixel.
- Erweiterungen der Datei „insertbar.xml“. In der Datei „insertbar.xml“ ist festgelegt, an welcher Stelle der Einfügeleiste das Objekt angezeigt wird.

Ein Benutzer kann ein Objekt auswählen, indem er auf ein Symbol in der Einfügeleiste klickt oder ein Element im Menü „Einfügen“ auswählt. Wenn ein Benutzer ein Objekt auswählt, geschieht Folgendes:

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

   Die Objektdatei wird nach dem Tag `form` durchsucht. Wenn ein Formular vorhanden ist und Sie im Dialogfeld „Voreinstellungen“ die Option „Beim Einfügen von Objekten Dialogfeld anzeigen“ auswählen, ruft Dreamweaver die Funktion `windowDimensions()` auf, sofern diese definiert ist. Dreamweaver ruft die Funktion auf, um die Größe des Dialogfelds festzulegen, in dem das Formular angezeigt wird. Wenn in der Objektdatei kein Formular vorhanden ist, wird kein Dialogfeld angezeigt. Schritt 2 wird in diesem Fall übersprungen.

2 Wenn Dreamweaver in Schritt 1 ein Dialogfeld anzeigt, gibt der Benutzer die Parameter für das Objekt in den Textfeldern des Dialogfelds ein (z. B. die Anzahl der Zeilen und Spalten in einer Tabelle) und klickt auf „OK“.

3 Dreamweaver ruft die Funktion `objectTag()` auf und fügt den entsprechenden 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.

Definitionsdatei für die Einfügeleiste

In der Datei „Configuration/Objects/insertbar.xml“ werden die Eigenschaften der Einfügeleiste 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 Einfügeleiste horizontal über dem Dokument angezeigt. Anschließend werden Anzeige und Position der Einfügeleiste in der Registrierung gespeichert.

Tag-Hierarchie der Datei „insertbar.xml“

Im folgenden Beispiel sind das Format und die Hierarchie verschachtelter Tags in der Datei „insertbar.xml“ angegeben:

```
<?xml version="1.0" ?>
<!DOCTYPE insertbarset SYSTEM "-//Adobe//DWExtension insertbar 10.0">
<insertbar xmlns:MMString="http://www.adobe.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:** Für die Tags `insertbar` und `category` werden die schließenden Tags `</insertbar>` und `</category>` verwendet, um das Ende des Inhalts anzugeben. Bei den Tags „button“, „checkbutton“ und „separator“ wird das Ende der Attribute und des Inhalts mit einem Schrägstrich (/) vor der schließenden Klammer angegeben.*

Tags zur Definition der Einfügeleiste

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

<insertbar>

Beschreibung

Dieses Tag kennzeichnet den Anfang des Inhalts der Definitionsdatei für die Einfügeleiste. Das schließende Tag `/insertbar` 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 (z. B. „Allgemein“, „Formulare“ oder „HTML“). Das Ende des Inhalts der Kategorie wird mit dem schließenden Tag`</category>` angegeben.

Hinweis: *Die Einfügeleiste ist standardmäßig in Verwendungskategorien untergliedert (z. B. „Allgemein“, „Formulare“ oder „HTML“). In früheren Versionen von Dreamweaver war diese Leiste auf ähnliche Weise in Registerkarten unterteilt. Benutzer können selbst festlegen, wie die Objekte der Einfügeleiste angeordnet werden sollen (in Kategorien oder Registerkarten). Wenn sich der Benutzer für Registerkarten entscheidet, werden die einzelnen Registerkarten mithilfe von 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 Einfügeleiste, über die 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"
    />
```

<checkbutton />

Beschreibung

Bei einem Kontrollkästchen handelt es sich um eine Schaltfläche, die aktiviert oder deaktiviert sein kann. Wenn ein Benutzer auf ein Kontrollkästchen klickt, wird es gedrückt und hervorgehoben angezeigt. Wenn es deaktiviert ist, wird es flach dargestellt. In Dreamweaver sind folgende Statuswerte für Kontrollkästchen festgelegt: Mauszeiger über dem Kontrollkästchen, gedrücktes Kontrollkästchen, Mauszeiger über dem gedrückten Kontrollkästchen und deaktiviertes gedrücktes Kontrollkästchen. Mit dem Befehl muss sichergestellt werden, dass sich der Status des Kontrollkästchens ändert, wenn darauf geklickt wird.

Attribute

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

Beispiel

```
<checkbutton 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 Einfügeleiste 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 entsprechende Element sein.

Beispiel

```
id="DW_Anchor"
```

image="image_path"

Beschreibung

Dieses Attribut gibt relativ zum Dreamweaver-Ordner „Configuration“ den Pfad zur Datei des Symbols an, das in der Einfügeleiste angezeigt wird. Das Symbol kann jedes Format haben, das in Dreamweaver dargestellt werden 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. Wenn das Attribut nicht angegeben wird, gilt der 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. Wenn Sie `showIf` nicht angeben, wird die Schaltfläche immer angezeigt. Mögliche Enabler sind `_SERVERMODEL_ASP`, `_SERVERMODEL_ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von Adobe 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

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 ist, wenn der Wert von *DW_enabler* auf `true` gesetzt ist. Wenn Sie die Funktion `enabled` nicht angeben, ist das Element standardmäßig immer aktiviert. Mögliche Enabler sind `_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
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 Attribut `checked` ist erforderlich, wenn Sie das Tag `checkbutton` verwenden.

Das Element ist aktiviert, wenn *DW_enabler* auf `true` gesetzt ist. Mögliche Enabler sind `_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_function"

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

Beispiel

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

file="file_path"

Beschreibung
Das Attribut `file` gibt den Pfad zu einer Objektdatei relativ zum Dreamweaver-Ordner „Configuration“ an. 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 angewiesen, 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 die Code- und Entwurfsansicht auf, legt den Fokus auf den Code und ruft den Tag-Editor auf.

Beispiel

```
tag = "form"
```

name="tooltip_text"

Beschreibung

Das Attribut `name` gibt die QuickInfo an, die angezeigt wird, wenn sich der Mauszeiger über dem Objekt befindet. Wenn Sie eine Objektdatei angeben, jedoch das Attribut `name` nicht festlegen, verwendet Dreamweaver den Namen der Objektdatei für die QuickInfo.

Hinweis: *Wenn das name-Attribut nicht angegeben ist, kann das Objekt nicht in der Favoritenkategorie der Einfügeleiste gruppiert werden.*

Einige Objekte der Einfügeleiste verwenden eine Variante des name-Attributs mit dem Präfix `MMString`. MMString kennzeichnet einen lokalisierten String. Eine Erläuterung dieser Werte finden Sie unter „Lokalisieren von Erweiterungen“ auf Seite 86.

Beispiel

```
name = "cfoutput"
```

Ändern der Einfügeleiste

Sie können Objekte aus einer Kategorie in eine andere verschieben, Kategorien umbenennen und Objekte vollständig aus dem Bedienfeld entfernen. Damit die Änderungen in der Einfügeleiste angezeigt werden, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erneutes Laden von Erweiterungen“ auf Seite 84.

Kopieren oder Verschieben eines Objekts von einer Kategorie der Einfügeleiste 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 das zu verschiebende oder zu kopierende Objekt angibt. 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"` hat.
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 Wechseln Sie zu der Stelle in der Kategorie, an der 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.

Entfernen eines Objekts aus der Einfügeleiste

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 das zu entfernende Objekt angibt.

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 dem aktuellen Ordner in einen Ordner, der nicht in der Datei „insertbar.xml“ aufgeführt ist. Sie können z. B. unter „Configuration/Objects“ einen neuen Ordner mit dem Namen „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 jedoch immer empfehlenswert, Sicherungskopien dieser Dateien zu behalten, falls Sie das Objekt später wiederherstellen müssen.)

7 Laden Sie die Erweiterungen neu.

Ändern der Reihenfolge der Kategorien in der Einfügeleiste

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.

Erstellen einer neuen 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 Gruppe von Objekten an, die in der Kategorie angezeigt werden sollen.

4 Informationen zur Syntax der Tags in der Datei „insertbar.xml“ finden Sie unter „Tags zur Definition der Einfügeleiste“ auf Seite 114.

5 Speichern Sie die Datei „insertbar.xml“.

6 Laden Sie die Erweiterungen neu.

Hinzufügen eines neuen Objekts zur Einfügeleiste

Sie können der Einfügeleiste Objekte hinzufügen. Damit die Änderungen in der Einfügeleiste angezeigt werden, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erneutes Laden von Erweiterungen“ auf Seite 84.

1 Definieren Sie den entsprechenden Codestring für das Dokument des Benutzers mit HTML-Code und optional mit JavaScript-Code.

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. Wenn Sie kein Bild für das Objekt erstellen, wird auf der Einfügeleiste 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 „Definitionsdatei für die Einfügeleiste“ auf Seite 113).

5 Starten Sie Dreamweaver neu oder laden Sie die Erweiterungen neu.

Das neue Objekt wird an der angegebenen Position auf der Einfügeleiste angezeigt.

Hinweis: *Auch wenn Objektdateien in separaten Ordnern gespeichert werden können, dürfen die Dateinamen jeweils nur einmal vorkommen. Die Funktion* `dom.insertObject()` *sucht z. B. nach Dateien im gesamten Ordner „Objects“, ohne nach Unterordnern zu unterscheiden. (Weitere Informationen zur 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. Wenn zwei verschiedene Instanzen der Datei „Button.htm“ vorhanden sind, zeigt* `dom.insertObject()` *zwei Objekte mit dem Namen „Button“ an, ohne dass der Benutzer einen Unterschied erkennt.*

Hinzufügen von Objekten zum Menü „Einfügen“

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 zum Bearbeiten der Datei „menus.xml“ finden Sie unter „Menüs und Menübefehle“ auf Seite 153.

Wenn Sie die Erweiterung an andere Dreamweaver-Benutzer weitergeben möchten, finden Sie unter „Arbeiten mit dem Extension Manager“ auf Seite 87 weitere Informationen zum Komprimieren von Erweiterungen.

Einfaches Beispiel für das Einfügen eines Objekts

In diesem Beispiel wird der Einfügeleiste ein Objekt hinzugefügt, sodass Benutzer durch Klicken auf eine Schaltfläche einen ausgewählten Text durchgestrichen formatieren können. Dieses Objekt ist z. B. nützlich, wenn ein Benutzer ein Dokument redigiert.

Da in diesem Beispiel Text bearbeitet wird, bietet es sich an, einige der Objekte des Popupmenüs „Text“ in der Kategorie „HTML“ der Einfügeleiste als Modelle heranzuziehen. In den Objektdateien für die Formatierungen „Fett“, „Hervorhebung“ und „Überschrift“ finden Sie beispielsweise eine vergleichbare Funktionsweise, bei der ein ausgewählter Text in Tags eingeschlossen ist.

Zum Erstellen des Objekts zum Einfügen der durchgestrichenen Textformatierung führen Sie die folgenden Schritte aus.

Erstellen der HTML-Datei

Der Titel des Objekts wird zwischen dem öffnenden und dem schließenden `title`-Tag angegeben. Legen Sie außerdem JavaScript als Skriptsprache fest.

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie 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“.

Hinzufügen der JavaScript-Funktionen

In diesem Beispiel definieren die JavaScript-Funktionen das Verhalten und den einzufügenden Code des Objekts „Strikethrough“. Sie müssen alle API-Funktionen in den head-Bereich der Datei einfügen. Die vorhandenen Objektdateien, z. B. „Configuration/Objects/Text/Em.htm“, weisen eine ähnliche Abfolge von Funktionen und Kommentaren auf.

Die erste von der Objektdefinitionsdatei verwendete Funktion ist `isDOMRequired()`. Sie gibt an, ob die Entwurfsansicht vor dem Fortsetzen der Programmausführung mit der vorhandenen Codeansicht synchronisiert werden muss. Da das Objekt jedoch zusammen mit zahlreichen anderen Objekten in der Codeansicht verwendet werden kann, ist keine Synchronisierung erforderlich.

Hinzufügen der isDOMRequired()-Funktion

1 Fügen Sie in den head-Bereich der Datei „Strikethrough.htm“ zwischen dem öffnenden und dem schließenden `script`-Tag die folgende Funktion ein:

```
<script language="javascript">
    function isDOMRequired() {
        // Return false, indicating that this object is available in Code view.
        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 131).

Innerhalb der Funktion `objectTag()` können Sie `dw.getFocus()` verwenden, um festzustellen, ob es sich bei der aktuellen Ansicht um die Codeansicht handelt. Wenn der Eingabefokus auf der Codeansicht liegt, sollte die Funktion den ausgewählten Text mit dem entsprechenden Tag (Groß- oder Kleinbuchstaben) umschließen. Wenn der Eingabefokus auf der Entwurfsansicht liegt, kann der ausgewählte Text mithilfe von `dom.applyCharacterMarkup()` formatiert werden. Beachten Sie, dass diese Funktion nur für unterstützte Tags ausgeführt wird (siehe hierzu `dom.applyCharacterMarkup()` im *Dreamweaver API-Referenzhandbuch*). Für andere Tags oder Operationen sind möglicherweise andere API-Funktionen erforderlich. Nach dem Anwenden der Formatierung sollte die Einfügemarke (Cursor) ohne weitere Meldung oder Eingabeaufforderung wieder in das Dokument gesetzt werden. Im folgenden Verfahren ist der neue Aufbau der Funktion `objectTag()` dargestellt.

Hinzufügen der objectTag()-Funktion

1 Fügen Sie im head-Bereich der Datei „Strikethrough.htm“ nach der Funktion `isDOMRequired()` die folgende Funktion ein:

```
function objectTag() {
    // Determine if the user is in Code view.
    var dom = dw.getDocumentDOM();
    if (dw.getFocus() == 'textView' || dw.getFocus(true) == 'html'){
        var upCaseTag = (dw.getPreferenceString("Source Format", "Tags Upper Case", "") ==
        'TRUE');
        // Manually wrap tags around selection.
        if (upCaseTag){
            dom.source.wrapSelection('<S>','</S>');
        }else{
            dom.source.wrapSelection('<s>','</s>');
        }
    // If the user is not in Code view, apply the formatting in Design view.
    }else if (dw.getFocus() == 'document'){
        dom.applyCharacterMarkup("s");
    }
    // Just return--don't do anything else.
    return;
}
```

2 Speichern Sie die Datei unter dem Namen „Strikethrough.htm“ im Ordner „Configuration/Objects/Text“.

Als Alternative zum Einfügen der JavaScript-Funktionen im head-Bereich der HTML-Datei können Sie eine separate JavaScript-Datei erstellen. Diese Vorgehensweise empfiehlt sich bei Objekten, die mehrere Funktionen enthalten oder die auch von anderen Objekten verwendet werden können.

Trennen der HTML-Objektdefinitionsdatei von den unterstützenden JavaScript-Funktionen

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie alle JavaScript-Funktionen in die Datei ein.

3 Löschen Sie die Funktionen in 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“.

Erstellen des Bilds für die Einfügeleiste

1 Erstellen Sie ein GIF-Bild (18 x 18 Pixel), wie in der folgenden Abbildung dargestellt:

2 Speichern Sie die Datei unter dem Namen „Strikethrough.gif“ im Ordner „Configuration/Objects/Text“.

Bearbeiten der Datei „insertbar.xml“

Als Nächstes müssen Sie die Datei „insertbar.xml“ bearbeiten, damit diese beiden Elemente mit der Benutzeroberfläche für die Einfügeleiste verknüpft werden können.

Hinweis: *Vor dem Bearbeiten der Datei „insertbar.xml“ sollten Sie eine Sicherungskopie mit dem Namen „insertbar.xml.bak“ erstellen.*

Der Code in der Datei „insertbar.xml“ gibt alle in der Einfügeleiste vorhandenen Objekte an.

- Jedes `category`-Tag in der XML-Datei erstellt eine Kategorie in der Benutzeroberfläche.
- Jedes `menubutton`-Tag erstellt ein Popupmenü auf der Einfügeleiste.
- Jedes `button`-Tag in der XML-Datei fügt ein Symbol auf der Einfügeleiste ein und verknüpft es mit der entsprechenden HTML-Datei oder HTML-Funktion.

Hinzufügen des neuen Objekts zur Einfügeleiste

1 Suchen Sie am Anfang der Datei „insertbar.xml“ die folgende Zeile:

```
<category id="DW_Insertbar_Common" MMString:name="insertbar/category/common"
folder="Common">
```

Diese Zeile gibt den Anfang der Kategorie „Allgemein“ in der Einfügeleiste 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 ID muss es sich um einen eindeutigen Namen für die Schaltfläche handeln (verwenden Sie entsprechend den Benennungskonventionen für dieses Objekt die Bezeichnung `DW_Text_Strikethrough`). Die Attribute `image` und `file` geben in Dreamweaver 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 „Erneutes Laden von Erweiterungen“ auf Seite 84).

Das neue Objekt wird am Anfang der Kategorie „Allgemein“ in der Einfügeleiste angezeigt.

Hinzufügen eines Dialogfelds

Sie können dem Objekt ein Formular hinzufügen, in dem der Benutzer Parameter eingeben kann, bevor der angegebene Code eingefügt wird (der Benutzer muss z. B. für das Objekt „Hyperlink“ die Werte für Text, Hyperlink, Ziel, Kategorie, Index, Titel und Zugriffstaste eingeben). In diesem Beispiel fügen Sie dem Objekt „Strikethrough“ aus dem vorhergehenden Beispiel ein Formular hinzu. Das Formular öffnet ein Dialogfeld, in dem der Benutzer die Textfarbe in rot ändern und ein Tag zum Durchstreichen hinzufügen kann.

Bei diesem Beispiel wird vorausgesetzt, dass Sie bereits eine separate JavaScript-Datei mit dem Namen „Strikethrough.js“ erstellt haben.

Fügen Sie zunächst der Datei „Strikethrough.js“ die Funktion hinzu, die vom Formular aufgerufen wird, wenn der Benutzer die Textfarbe ändert. Diese Funktion gleicht der Funktion `objectTag()` für das Objekt „Strikethrough“, ist jedoch optional.

Erstellen der Funktion

1 Erstellen Sie nach der Funktion `objectTag()` in der Datei „Strikethrough.js“ eine Funktion mit dem Namen `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');
            // Manually wrap tags around selection.
            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");
                }
    // Just return -- don't do anything else.
    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, wenn der Benutzer darauf klickt. Definieren Sie das Formular mit dem Tag `form`. Mit dem table-Tag können Sie das Layout festlegen (andernfalls kann es vorkommen, dass im Dialogfeld Wörter umbrochen werden oder das Dialogfeld eine ungewöhnliche Größe hat).

Hinzufügen des Formulars

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 „Erneutes Laden von Erweiterungen“ auf Seite 84).

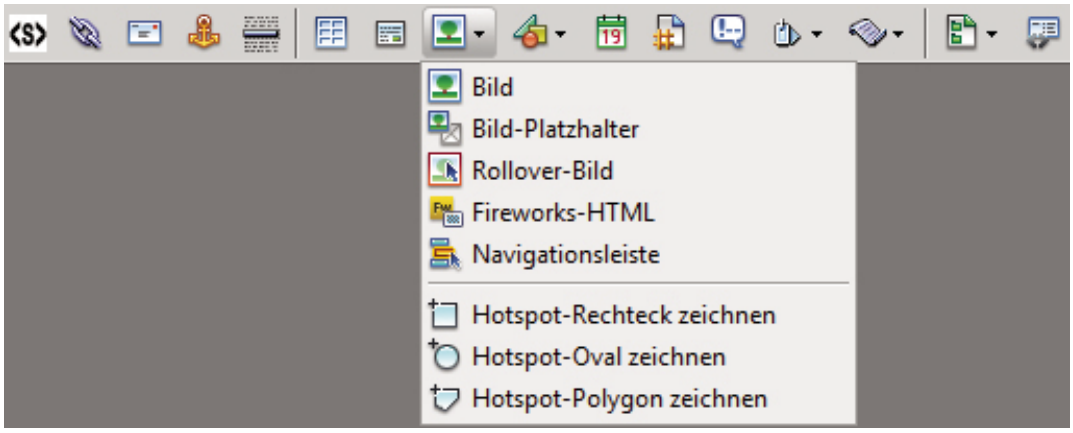
Testen des Dialogfelds

1 Aktivieren Sie das Kontrollkästchen „Red Text“.

2 Klicken Sie auf „OK“, um die Funktion `objectTag()` auszuführen, die den Text durchstreicht.

Erstellen eines Popupmenüs für die Einfügeleiste

In der Einfügeleiste von Dreamweaver sind Objekte jetzt neu angeordnet. Die Einfügeleiste unterstützt nun Popupmenüs, mit denen Objekte in kleineren Gruppen angeordnet werden können (siehe folgende Abbildung).



Im folgenden Beispiel wird in der Einfügeleiste eine neue Kategorie mit dem Namen „Editorial“ erstellt, der dann ein Popupmenü hinzugefügt wird. Dieses Popupmenü enthält das bereits erstellte Objekt „Strikethrough“. Nun erstellen Sie das Objekt „Blue Text“ und gruppieren die beiden Objekte im Popupmenü. Mit den Objekten in der Kategorie „Editorial“ können Benutzer folgende Aktionen ausführen:

- Redigieren einer Datei
- Durchstreichen des zu löschenden Inhalts oder Kennzeichnen des neuen Texts in Blau

Organisieren der Dateien

1. Erstellen Sie im Installationsordner von Dreamweaver den neuen Ordner „Configuration/Objects/Editorial“.
2. Kopieren Sie die Dateien für das erstellte Beispielobjekt „Strikethrough“ („Strikethrough.htm“, „Strikethrough.js“ und „Strikethrough.gif“) in den Ordner „Editorial“.

Erstellen des Objekts „Blue Text“

1. Erstellen Sie eine HTML-Datei.
2. Fügen Sie folgenden Code hinzu:

```
<html>
<head>
<title>Blue Text</title>
<script language="javascript">
//--------------- API FUNCTIONS---------------
function isDOMRequired() {
    // Return false, indicating that this object is available in Code view.
    return false;
}
function objectTag() {
    // Manually wrap tags around selection.
    var dom = dw.getDocumentDOM();
    if (dw.getFocus() == 'textView' || dw.getFocus(true) == 'html'){
        var upCaseTag = (dw.getPreferenceString("Source Format", "Tags Upper Case", "") ==
        'TRUE');
         // Manually wrap tags around selection.
         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");
            }
    // Just return -- don't do anything else.
    return;
}
</script>
</head>
<body>
</body>
</html>
```

3 Speichern Sie die Datei unter dem Namen „AddBlue.htm“ im Ordner „Editorial“.

Nun können Sie ein Bild für das Objekt „Blue Text“ erstellen.

Erstellen des Bilds

1 Erstellen Sie eine GIF-Datei im Format 18 x 18 Pixel (siehe folgende Abbildung).

2 Speichern Sie das Bild unter dem Namen „AddBlue.gif“ im Ordner „Editorial“.

Als Nächstes bearbeiten Sie die Datei „insertbar.xml“. Diese Datei definiert die Objekte und deren Positionen auf der Einfügeleiste. Innerhalb der `category`-Tags sind verschiedene `menubutton`-Tags mit Attributen vorhanden. Diese `menubutton`-Tags definieren die einzelnen Popupmenüs der Kategorie „HTML“. Innerhalb der `menubutton`-Tags definiert jedes `button`-Tag ein Element im Popupmenü.

Bearbeiten der Datei „insertbar.xml“

1 Suchen Sie im Anfangsbereich der Datei die folgende Codezeile:

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

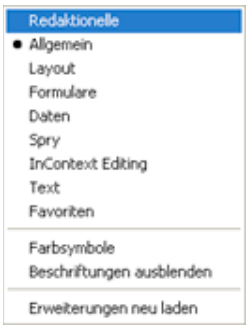
Das Tag `insertbar` definiert den Anfang des Bereichs für die Einfügeleiste.

2 Fügen Sie nach dieser Zeile ein neues `category`-Tag 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 „Erneutes Laden von Erweiterungen“ auf Seite 84.

Auf der Einfügeleiste wird die Kategorie „Editorial“ angezeigt:



4 Fügen Sie zwischen dem öffnenden und dem schließenden `category`-Tag 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 Einfügeleiste“ auf Seite 116.

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 Objekts „Strikethrough“ wie folgt das Objekt „Hypertext“ 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 gibt die gesamte Kategorie mit dem Popupmenü und den beiden Objekten an:

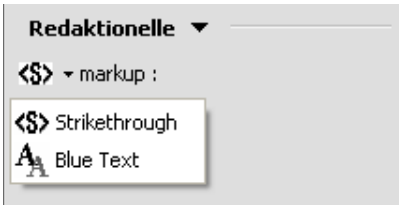
```
<category id="DW_Insertbar_Editorial" name="Editorial" folder="Editorial">
    <menubutton id="DW_Insertbar_Markup" name="markup"
    image="Editorial\Strikethrough.gif" folder="Editorial">
        <button id="DW_Editorial_Strikethrough"
        image="Editorial\Strikethrough.gif" file="Editorial\Strikethrough.htm"/>
        <button id="DW_Blue_Text" image="Editorial\AddBlue.gif" name="Blue Text"
        file="Editorial\AddBlue.htm"/>
    </menubutton>
</category>
```

Testen des neuen Popupmenüs

1 Laden Sie die Erweiterungen neu. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erneutes Laden von Erweiterungen“ auf Seite 84.

2 Klicken Sie auf das Menü „Editorial“.

Das folgende Popupmenü wird angezeigt:



API-Funktionen für Objekte

In diesem Abschnitt werden die Funktionen der API für Objekte beschrieben. Sie müssen entweder die Funktion `insertObject()` oder die Funktion `objectTag()` definieren. Weitere Informationen zu diesen Funktionen finden Sie unter „insertObject()“ auf Seite 131. 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, wenn 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. In dieser Datei ist die Verwendung der Erweiterung definiert.

```
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/myObjectHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

isDOMRequired()

Beschreibung
Diese Funktion ermittelt, ob für die Funktionsfähigkeit des Objekts ein gültiges DOM erforderlich ist. 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 die 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 den Wert `true`, wenn für die Funktionsfähigkeit eines Befehls ein gültiges DOM erforderlich ist, andernfalls `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. Diese Funktion fügt entweder Code in das Dokument des Benutzers ein und schließt das Dialogfeld oder sie zeigt eine Fehlermeldung an und schließt das Dialogfeld nicht. Diese Funktion kann in Objekten anstelle der Funktion `objectTag()` verwendet werden. Dabei wird nicht davon ausgegangen, dass der Benutzer an der aktuellen Einfügemarke Text einfügt. Die Überprüfung der Daten erfolgt, wenn 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, wenn 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 = "",
     var isValid = true;
     // ensure that field values are complete and valid
     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) {
         // do some document manipulation here. Exercise left to the reader
        }
    return errMsg;
}
```

objectTag()

Beschreibung

Die Funktionen `objectTag()` und `insertObject()` schließen sich gegenseitig aus. Wenn in einem Dokument beide Funktionen definiert sind, wird `objectTag()` verwendet. Weitere Informationen finden Sie unter „insertObject()“ auf Seite 131.

Diese Funktion fügt einen Codestring in das Dokument des Benutzers ein. Die Rückgabe eines leeren Strings oder des Werts `null` (auch als „`return`“ bezeichnet) gibt an, dass keine Operation in Dreamweaver ausgeführt wird.

Hinweis: *Dabei wird davon ausgegangen, dass die Änderungen bereits manuell vor der Anweisung* `return` *erfolgt sind. Die Tatsache, dass keine Operation ausgeführt wird, ist in diesem Fall nicht gleichbedeutend mit dem Klicken auf „Abbrechen“.*

Wenn in Dreamweaver die Codeansicht den Eingabefokus hat und es sich bei der Auswahl um einen Bereich handelt (d. h. nicht um einen Einfügepunkt), wird der Bereich durch den String ersetzt, den die Funktion `objectTag()` zurückgibt. Dabei handelt es sich um den Wert `true`, auch wenn die Funktion `objectTag()` einen leeren String oder keinen Wert zurückgibt. Die Funktion `objectTag()` gibt einen leeren String oder den Wert `null` zurück, da das Dokument bereits manuell bearbeitet wurde. Andernfalls wird die Bearbeitung häufig durch doppelte Anführungszeichen (" ") gelöscht und die Auswahl ersetzt.

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet den String, der in das Dokument des Benutzers eingefügt werden soll.

Beispiel

Im folgenden Beispiel der Funktion `objectTag()` wird eine Kombination des Typs `OBJECT/EMBED` für ein bestimmtes ActiveX-Steuerelement und ActiveX-Plug-In eingefügt:

```
function objectTag() {
return '\n' +
'<OBJECT CLASSID="clsid:166F1OOB-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: *Definieren Sie diese Funktion nur, 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 Plattform des Benutzers.

Rückgabewerte

Dreamweaver erwartet einen String des Typs `"widthInPixels,heightInPixels"`.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da der Bereich für die Schaltflächen „OK“ und „Abbrechen“ nicht einbezogen ist. Wenn im Fenster mit den zurückgegebenen Abmessungen nicht alle Optionen angezeigt werden können, werden Bildlaufleisten eingeblendet.

Beispiel

Im folgenden Beispiel der Funktion `windowDimensions()` werden die Abmessungen des Dialogfelds „Parameter“ auf 648 x 520 Pixel (Windows) bzw. 660 x 580 Pixel (Macintosh) festgelegt:

```
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 können Sie mithilfe der Funktion für die Browserkompatibilitätsprüfung (BCC, Browser Compatibility Check) Seitenlayouts erstellen, die in verschiedenen Browsern zuverlässig funktionieren (d. h. gleich aussehen und gleich funktionieren). Dabei wird nach HTML-CSS-Kombinationen gesucht, die Fehler in der Browserwiedergabe auslösen können. Diese Funktion verwendet JavaScript-Code, um das Dokument des Benutzers nach problematischen Kombinationen von HTML- und CSS-Code zu durchsuchen. Der JavaScript-Code wird in HTML-Dateien gespeichert, den sogenannten Problemerkennungsdateien. Damit diese korrekt funktionieren, müssen sie im Ordner „Configuration/BrowserProfiles/Issues/“ gespeichert werden.

Funktionsweise der Erkennung

Beim erstmaligen Ausführen einer Browserkompatibilitätsprüfung (und immer, wenn der Benutzer im Dialogfeld „Zielbrowser“ auf „OK“ klickt), werden folgende Vorgänge durchgeführt:

1. Dreamweaver liest die Profile für die ausgewählten Browser aus dem Ordner „Configuration/BrowserProfiles/“ ein.
2. Dreamweaver ruft für jede 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, sofern sie definiert wurde.
4. Dreamweaver ruft für jedes Problem die Funktion `getAffectedBrowserProfiles()` auf, um festzustellen, ob sich das Problem auf einen oder mehrere der ausgewählten 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 erkannt wurde.
6. In Dreamweaver wird für jedes erkannte Problem die Funktion `getIssueDescription` aufgerufen, um den Text zu ermitteln, der rechts im Bedienfeld „Ergebnisse“ und in der Codeansicht als QuickInfo angezeigt wird, wenn der Benutzer den Mauszeiger über eines der Probleme bewegt.

Nach Schritt 6 der oben aufgeführten Schritte und bei allen anschließend durchgeführten Browserkompatibilitätsprüfungen werden für jeden im Dialogfeld „BCC-Einstellungen“ ausgewählten Browser die folgenden Ereignisse ausgelöst.

Reihenfolge der Ereignisse

1. Dreamweaver analysiert die Stile, die für das aktuelle Dokument gelten, wie sie im jeweiligen Browser gelesen werden. Dabei werden Inline-Stile sowie im head-Bereich oder in einem externen Stylesheet definierte Stile geprüft.
2. Dreamweaver ruft für jede Problemdatei des entsprechenden Browsers die Funktion `findIssue()` auf.

Beispiele für Probleme

Bei den folgenden Beispielen handelt es sich um die Dateien `ColAndColgroupCapturedByCaption.htm` und `ColAndColgroupCapturedByCaption.js` im Ordner „Configuration/BrowserProfiles/Issues/“.

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>
//--------------- LOCALIZEABLE GLOBALS------------------
var ISSUE_NAME = "Col and Colgroup/Caption Conflict";
var ISSUE_DESC = "If the caption tag is placed directly after the opening table tag as required
by the HTML 4.01 specification, any styles applied to col and colgroup tags in the same table
are ignored.";

//--------------- END LOCALIZEABLE---------------
</script>
</head>

<body>
</body>
</html>
```

ColAndColgroupCapturedByCaption.js

```
function findIssue(){
  var DOM = dw.getDocumentDOM();
  var issueNodes = new Array();

  if (DOM){
    // first see if there are any caption tags in the doc.
    var captions = DOM.getElementsByTagName('caption');

    // declare a mess of variables that we'll need in the
    // for loop below.
    var currCap = null, props = null, parentTable = null;
    var colgroups = null, cols = null, allcol = null;
    var property = "", definedStyles = new Array();

    // ok, now loop through all the captions, if any.
    for (var i=0; i < captions.length; i++){
      currCap = captions[i];
      parentTable = currCap.parentNode;

      // the caption is only a problem if it's in the valid
      // spot (i.e., the first child of the table)
      if (currCap == parentTable.childNodes[0]){

        // find all colgroup and col tags that are in the
        // same table as the caption.
        colgroups = parentTable.getElementsByTagName('colgroup');
        cols = parentTable.getElementsByTagName('col');
        allcol = colgroups.concat(cols);

        for (var x=0; x < allcol.length; x++){
```

```
          // if styles are declared for any colgroup or col
          // tag in this table, we have a problem node. don't
          // bother looking further.
          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(){
  //DETCON 4
  return issueUtils.CONFIDENCE_HIGH;
}
```

Funktionen der Problem-API

Alle Funktionen der Problem-API sind obligatorisch, mit Ausnahme von `getAffectedBrowserDisplayNames()`. Wie bei allen APIs für Erweiterungen müssen Sie den body-Bereich jeder Funktion selbst programmieren und den geeigneten Wert an Dreamweaver zurückgeben. Informationen zu Funktionen für die Browserkompatibilitätsprüfung finden Sie im Abschnitt „Seiteninhalt“ im *Dreamweaver API-Referenzhandbuch*.

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 von Elementknoten, die das Problem angeben. Dreamweaver wählt diese Knoten aus, wenn der Benutzer die verschiedenen Browserkompatibilitätsprobleme auswählt.

Beispiel
Die folgende `findIssue()`-Funktion gibt ein Array von `<button>`-Tags zurück, auf die `float: left` oder `float: right` angewendet wurde.

```
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.getDeclaredStyle(buttons[i]);
        if (props.cssFloat == "left" || props.cssFloat == "right"){
            issueNodes.push(buttons[i]);
        }
    }
    return issueNodes;
}
```

getAffectedBrowserProfiles()

Verfügbarkeit
Dreamweaver CS3.

Beschreibung
Zeigt in Dreamweaver eine Liste mit Browsern an, für die das Problem relevant ist.

Argumente
Keine.

Rückgabewerte
Ein Array von Browsernamen, von denen jeder mit der ersten Zeile in einem gültigen Browserprofil genau übereinstimmen muss (siehe 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
Zeigt in Dreamweaver eine Liste mit den dem Benutzer angezeigten Browsernamen an, 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 von 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
Gibt in Dreamweaver eine eindeutige ID für das Problem zurück.

Argumente
Keine.

Rückgabewerte
Ein String mit einem eindeutigen Bezeichner für das Problem.

Beispiel
```
function getIssueID() {
    return "EXPANDING_BOX_PROBLEM";
}
```

getIssueName()

Verfügbarkeit
Dreamweaver CS3.

Beschreibung
Zeigt in Dreamweaver den Namen oder die Kurzbeschreibung des Problems an.

Argumente
Keine.

Rückgabewerte
Ein String mit dem Namen oder einer Kurzbeschreibung des Problems.

Beispiel

```
function getIssueName(){
    return "The Expanding Box Problem";
}
```

getIssueDescription()

Verfügbarkeit
Dreamweaver CS3.

Beschreibung
Zeigt in Dreamweaver eine ausführliche Beschreibung des Problems an.

Argumente
Keine.

Rückgabewerte
Ein String mit dem Namen oder einer Kurzbeschreibung 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 können praktisch alle gewünschten Bearbeitungsvorgänge am aktuellen Dokument des Benutzers, an anderen geöffneten Dokumenten oder an HTML-Dokumenten auf dem lokalen Laufwerk durchgeführt werden. Befehle dienen dazu, HTML-Tags und HTML-Attribute sowie Kommentare und Text 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 (z. B. 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 sind die Dateien zum Erstellen von Befehlen aufgeführt.

Pfad	Datei	Beschreibung
Configuration/Commands/	*Befehlsname*.htm	Gibt die Benutzeroberfläche an.
Configuration/Commands/	*Befehlsname*.js	Enthält die auszuführenden Funktionen.

Funktionsweise von Befehlen

Wenn ein Benutzer auf ein Menü klickt, das einen Befehl enthält, werden folgende Vorgänge durchgeführt:

1. Dreamweaver ruft die Funktion `canAcceptCommand()` auf, um zu prüfen, ob das Menüelement deaktiviert werden muss. 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. Wenn sie definiert ist, wird die Funktion `receiveArguments()` in der ausgewählten Befehlsdatei 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. Wenn sie definiert ist, wird die Funktion `commandButtons()` aufgerufen, um zu prüfen, welche Schaltflächen auf der rechten Seite des Dialogfelds „Optionen“ 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 form-Tag vorhanden ist, ruft Dreamweaver die Funktion `windowDimensions()` auf, mit der die Größe des Dialogfelds „Optionen“ geändert wird, in dem sich die body-Elemente der Datei befinden. Wenn die Funktion `windowDimensions()` nicht definiert ist, wird die Größe des Dialogfelds automatisch angepasst.
6. Wenn das `body`-Tag der Befehlsdatei eine Ereignisprozedur des Typs `onLoad` enthält, wird diese in Dreamweaver ausgeführt (unabhängig davon, ob ein Dialogfeld angezeigt wird). Wenn kein Dialogfeld angezeigt wird, entfallen die restlichen Schritte.
7. Der Benutzer wählt Optionen für den Befehl aus. Die den einzelnen Feldern zugeordneten Ereignisprozeduren werden in Dreamweaver ausgeführt, wenn die Felder vom Benutzer aktiviert werden.
8. Der Benutzer klickt auf eine der durch die Funktion `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.

Hinzufügen von Befehlen zum Menü „Befehle“

Dateien, die sich im Unterordner „Configuration/Commands“ befinden, werden in Dreamweaver automatisch am Ende des Menüs „Befehle“ eingefügt. Wenn ein Befehl nicht im Menü „Befehle“ aufgeführt werden soll, müssen Sie den folgenden Kommentar in die erste Zeile der Datei einfügen:

```
<!-- MENU-LOCATION=NONE -->
```

Wenn diese Zeile vorhanden ist, erstellt Dreamweaver keine Menüoption für die Datei. Sie müssen den Befehl dann über `dw.runCommand()` aufrufen.

Einfaches Befehlsbeispiel

Mit dieser einfachen Erweiterung wird dem Menü „Befehle“ ein Element hinzugefügt, mit dem Sie ausgewählten Text in Ihrem Dokument entweder in Groß- oder Kleinbuchstaben umwandeln können. Wenn Sie auf das Menüelement klicken, wird eine Benutzeroberfläche mit drei Schaltflächen angezeigt, über die Sie Ihre Auswahl festlegen können.

Zum Erstellen dieser Erweiterung erstellen Sie die Benutzeroberfläche, programmieren den JavaScript-Code und testen dann die Erweiterung.

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-Bereich einfügen.

Erstellen der Benutzeroberfläche

Die Benutzeroberfläche besteht aus einem Formular mit zwei Optionen, mit denen der Benutzer entweder die Klein- oder Großschreibung auswählen kann.

1 Erstellen Sie eine leere Datei.

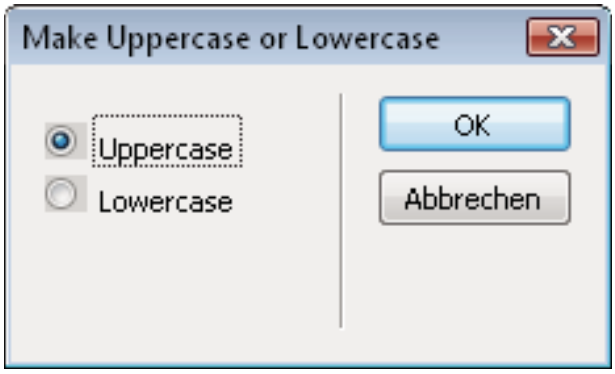
2 Fügen Sie der Datei den folgenden Code zum Erstellen des Formulars hinzu:

```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//dialog">
<HTML>
<HEAD>
<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 Titelleiste des Dialogfelds angezeigt. Innerhalb des Formulars wird das Layout der Elemente durch eine Tabelle mit zwei Zellen festgelegt. In den Tabellenzellen befinden sich die beiden Optionen, „Uppercase“ und „Lowercase“. Die Option „Uppercase“ hat das Attribut `checked`. Dadurch ist sie standardmäßig ausgewählt und es wird sichergestellt, dass der Benutzer entweder eine der beiden Optionen auswählt oder den Befehl abbricht.

Das Formular ist in der folgenden Abbildung dargestellt.



Die Funktion `commandButtons()` stellt die Schaltflächen „OK“ und „Abbrechen“ bereit, mit denen ein Benutzer eine Auswahl übermitteln oder den Vorgang abbrechen kann. Weitere Informationen finden Sie unter „commandButtons()“ auf Seite 149.

Programmieren des JavaScript-Codes

Das folgende Beispiel besteht aus den beiden erweiterten API-Funktionen `canAcceptCommand()` und `commandButtons()`, die von Dreamweaver aufgerufen werden, sowie der benutzerdefinierten Funktion `changeCase()`, die über die Funktion `commandButtons()` aufgerufen wird.

In diesem Beispiel programmieren Sie JavaScript-Code für die folgenden Aufgaben:

Ermitteln, ob der Befehl aktiviert oder abgeblendet ist

Beim Erstellen eines Befehls müssen Sie zunächst ermitteln, wann ein Element aktiviert und wann es abgeblendet dargestellt wird. Wenn ein Benutzer auf das Menü „Befehle“ klickt, ruft Dreamweaver für jedes Menüelement die Funktion `canAcceptCommand()` auf, um festzustellen, ob es aktiviert ist. Wenn `canAcceptCommand()` den Wert `true` zurückgibt, zeigt Dreamweaver den Text des Menüelements aktiviert an. Wenn `canAcceptCommand()` den Wert `false` zurückgibt, wird das Menüelement abgeblendet. In diesem Beispiel ist das Menüelement aktiv, wenn der Benutzer im Dokument Text ausgewählt hat.

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie folgenden Code ein:

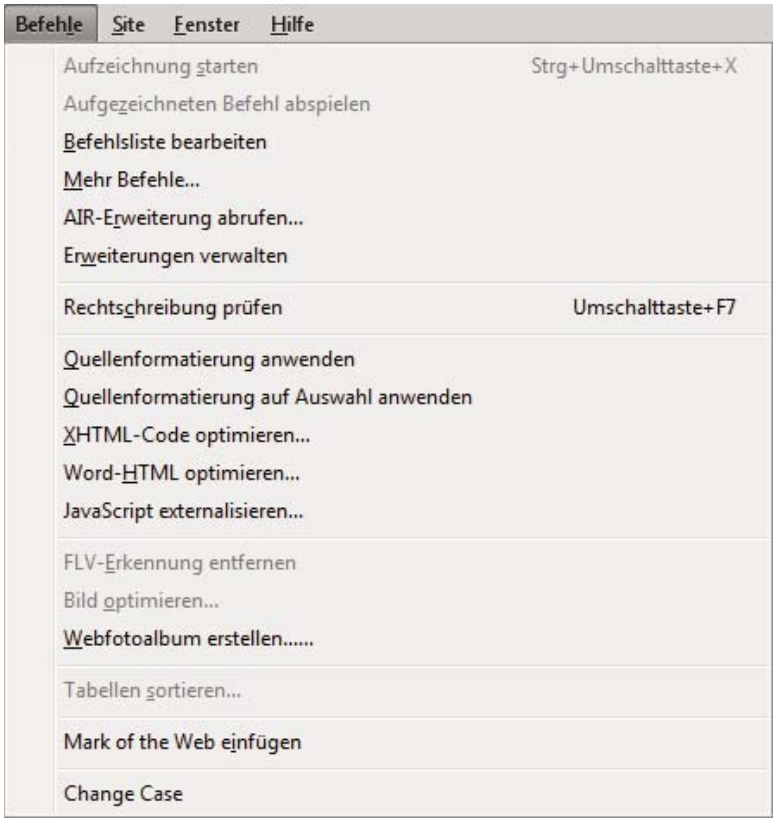
```
function canAcceptCommand(){
    var theDOM = dw.getDocumentDOM(); // Get the DOM of the current document
    var theSel = theDOM.getSelection(); // Get start and end of selection
    var theSelNode = theDOM.getSelectedNode(); // Get the selected node
    var theChildren = theSelNode.childNodes; // Get children of selected node
    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“.

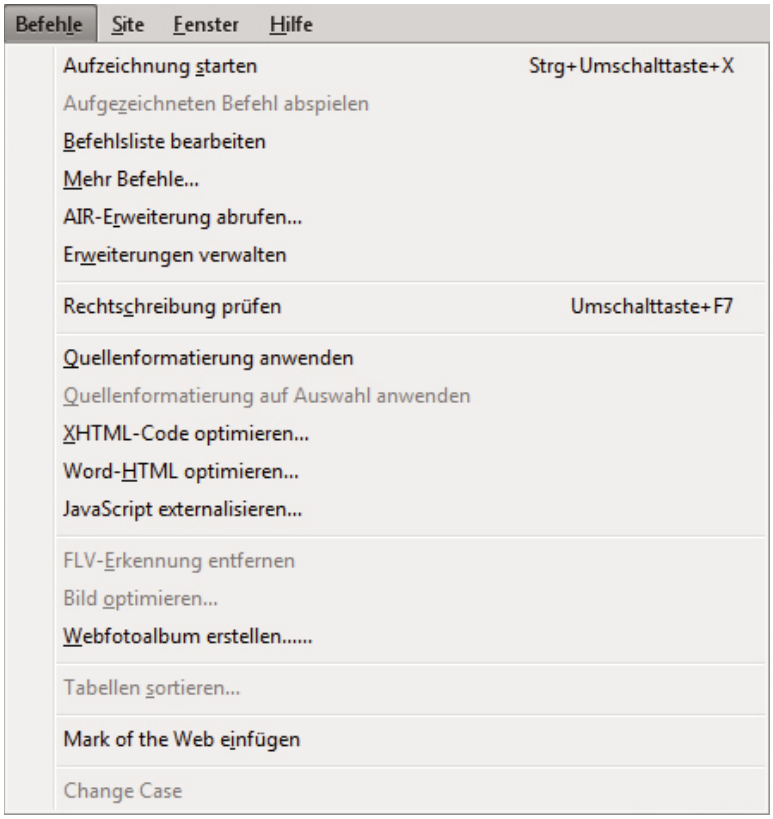
Die ersten Zeilen der Funktion `canAcceptCommand()` rufen den ausgewählten Text ab, indem sie das DOM des Benutzerdokuments abrufen und für das Dokumentobjekt die Funktion `getSelection()` aufrufen. Anschließend ruft die Funktion den Knoten ab, der den ausgewählten Text enthält, gefolgt von den untergeordneten Knoten, wie im Codebeispiel dargestellt. Mit der letzten Zeile wird überprüft, ob es sich bei der Auswahl oder dem ersten untergeordneten Knoten um Text handelt. Als Ergebnis wird `true` oder `false` zurückgegeben.

Der erste Teil der `return`-Anweisung (`theSel[0] != theSel[1]`) überprüft, ob der Benutzer einen Bereich 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 kein Inhalt ausgewählt.

Der nächste Teil der `return`-Anweisung (`&& (theSelNode.nodeType == Node.TEXT_NODE`) überprüft, ob der ausgewählte Knoten den Typ „Text“ hat. Wenn ja, gibt die Funktion `canAcceptCommand()` den Wert `true` zurück. Wenn der Knoten nicht den Typ „Text“ hat, wird der Test fortgesetzt, um zu überprüfen, ob untergeordnete Knoten vorliegen (`|| theSelNode.hasChildNodes()`) ob der erste untergeordnete Knoten den Typ „Text“ hat (`&& (theChildren[0].nodeType == Node.TEXT_NODE)`). Wenn beide Bedingungen wahr sind, gibt `canAcceptCommand()` den Wert `true` zurück und Dreamweaver aktiviert das Menüelement am Ende des Menüs „Befehle“ (siehe folgende Abbildung).



Andernfalls gibt `canAcceptCommand()` den Wert `false` zurück und Dreamweaver blendet das Menüelement ab (siehe folgende Abbildung).



Verknüpfen von Funktionen mit den Schaltflächen „OK“ und „Abbrechen“

Wenn der Benutzer auf „OK“ oder „Abbrechen“ klickt, muss mit der Erweiterung die entsprechende Aktion ausgeführt werden. Sie legen die entsprechende Aktion fest, indem Sie angeben, welche JavaScript-Funktion beim Klicken auf die jeweilige Schaltfläche ausgeführt werden soll.

1 Öffnen Sie die Datei „ChangeCase.js“ im Ordner „Configuration/Commands“.

2 Fügen Sie am Ende der Datei folgenden Code ein:

```
function commandButtons() {
    return new Array("OK", "changeCase()", "Cancel", "window.close()");
}
```

3 Speichern Sie die Datei.

Die Funktion `commandButtons()` sorgt dafür, dass in Dreamweaver die Schaltflächen „OK“ und „Abbrechen“ hinzugefügt werden, und gibt an, welche Aktionen beim Klicken auf die Schaltflächen ausgeführt werden. Dreamweaver wird durch die Funktion `commandButtons()` angewiesen, `changeCase()` aufzurufen, wenn der Benutzer auf „OK“ klickt, und `window.close()`, wenn der Benutzer auf „Abbrechen“ klickt.

Auswählen der Groß- oder Kleinschreibung durch den Benutzer

Wenn der Benutzer auf ein Menüelement klickt, muss für die Erweiterung ein Mechanismus angegeben werden, mit dem der Benutzer die Groß- oder Kleinschreibung auswählen kann. Die Benutzeroberfläche stellt diesen Mechanismus in Form von zwei Optionsschaltern zur Verfügung, mit denen Benutzer diese Auswahl treffen können.

1 Öffnen Sie die Datei „ChangeCase.js“.

2 Fügen Sie am Ende der Datei folgenden Code ein:

```
function changeCase() {
    var uorl;
    // Check whether user requested uppercase or lowercase.
    if (document.forms[0].elements[0].checked)
        uorl = 'u';
    else
        uorl = 'l';
    // Get the DOM.
    var theDOM = dw.getDocumentDOM();
    // Get the outerHTML of the HTML tag (the
    // entire contents of the document).
    var theDocEl = theDOM.documentElement;
    var theWholeDoc = theDocEl.outerHTML;
    // Get the node that contains the selection.
    var theSelNode = theDOM.getSelectedNode();
    // Get the children of the selected node.
    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 {
    // Get the offsets of the selection
    var theSel = theDOM.getSelection();
    // Extract the selection
    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]);
    }
// Set the selection back to where it was when you started
    theDOM.setSelection(theSel[0],theSel[1]);
    window.close(); // close extension UI
}
```

3 Speichern Sie die Datei unter dem Namen „ChangeCase.js“ im Ordner „Configuration/Commands“.

Die Funktion `changeCase()` ist eine benutzerdefinierte Funktion, die von `commandButtons()` aufgerufen wird, wenn der Benutzer auf „OK“ klickt. Diese Funktion wandelt den ausgewählten Text in Klein- bzw. Großschreibung um. Da in der Benutzeroberfläche Optionsschalter verwendet werden, ist sichergestellt, dass eine Option aktiviert wird. Es muss daher nicht geprüft werden, ob der Benutzer keine Auswahl getroffen hat.

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, d. h. in der Benutzeroberfläche der Erweiterung. Die Eigenschaft `checked` hat den Wert `true`, wenn das Element ausgewählt (bzw. aktiviert) ist, und `false`, wenn es nicht ausgewählt ist. In der Benutzeroberfläche bezieht sich `elements[0]` auf den ersten Optionsschalter, d. h. 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 Benutzerauswahl zu speichern.

Der restliche Code in der Funktion ruft den ausgewählten Text ab, wandelt ihn in Groß- oder Kleinschreibung um und kopiert ihn zurück in das Dokument.

Zum Abrufen des ausgewählten Texts aus dem Dokument des Benutzers ruft die Funktion das DOM ab. Dann wird das Stammelement des Dokuments, das `html`-Tag, abgerufen. Abschließend wird das gesamte Dokument in die Variable `theWholeDoc` extrahiert.

Anschließend ruft `changeCase()` mithilfe von `getSelectedNode()` den Knoten ab, der den ausgewählten Text enthält. Zudem werden alle untergeordneten Knoten (`theSelNode.childNodes`) abgerufen, falls ein Tag ausgewählt wurde, das Text enthält, z. B. `<b>text</b>`.

Wenn untergeordnete Knoten vorhanden sind, gibt `hasChildNodes()` den Wert `true` zurück und der Befehl führt eine Schleife durch die untergeordneten Knoten aus, um nach einem Textknoten zu suchen. Wenn ein Textknoten gefunden wird, 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 abschnittsweise in das Dokument zurück: zunächst den Anfang des Dokuments bis zum Beginn der Auswahl, dann den ausgewählten Text in Großbuchstaben (`selText.toUppercase()`) und zuletzt das Ende des ausgewählten Texts 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 umzuwandeln.

Schließlich setzt `changeCase()` die Auswahl zurück und ruft `window.close()` auf, um die Benutzeroberfläche zu schließen.

Testen der Erweiterung

Nachdem Sie die Dateien im Ordner „Commands“ abgelegt haben, können Sie die Erweiterung testen.

1 Starten Sie Dreamweaver neu oder laden Sie die Erweiterungen neu. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erneutes Laden von Erweiterungen“ auf Seite 84.

   Daraufhin wird im Menü „Befehle“ der Eintrag „Change Case“ angezeigt.

2 Geben Sie Text in einem Dokument ein.

3 Wählen Sie den Text aus.

Hinweis: *Der Eintrag „Change Case“ wird abgeblendet angezeigt, bis in einem Dokument Text ausgewählt wird.*

4 Wählen Sie „Change Case“ im Menü „Befehle“ aus.

Die Groß-/Kleinschreibung des Texts ändert sich.

API-Funktionen für Befehle

Die benutzerdefinierten Funktionen der Befehls-API sind nicht obligatorisch.

canAcceptCommand()

Beschreibung

Diese Funktion ermittelt, ob der Befehl für die aktuelle Auswahl geeignet ist.

Hinweis: *Definieren Sie* `canAcceptCommand()` *nur, wenn mindestens in einem Fall* `false` *zurückgegeben wird. Ist die Funktion nicht definiert, gilt der Befehl als geeignet. Durch diese Annahme wird der Vorgang beschleunigt und die Leistung verbessert.*

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet den Wert `true`, wenn der Befehl zulässig ist, andernfalls `false`. In diesem Fall wird der Befehl im Menü abgeblendet.

Beispiel

Im folgenden Beispiel für `canAcceptCommand()` ist der Befehl nur verfügbar, 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, die im Dialogfeld „Optionen“ angezeigt werden, und das entsprechende Verhalten, wenn auf die Schaltflächen geklickt wird. Wenn diese Funktion nicht definiert ist, werden keine Schaltflächen angezeigt und der body-Bereich der Befehlsdatei füllt das gesamte Dialogfeld aus.

Diese Schaltflächen werden standardmäßig im oberen Bereich des Dialogfelds angezeigt. Sie können festlegen, dass diese Schaltfläche im unteren Bereich des Dialogfelds angezeigt werden, indem Sie in der Funktion `commandButtons()` zwei weitere Werte angeben.

Normalerweise sind Schaltflächen rechts ausgerichtet. Durch den Wert *PutButtonOnLeft* werden weitere Schaltflächen in der gleichen Zeile links ausgerichtet.

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 Verhalten beim Klicken auf die oberste Schaltfläche festgelegt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel
Mit der folgenden Instanz von `commandButtons()` werden drei Schaltflächen definiert: OK, Cancel (Abbrechen) und Help (Hilfe), die rechts oben im Dialogfeld (Standardposition) angezeigt werden.

```
function commandButtons(){
return new Array("OK" , "doCommand()" ,
                "Cancel" , "window.close()" ,
                "Help" , "showHelp()");
}
```

Sie können die Position und Ausrichtung der Schaltflächen anpassen.

Beispiel
Mit der folgenden Instanz von `commandButtons()` werden die Schaltflächen im unteren Bereich des Dialogfelds angezeigt. Wenn es sich bei dem ersten Wert im zurückgegebenen Array um *PutButtonsOnBottom* handelt, können Sie den zweiten Wert zusammen mit einem der Schaltflächennamen als *defaultButton* angeben. Diese Schaltfläche wird standardmäßig ausgewählt und beim Drücken der Eingabetaste verwendet. In diesem Beispiel ist `OKbutton` als Standardschaltfläche definiert.

```
function commandButtons(){
return new Array("PutButtonsOnBottom" , "OkButton defaultButton" ,
                "OK" , "doCommand()" ,
                "Cancel" , "window.close()" ,
                "Help" , "showHelp()");
}
```

Beispiel
Im folgenden Beispiel wird die Schaltfläche „Help“ (Hilfe) mithilfe von *PutButtonsOnLeft* links ausgerichtet.

```
function commandButtons(){
return new Array("PutButtonsOnBottom", "OkButton defaultButton",
                 "OK", "doCommand()",
                 "Cancel", "window.close()",
                 "PutButtonOnLeft",
                 "Help" , "showHelp()");}
```

isDOMRequired()

Beschreibung
Diese Funktion ermittelt, ob für die Funktionsfähigkeit des Befehls ein gültiges DOM erforderlich ist. 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 den Wert `true`, wenn für die Funktionsfähigkeit eines Befehls ein gültiges DOM erforderlich ist, andernfalls `false`.

receiveArguments()

Beschreibung
Diese Funktion verarbeitet alle 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 mindestens ein Argument 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: *Definieren Sie diese Funktion nur, 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 Plattform des Benutzers.

Rückgabewerte
Dreamweaver erwartet einen String des Typs `"widthInPixels,heightInPixels"`.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da der Bereich für die Schaltflächen „OK“ und „Abbrechen“ nicht einbezogen ist. Wenn im Fenster mit den zurückgegebenen Abmessungen nicht alle Optionen angezeigt werden können, werden Bildlaufleisten eingeblendet.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Dialogfelds „Parameter“ auf 648 x 520 Pixel gesetzt.

```
function windowDimensions(){
    return "648,520";
}
```

Kapitel 10: Menüs und Menübefehle

Die Menüs in Adobe Dreamweaver basieren auf der in der Datei „menus.xml“ im Dreamweaver-Ordner „Configuration/Menus“ definierten Struktur. Durch Bearbeiten der Datei „menus.xml“ können Sie Menübefehle neu anordnen, umbenennen und entfernen. Außerdem haben Sie die Möglichkeit, Tastaturbefehle für Menübefehle hinzuzufügen, zu ändern oder zu entfernen. Dies kann in der Regel jedoch einfacher mithilfe des Tastaturbefehl-Editors durchgeführt werden (siehe Dreamweaver-Hilfe). Änderungen an Dreamweaver-Menüs werden nach dem Neustart von Dreamweaver oder dem Neuladen von Erweiterungen wirksam.

Datei „menus.xml“

Die Datei „menus.xml“ enthält eine Auflistung mit Menüleisten, Menüs, Menübefehlen, Trennzeichen, Kurzbefehlen und Tastaturbefehlen. Diese Elemente werden mit XML-Tags beschrieben und können in einem Texteditor bearbeitet werden.

Hinweis: *Gehen Sie bei Menüänderungen sorgfältig vor. In Dreamweaver werden keine Menüs oder Menübefehle verarbeitet, die XML-Syntaxfehler aufweisen.*

In einem Mehrbenutzer-Betriebssystem können Sie Änderungen in Dreamweaver vornehmen, die sich auf die Datei „menus.xml“ auswirken. Dazu gehören z. B. Änderungen der Tastaturbefehle mithilfe des Tastaturbefehl-Editors. In diesen Fällen erstellt Dreamweaver eine Datei „menus.xml“ in Ihrem Benutzer-Konfigurationsordner. Wenn Sie die Datei „menus.xml“ in einem Mehrbenutzer-Betriebssystem anpassen möchten, bearbeiten Sie die Kopie der Datei in Ihrem Benutzer-Konfigurationsordner. Kopieren Sie die Masterdatei „menus.xml“ in Ihren Benutzer-Konfigurationsordner, wenn in diesem Ordner noch keine Kopie erstellt wurde. Weitere Informationen finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 83.

Beim Öffnen der Datei „menus.xml“ in einem XML-Editor werden unter Umständen Fehlermeldungen bezüglich der Et-Zeichen (&) in der Datei angezeigt. Es empfiehlt sich, die Datei „menus.xml“ in einem Texteditor und nicht in Dreamweaver zu bearbeiten. 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. Falls Sie keine Sicherungskopie erstellt haben sollten, ersetzen Sie die Datei „menus.xml“ durch eine Kopie der Datei „menus.bak“ im Konfigurationsordner.*

Eine Menüleiste (mit öffnendem und schließendem `menu bar`-Tag) kann sich aus einem einzelnen Menü oder einer Gruppe von Menüs zusammensetzen. So gibt es beispielsweise eine Hauptmenüleiste, eine separate Menüleiste für das Bedienfeld „Dateien“ (wird nur unter Windows angezeigt) und eine Menüleiste für jedes Kontextmenü. Jede Menüleiste enthält mindestens ein Menü. Ein Menü wird durch ein menu-Tag definiert. Jedes Menü enthält mindestens einen Menübefehl, der jeweils durch ein `menuitem`-Tag und die entsprechenden Attribute definiert ist. Ein Menü kann außerdem Trennzeichen (definiert durch separator-Tags) und Untermenüs enthalten.

Neben den grundlegenden Tastaturbefehlen für Menübefehle umfasst Dreamweaver auch alternative und kontextspezifische Tastaturbefehle. Strg+Y (Windows) bzw. Befehl+Y (Macintosh) ist beispielsweise der Kurzbefehl zum Wiederherstellen eines Bearbeitungsschritts. Gleichzeitig ist auch Strg+Umschalt+Z bzw. Befehl+Umschalt+Z ein alternativer Kurzbefehl für das Wiederherstellen. Diese alternativen und kontextspezifischen Kurzbefehle, die nicht durch Tags für Menübefehle wiedergegeben werden können, sind in den Kurzbefehlslisten 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 in der Datei „menus.xml“ beschrieben. Optionale Attribute sind in den Attributlisten durch geschweifte Klammern ({}) gekennzeichnet. Nicht durch geschweifte Klammern gekennzeichnete Attribute müssen angegeben werden.

<menubar>

Beschreibung
Gibt Informationen zu einer Menüleiste in der Dreamweaver-Menüstruktur an.

Attribute
`name, {app}, id, {platform}`

- `name` Der Name der Menüleiste. Obwohl 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 derzeit 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 das folgende menubar-Tag verwendet:

```
<menubar name="Main Window" id="DWMainWindow">
<!-- menu tags here -->
</menubar>
```

<menu>

Beschreibung
Gibt Informationen zu einem Menü oder einem Untermenü in der Dreamweaver-Menüstruktur an.

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 derzeit 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 sind `_SERVERMODEL_ASP`, `_SERVERMODEL_ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von Adobe 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 einer AND-Verknüpfung) trennen. Zur Angabe von NOT können Sie `"!"` verwenden. Wenn z. B. ein Menü auf einer ASP-Seite nur in der Codeansicht angezeigt werden soll, geben Sie das Attribut wie folgt an: `showIf="_VIEW_CODE, _SERVERMODEL_ASP"`.

Inhalt

Dieses Tag kann ein oder mehrere `menuitem`-Tags und `separator`-Tags enthalten. Außerdem kann es weitere `menu`-Tags (zum Erstellen von Untermenüs) und Tags für HTML-Standardkommentare enthalten.

Container

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

Beispiel

```
<menu name="_File" id="DWMenu_File">
    <!-- menuitem, separator, menu, and comment tags here -->
</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 angegeben, dass es sich beim nächsten Buchstaben um die Zugriffstaste (Abkürzungstaste) für den Befehl handelt. Diese Funktion steht nur für Windows zur Verfügung.
- `id` Wird in Dreamweaver als Bezeichner des Elements verwendet. 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 der der Menübefehl angezeigt wird. Wird derzeit nicht verwendet.
- `key` Der Tastaturbefehl für den Menübefehl, sofern vorhanden. Legen Sie Modifizierungstasten mit den folgenden Strings fest:
- `Cmd` bezeichnet die Strg-Taste (Windows) bzw. die Befehlstaste (Macintosh).
- `Alt` und `Opt` bezeichnen beide die Alt-Taste (Windows) bzw. die Wahltaste (Macintosh).
- `Shift` bezeichnet auf beiden Plattformen die Umschalttaste.
- `Ctrl` bezeichnet auf beiden Plattformen die Steuerungstaste.

- Durch ein Pluszeichen (+) werden 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 entsprechenden 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`" (Windows) oder "`mac`" (Macintosh). Wenn Sie das Attribut `platform` nicht angeben, wird der Menübefehl auf beiden Plattformen angezeigt. Wenn ein Menübefehl auf verschiedenen Plattformen unterschiedlich ausgeführt werden soll, müssen Sie zwei Menübefehle mit dem gleichen Namen (aber unterschiedlichen IDs) angeben: einen Menübefehl mit `platform="win"` und den anderen mit `platform="mac"`.
- `enabled` Gibt JavaScript-Code an (in der Regel einen JavaScript-Funktionsaufruf), der ermittelt, ob der Menübefehl derzeit aktiviert ist. Wenn die Funktion den Wert `false` zurückgibt, ist der Menübefehl abgeblendet. Der Standardwert ist `"true"`. Der Eindeutigkeit halber empfiehlt es sich jedoch, immer einen Wert anzugeben, auch wenn der Wert `"true"` ist.
- `arguments` Gibt Argumente an, die Dreamweaver an den Code in der mit dem Attribut `file` angegebenen JavaScript-Datei übergeben soll. Setzen Sie Argumente, die sich innerhalb doppelter Anführungszeichen (") zur Begrenzung eines Attributwerts befinden, 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-Code zur Steuerung des Menübefehls. Geben Sie einen Pfad zur Datei relativ zum Ordner „Configuration“ an. (Der Menübefehl „Hilfe“ > „Willkommen“ wird z. B. durch `file="Commands/Welcome.htm"` angegeben.) Das Attribut `file` setzt die Attribute `command`, `enabled` und `checked` außer Kraft. Sie müssen für jeden Menübefehl entweder das Attribut `file` oder `command` angeben. Informationen zum Erstellen einer Befehlsdatei im Bedienfeld „Verlauf“ finden Sie in der Dreamweaver-Hilfe. Informationen zum Programmieren benutzerdefinierter JavaScript-Befehle finden Sie unter „Befehle“ auf Seite 141.
- `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, gibt es an, dass der Menübefehl dynamisch durch eine HTML-Datei festgelegt wird. Die Datei enthält JavaScript-Code zum Festlegen von Text und Status des Menübefehls. Wenn Sie ein Tag als `dynamic` festlegen, müssen Sie auch das Attribut `file` angeben.
- `isdomrequired` Gibt an, ob die Entwurfs- und die Codeansicht vor dem Ausführen des Codes für diesen Menübefehl synchronisiert werden sollen. Gültige Werte sind "`true`" (Standardwert) und "`false`". Wenn Sie dieses Attribut auf "`false`" setzen, wird das Dreamweaver-Dokumentobjektmodell (DOM) nicht für Änderungen an der Datei verwendet, die mit diesem Menübefehl erstellt wird. Informationen zum DOM finden Sie unter „Dokumentobjektmodell von Dreamweaver“ auf Seite 102).
- `showIf` Gibt an, dass das Menüelement nur angezeigt wird, wenn der jeweilige Dreamweaver-Enabler auf `true` gesetzt ist. Mögliche Enabler sind `_SERVERMODEL_ASP`, `_SERVERMODEL_ASPNET`, `_SERVERMODEL_JSP`, `_SERVERMODEL_CFML` (für alle Versionen von Adobe 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 einer AND-Verknüpfung) trennen. Zur Angabe von NOT können Sie `"!"` verwenden. Wenn der Menübefehl beispielsweise in der Codeansicht, jedoch nicht auf einer ColdFusion-Seite angezeigt werden soll, geben Sie für das Attribut `showIf="_VIEW_CODE, !_SERVERMODEL_CFML"` an.

Inhalt
Kein (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 derzeit nicht verwendet.

Inhalt
Kein (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 derzeit 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 Kommentar-Tags enthalten (die die gleiche Syntax wie HTML-Kommentar-Tags aufweisen).

Container
Keine.

Beispiel
```
<shortcutlist id="DWMainWindow">
<!-- shortcut and comment tags here -->
</shortcutlist>
```

<shortcut>

Beschreibung
Gibt einen Tastaturkurzbefehl 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 155.
- `app` Der Name der Anwendung, in der der Kurzbefehl verfügbar ist. Wird derzeit 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 zu einer Datei mit dem JavaScript-Code, den Dreamweaver bei Verwendung des Tastaturbefehls ausführt. Das Attribut `file` setzt das Attribut `command` außer Kraft. Sie müssen für jeden Kurzbefehl entweder das Attribut `file` oder `command` angeben.
- `arguments` Gibt Argumente an, die Dreamweaver an den Code in der mit dem Attribut `file` angegebenen JavaScript-Datei übergeben soll. Setzen Sie Argumente, die sich innerhalb doppelter Anführungszeichen (") zur Begrenzung eines Attributwerts befinden, 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 Bezeichner für einen Kurzbefehl.
- `name` Ein Name für den über einen Tastaturbefehl ausgeführten Befehl in Form eines Menübefehls. Das Attribut `name` für den F12-Kurzbefehl ist beispielsweise "`Vorschau im Primärbrowser`".

Inhalt
Kein (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
Gibt ein Tool an. Das Tag enthält sämtliche Kurzbefehle für das Tool als Subtags in der Datei „menus.xml“.

Attribute
`{name}, id`

- `name` Eine lokalisierte Version des Toolnamens.
- `id` Die interne Tool-ID des Tools, auf das sich die Kurzbefehle beziehen.

Inhalt
Dieses Tag kann eines oder mehrere `activate`-, `override`- und `action`-Tags 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 gedrückt wird.

Attribute
`{name}, key, command, id`

- `name` Eine lokalisierte Version der Aktion.
- `key` Die Tastenkombination zum Ausführen der Aktion. Weitere Informationen zur Syntax finden Sie unter „<menuitem>“ auf Seite 155.
- `command` Die auszuführenden JavaScript-Anweisungen. Dieses Attribut hat dasselbe Format wie das Attribut `command` von „<shortcut>“ auf Seite 158.
- `id` Eine eindeutige ID für Verweise auf die Aktion.

Inhalt
Kein (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 155.
- `id` Eine eindeutige ID für Verweise auf die Aktion.

Inhalt
Kein (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 modalen Tools kann der Benutzer diese Tastenkombination gedrückt halten, um zu diesem Tool zu wechseln.

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 155.
- `id` Eine eindeutige ID für Verweise auf die Aktion.

Inhalt
Kein (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 sowie Menüs auf einer Menüleiste oder zwischen verschiedenen Menüleisten verschieben.

Zudem können Sie Elemente auf die gleiche Weise in oder aus Kontextmenüs verschieben.

Informationen dazu finden Sie unter „Datei „menus.xml““ auf Seite 153.

Verschieben eines Menübefehls

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 ab dem öffnenden `<menuitem`-Tag bis zum `/>`-Tag am Ende aus.
5. Setzen Sie die Einfügemarke an die gewünschte Stelle für den Menübefehl. (Stellen Sie sicher, dass sich die Einfügemarke zwischen einem `menu`-Tag und dem zugehörigen `/menu`-Tag befindet.)
6. Fügen Sie den Menübefehl an der neuen Position ein.

Erstellen eines Untermenüs und Verschieben eines Menübefehls

1. Setzen Sie die Einfügemarke in ein Menü (an eine beliebige Stelle zwischen einem `menu`-Tag und dem zugehörigen `/menu`-Tag).
2. Fügen Sie in das Menü ein neues `menu` ... `/menu`-Tagpaar ein.
3. Fügen Sie dem neuen Untermenü neue Menübefehle hinzu.

Einfügen eines Trennzeichens zwischen zwei Menübefehlen

- Fügen Sie ein `separator/`-Tag zwischen zwei `menuitem`-Tags ein.

Entfernen eines vorhandenen Trennzeichens

- Löschen Sie die entsprechende `separator/`-Zeile.

Verschieben eines 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 BBEdit, HomeSite oder WordPad. (Öffnen Sie die Datei jedoch nicht in Dreamweaver.)
4. Schneiden Sie ein ganzes Menü und den entsprechenden Inhalt ab dem öffnenden `menu`-Tag bis zum schließenden `/menu`-Tag aus.
5. Setzen Sie die Einfügemarke an die gewünschte Stelle für das Menü. (Stellen Sie sicher, dass sich die Einfügemarke zwischen einem `menubar`-Tag und dem zugehörigen `/menubar`-Tag befindet.)
6. Fügen Sie das Menü an der neuen Position ein.

Ändern des Namens eines Menübefehls oder Menüs

Sie können auf einfache Weise den Namen eines Menübefehls oder Menüs in der Datei „menus.xml“ ändern.

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 jedoch auf keinen Fall das `id`-Attribut.

5 Speichern und schließen Sie die Datei „menus.xml“. Starten Sie anschließend Dreamweaver neu, um die Änderungen anzuzeigen.

Ändern von Tastaturbefehlen

Wenn die Standardtastaturbefehle nicht Ihren Vorstellungen entsprechen, können Sie vorhandene Tastaturbefehle entfernen und neue hinzufügen. Diese Änderungen können Sie am einfachsten mit dem Tastaturbefehl-Editor vornehmen. (Weitere Informationen finden Sie in der Dreamweaver-Hilfe). Sie haben auch die Möglichkeit, Tastaturbefehle direkt in der Datei „menus.xml“ zu ändern. Hierbei können Ihnen jedoch leichter Fehler unterlaufen als bei Verwendung des Tastaturbefehl-Editors.

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 in diesem Fall den Menübefehl, dem der Tastaturbefehl zugeordnet ist, und entfernen Sie das Attribut `key="shortcut"` für diesen Menübefehl.

6 Suchen Sie den Menübefehl, dem der Tastaturbefehl zugewiesen oder neu zugewiesen werden soll.

7 Wenn für den Menübefehl bereits ein Tastaturbefehl vorhanden ist, suchen Sie das key-Attribut in der entsprechenden Zeile. Wenn noch kein Kurzbefehl vorhanden ist, fügen Sie `key=""` an einer beliebigen Stelle zwischen den Attributen im `menuitem`-Tag ein.

8 Geben Sie zwischen den 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 155.

   Wenn der Tastaturbefehl 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.

Popupmenüs und Kontextmenüs

Dreamweaver enthält in vielen Bedienfeldern und Dialogfeldern Popup- und Kontextmenüs. 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 programmieren.

Die folgenden Popupmenüs und Kontextmenüs in Dreamweaver sind in XML-Dateien definiert. Dabei werden die gleichen Tags wie in der Datei „menus.xml“ verwendet.

- Datenquellen (aufgeführt im Popupmenü mit dem Pluszeichen (+) des Bedienfelds „Bindungen“) sind in Dateien des Typs „DataSources.xml“ in Unterordnern des Ordners „DataSources“ definiert.
- Serververhalten (aufgeführt im Popupmenü mit dem Pluszeichen (+) des Bedienfelds „Serververhalten“) sind in Dateien des Typs „ServerBehaviors.xml“ in Unterordnern des Ordners „ServerBehaviors“ definiert.
- Serverformate (aufgeführt im Popupmenü mit dem Pluszeichen (+) des Dialogfelds „Formatliste bearbeiten“) sind in Dateien des Typs „ServerFormats.xml“ in Unterordnern des Ordners „ServerFormats“ definiert.
- Elemente im Popupmenü „Formate“ für eine Bindung im Bedienfeld „Bindungen“ sind in Dateien mit dem Namen „Formats.xml“ in Unterordnern des Ordners „ServerFormats“ definiert. Sie können diesem Menü in Dreamweaver über das Dialogfeld „Format hinzufügen“ Einträge hinzufügen.
- Menübefehle für das Dialogfeld „Tag-Bibliothek-Editor“ sind in der Datei „TagImporters.xml“ im Ordner „TagLibraries/TagImporters“ definiert.
- Menübefehle für Parameter im Dialogfeld „Verhalten generieren“ der Serververhalten-Erstellung sind in der Datei „Controls.xml“ im Ordner „Shared/Controls/String Menu“ definiert.
- Elemente für mit ColdFusion-Komponenten verknüpfte Kontextmenüs sind in der Datei „CFCsMenus.xml“ im Ordner „Components/ColdFusion/CFCs“ definiert.
- Elemente für mit ColdFusion-Datenquellen verknüpfte Kontextmenüs sind in der Datei „DataSourcesMenus.xml“ im Ordner „Components/ColdFusion/DataSources“ 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. Mit Menübefehlen können praktisch alle Bearbeitungsvorgänge am aktuellen Dokument, an anderen geöffneten Dokumenten oder an HTML-Dokumenten auf dem lokalen Laufwerk durchgeführt werden.

Menübefehle sind HTML-Dateien, die über das Attribut `file` eines `menuitem`-Tags 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, z. B. wie und nach welcher Spalte eine Tabelle sortiert werden soll. Der head-Bereich einer Menübefehlsdatei enthält JavaScript-Funktionen. Mit diesen Funktionen werden Formulareingaben aus dem body-Bereich verarbeitet und Änderungen am Dokument des Benutzers überprüft.

Menübefehle sind im Unterordner „Configuration/Menus“ des Anwendungsordners von Dreamweaver gespeichert.

Hinweis: *Befehle unter Mac OS X 10.3 sind unter „<Benutzername>/Library/Application Support/Adobe/Dreamweaver 9/Commands/<Befehlsname>.html“ gespeichert.*

In der folgenden Tabelle sind die Dateien aufgeführt, mit denen Sie Menübefehle erstellen.

Pfad	Datei	Beschreibung
Configuration/Menus/	menus.xml	Enthält eine strukturierte Liste der Menüleisten, Menüs, Menübefehle, Trennzeichen, Kurzbefehle und Tastaturbefehle. 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 benutzerdefinierte Menübefehle hinzufügen, sollten Sie diese auf der obersten Ebene des Ordners „Menus“ oder in einem eigens dafür erstellten Unterordner einfügen. Der Ordner „MM“ ist für die mit Dreamweaver gelieferten Menübefehle reserviert.*

Ändern des Menüs „Befehle“

Sie können im Menü „Befehle“ bestimmte Befehle hinzufügen und die entsprechenden Namen ändern, ohne dabei die Datei „menus.xml“ bearbeiten zu müssen. Weitere Informationen zur Datei „menus.xml“ finden Sie unter „Ändern von Menüs und Menübefehlen“ auf Seite 161.

Hinweis: *Der Begriff „Befehl“ hat in Dreamweaver zwei Bedeutungen. Genau genommen ist ein Befehl eine besondere Art von Erweiterung. In bestimmten Kontexten hingegen wird er synonym für den Begriff „Menüelement“ verwendet. Darunter fallen alle Elemente in einem Dreamweaver-Menü, ungeachtet der Funktions- und Implementierungsweise.*

Im Bedienfeld „Verlauf“ können Sie neue Befehle erstellen, die automatisch in das Menü „Befehle“ eingefügt werden. 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.

Umbenennen eines selbst erstellten Befehls

1. Wählen Sie „Befehle“ > „Befehlsliste bearbeiten“ aus.

   Ein Dialogfeld mit allen Befehlen, deren Namen Sie ändern können, wird angezeigt. (Befehle des Standardmenüs „Befehle“ werden in dieser Liste nicht angezeigt und können auf diese Weise nicht bearbeitet werden.)

2. Wählen Sie den umzubenennenden Befehl aus.
3. Geben Sie einen neuen Namen ein.
4. Klicken Sie auf „Schließen“.

Damit wird dem Befehl im Menü „Befehle“ der neue Name zugewiesen.

Löschen eines selbst erstellten Befehls

1. Wählen Sie „Befehle“ > „Befehlsliste bearbeiten“ aus.

   Ein Dialogfeld mit allen Befehlen, die gelöscht werden können, wird angezeigt. (Befehle des Standardmenüs „Befehle“ werden in dieser Liste nicht angezeigt und können auf diese Weise nicht gelöscht werden.)

2. Wählen Sie den zu löschenden Befehl aus.
3. Klicken Sie auf „Löschen“ und bestätigen Sie dann den Löschvorgang.

Der Befehl wird gelöscht. Die Datei mit dem Code für den Befehl wird ebenfalls gelöscht. Durch das Löschen eines Befehls wird daher nicht nur der Menübefehl aus dem Menü entfernt. Vergewissern Sie sich, dass Sie den Befehl wirklich löschen möchten, bevor Sie dieses Verfahren durchführen. 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 sie in einen anderen Ordner.

4 Klicken Sie auf „Schließen“.

Funktionsweise von Befehlen

Wenn der Benutzer auf ein Menü klickt, das ein Menüelement mit einem verknüpftem Menübefehl enthält, werden folgende Vorgänge durchgeführt:

1 Wenn 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 Für jede mit dem Menü verknüpfte Menübefehlsdatei wird die Funktion `canAcceptCommand()` aufgerufen, um zu prüfen, ob der Befehl für die aktuelle Auswahl zulässig ist.

- Wenn die Funktion `canAcceptCommand()` den Wert `false` zurückgibt, wird das Menüelement abgeblendet angezeigt.
- Wenn die Funktion `canAcceptCommand()` den Wert `true` zurückgibt oder nicht definiert ist, wird die Funktion `isCommandChecked()` aufgerufen, um festzulegen, 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 den Text festzulegen, der 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 Wenn sie definiert ist, wird die Funktion `receiveArguments()` für die ausgewählte Menübefehlsdatei aufgerufen, damit der Befehl die vom Menüelement übergebenen Argumente verarbeiten kann.

Hinweis: *Bei dynamischen Menüelementen wird als einziges Argument die ID des Menüelements übergeben.*

6 Wenn sie definiert ist, wird die Funktion `commandButtons()` aufgerufen, um zu prüfen, welche Schaltflächen auf der rechten Seite des Dialogfelds „Optionen“ angezeigt werden sollen und welcher Code beim Klicken auf die Schaltflächen jeweils ausgeführt werden soll.

7 Die Menübefehlsdatei wird nach dem `form`-Tag durchsucht.

- Wenn ein form-Tag vorhanden ist, wird die Funktion `windowDimensions()` aufgerufen, um die Größe des Dialogfelds „Optionen“ festzulegen, das die `BODY`-Elemente der Datei enthält.
- Wenn die Funktion `windowDimensions()` nicht definiert ist, wird die Größe des Dialogfelds automatisch angepasst.

8 Wenn das `body`-Tag der Menübefehlsdatei eine Ereignisprozedur vom Typ `onLoad` enthält, wird der entsprechende Code in 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. Die den einzelnen Feldern zugeordneten Ereignisprozeduren werden in Dreamweaver ausgeführt, wenn die Felder 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 wird angezeigt, bis eines der Skripts in der Menübefehlsdatei die Funktion `window.close()` aufruft.

Beispiel für einen einfachen Menübefehl

In diesem leicht nachvollbaren Beispiel wird die Funktionsweise der Menübefehle „Rückgängig“ und „Wiederherstellen“ beschrieben. Mit dem Menübefehl „Rückgängig“ wird ein Bearbeitungsschritt eines Benutzers rückgängig gemacht. Mit „Wiederherstellen“ wird der Vorgang „Rückgängig“ aufgehoben und der letzte Bearbeitungsschritt wiederhergestellt.

In diesem Beispiel erstellen Sie die Menübefehle, programmieren den JavaScript-Code und fügen die Befehlsdatei im Ordner „Menu“ ein.

Erstellen eines Menübefehls

Um ein Menü mit dem Namen „MyMenu“ mit den Menübefehlen für „Rückgängig“ und „Wiederherstellen“ zu erstellen, fügen Sie die folgenden menu-HTML-Tags ein. Fügen Sie die Tags nach dem letzten schließenden `</menu>`-Tag in „menus.xml“ ein.

```
<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 den Menübefehl zu laden. Mit dem Attribut `file` wird der Name der Menübefehlsdatei definiert, die Dreamweaver beim Laden des Befehls ausführt. Der Wert des Attributs `arguments` definiert die Argumente, die Dreamweaver beim Aufrufen der Funktion `receiveArguments()` übergibt.

In der folgenden Abbildung sind diese Menübefehle dargestellt:

JavaScript-Code für das Menü

Wenn der Benutzer im Menü „MyMenu“ das Menüelement „Rückgängig“ oder „Wiederherstellen“ auswählt, ruft Dreamweaver die Befehlsdatei „MyMenu.htm“ auf, die durch das Attribut `file` des Tags `menuitem` definiert ist. Erstellen Sie die Befehlsdatei „MyMenu.htm“ im Dreamweaver-Ordner „Configuration/Menus“. Fügen Sie dann für den Menübefehl die drei API-Funktionen `canAcceptCommand()`, `receiveArguments()` und `setMenuText()` 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 festzulegen, ob es jeweils aktiviert oder deaktiviert werden soll. Die Funktion `canAcceptCommand()` überprüft in der Datei „MyMenu.htm“ den Wert des Arguments `arguments[0]`, um zu ermitteln, ob das Menüelement „Rückgängig“ oder „Wiederherstellen“ verarbeitet wird. Wenn das Argument `"undo"` ist, ruft die Funktion `canAcceptCommand()` die Enabler-Funktion `dw.canUndo()` auf und gibt den zurückgegebenen Wert zurück, der entweder `true` oder `false` lautet. Wenn das Argument `"redo"` ist, ruft die Funktion `canAcceptCommand()` die Enabler-Funktion `dw.canRedo()` auf und gibt den entsprechenden Wert an Dreamweaver zurück. Wenn die Funktion `canAcceptCommand()` den Wert `false` zurückgibt, blendet Dreamweaver das Menüelement ab, für das die Funktion aufgerufen wurde. Im folgenden Beispiel ist der Code für die Funktion `canAcceptCommand()` angegeben:

```
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 die für das `menuitem`-Tag definierten Argumente zu verarbeiten. Für die Menüelemente „Rückgängig“ und „Wiederherstellen“ ruft die Funktion `receiveArguments()` die Funktion `dw.undo()` oder `dw.redo()` auf. Dies hängt davon ab, ob der Wert des Arguments `arguments[0]` den Wert `"undo"` oder `"redo"` hat. Mit der Funktion `dw.undo()` wird der vorherige Schritt des Benutzers im aktiven Dokumentfenster, Dialogfeld oder Bedienfeld rückgängig gemacht. Mit der Funktion `dw.redo()` wird der zuletzt rückgängig gemachte Vorgang wiederhergestellt.

Es folgt 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 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()` mit dem zweiten Argument als Parameter aufgerufen. Wenn das erste Argument `"bar"` lautet, wird die Funktion `doOperationY()` mit dem zweiten Argument als Parameter aufgerufen. 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"){
        doOperationX(arguments[1]);
    }
}
```

setMenuText()

Dreamweaver ruft die Funktion `setMenuText()` auf, um den für das Menüelement angezeigten Text festzulegen. Wenn die Funktion `setMenuText()` nicht definiert ist, verwendet Dreamweaver den im `name`-Attribut des `menuitem`-Tags angegebenen Text.

Die Funktion `setMenuText()` überprüft den von Dreamweaver übergebenen Wert des Arguments `arguments[0]`. Wenn der Wert des Arguments `"undo"` ist, ruft Dreamweaver die Funktion. `dw.getUndoText()` auf. Wenn der Wert `"redo"` lautet, ruft Dreamweaver die Funktion `dw.getRedoText()` auf. Mit der Funktion `dw.getUndoText()` wird der Text zu dem rückgängig gemachten Vorgang angezeigt. Wenn der Benutzer mehrere Vorgänge zum Wiederherstellen durchführt, gibt die Funktion `dw.getUndoText()` beispielsweise den Text „Bearbeiten der Quelle rückgängig machen“ zurück. Entsprechend gibt die Funktion `dw.getRedoText()` den Text zu dem wiederhergestellten Schritt zurück. Wenn der Benutzer mehrere Aktionen zum Rückgängigmachen ausführt, gibt die Funktion `dw.RedoText()` beispielsweise den Text „Bearbeiten der Quelle wiederherstellen“ zurück.

Es folgt 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 "";
}
```

Einfügen der Befehlsdatei im Ordner „Menus“

Zum Implementieren der Menüelemente „Rückgängig“ und „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 diese Datei zugreifen zu können, müssen Sie Dreamweaver neu starten oder die Erweiterungen neu laden. Informationen zum Neuladen von Erweiterungen finden Sie unter „Erneutes Laden von Erweiterungen“ auf Seite 84.

Wählen Sie zum Ausführen der Menübefehle das aktivierte Menüelement aus. Es werden die Funktionen in der Befehlsdatei aufgerufen (siehe dazu „Funktionsweise von Befehlen“ auf Seite 165).

Beispiel für dynamische Menüs

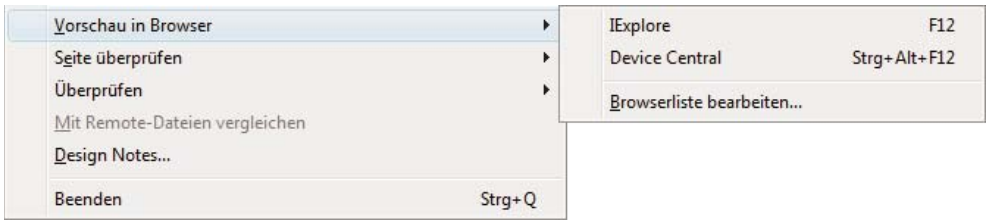
In diesem Beispiel wird das Dreamweaver-Untermenü „Vorschau in Browser“ implementiert, in dem die verfügbaren Browser aufgelistet sind. Zudem werden im benutzerdefinierten Browser die aktuelle Datei oder die im Bedienfeld „Dateien“ ausgewählten Dateien angezeigt. Zum Implementieren dieses dynamischen Menüs erstellen Sie die dynamischen Menüelemente und programmieren den JavaScript-Code.

Erstellen dynamischer Menüelemente

Mit den folgenden menu-Tags in der Datei „menus.xml“ wird das Untermenü „Vorschau in Browser“ des Menüs „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 im Dialogfeld „Voreinstellungen“ für das Element „Vorschau in Browser“ keinen Browser angegeben haben. Wenn Sie jedoch als Browser Microsoft Internet Explorer angeben, sieht das Untermenü folgendermaßen aus:



Im Attribut `name` des ersten Menüelements wird die Befehlsdatei „PIB_Dynamic.htm“ angegeben. Diese Datei enthält die folgende Zeile:

```
<SCRIPT LANGUAGE="javascript" SRC="PIB_Dynamic.js"></SCRIPT>
```

Das `script`-Tag enthält in der Datei „PIB_Dynamic.js“ den JavaScript-Code, 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 eingefügt werden.

JavaScript-Code für das dynamische Menüelement

Da das erste `menuitem`-Tag das Attribut `dynamic` enthält, ruft Dreamweaver die Funktion `getDynamicContent()` in der Datei „PIB_Dynamic.js“ auf. Siehe dazu das folgende Beispiel:

```
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 Bereich „Vorschau in Browser“ des Dialogfelds „Voreinstellungen“ angegebenen Browsernamen abzurufen. Das Array enthält den Namen der einzelnen Browser und den Pfad zu den ausführbaren Dateien. Die Funktion `getDynamicContents()` verschiebt dann 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]);`). Wenn der Browser als Primär- oder Sekundärbrowser festgelegt wurde, fügt die Funktion die Namen der Tastaturbefehle an, mit denen die Browser aufgerufen werden. Dann wird der String `";id="` gefolgt vom Browsernamen in einfachen Anführungszeichen angefügt (z. B. `;id='iexplore'`). Wenn das `itemID`-Argument `"DWPopup_PIB_Default"` lautet, setzt die Funktion den String `Vorschau in` vor das Array-Element. Nachdem ein Eintrag für jeden in den Voreinstellungen aufgeführten Browser erstellt wurde, gibt die Funktion `getDynamicContent()` das Array `browsers` an Dreamweaver zurück. Wenn keine Browser ausgewählt wurden, gibt die Funktion den Wert `null` zurück und im Menü wird die Option „Keine Browser ausgewählt“ angezeigt.

canAcceptCommand()

Dreamweaver ruft anschließend die Funktion `canAcceptCommand()` für jedes `menuitem`-Tag auf, das mit dem `file`-Attribut auf eine Befehlsdatei verweist. Wenn die Funktion `canAcceptCommand()` den Wert `false` zurückgibt, wird das Menüelement abgeblendet angezeigt. Wenn die Funktion `canAcceptCommand()` den Wert `true` zurückgibt, wird das Element im Menü aktiviert. Wenn die Funktion den Wert `true` zurückgibt oder nicht definiert ist, wird die Funktion `isCommandChecked()` aufgerufen, um zu ermitteln, 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 wird überprüft, ob das erste Argument (`arguments[0]`) den Wert „primary“ oder „secondary“ hat. 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`). Wenn die Bedingungen für *beide* Tests erfüllt sind, gibt die Funktion den Wert `true` zurück. Wenn die Bedingungen für mindestens einen Test nicht erfüllt sind, 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];

                    // For server connections, the files will
                    // already be 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 `bHavePreviewTarget` fest. Die Funktion führt durch Aufrufen der Funktion `dw.getFocus()` zwei grundlegende Tests durch, um zu ermitteln, welcher Teil der Anwendung derzeit den Eingabefokus hat. Beim ersten Test wird geprüft, ob das Bedienfeld „Dateien“ aktiv ist (if (dw.getFocus(true) == 'site')). Wenn das Bedienfeld „Dateien“ nicht aktiv ist, 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 nicht zutreffen, wird der Wert `false` zurückgegeben.

Wenn das Bedienfeld „Dateien“ aktiv ist, prüft die Funktion, ob die Einstellung für die Ansicht „Remote-Ansicht“ lautet. Ist dies der Fall, setzt die Funktion den Wert `bHavePreviewTarget` auf `true`, wenn Remote-Dateien (site.getRemoteSelection().length > 0) vorhanden sind und 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();) im URL-Format `file://` ab.

Die Funktion prüft für jedes Element in der ausgewählten Liste, ob der String "`://`" vorhanden ist. Wenn er nicht gefunden wird, werden im Code mehrere Tests für das Listenelement durchgeführt. Wenn für das Element nicht das URL-Format `file://` gilt (if (selFile.indexOf(urlPrefix) == -1)), wird der Rückgabewert auf `false` gesetzt. Wenn der restliche Teil des Strings nach dem Präfix `file://` keinen Schrägstrich (/) enthält (if (strTemp.indexOf("/") == -1)), wird der Rückgabewert auf `false` gesetzt. Wenn die Datei nicht vorhanden ist (else if (!DWfile.exists(selFile))), wird der Rückgabewert auf `false` gesetzt. 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. Wenn es sich beim Ziel um eine Datei handelt, setzt die Funktion den Wert `bHavePreviewTarget` auf `true`.

Wenn ein Dokument, die Textansicht oder der Codeinspektor der Eingabefokus hat (`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 `bHavePreviewTarget` gespeicherten Wert zurück.

receiveArguments()

Dreamweaver ruft die Funktion `receiveArguments()` auf, damit der Befehl alle vom Menüelement übergebenen Argumente verarbeiten kann. Für das Menü „Vorschau in Browser“ ruft die Funktion `receiveArguments()` beispielsweise den vom Benutzer ausgewählten Browser auf. Die Funktion `receiveArguments()` sieht wie folgt aus:

```
function receiveArguments()
{
    var whichBrowser = arguments[0];
    var theBrowser = null;
    var i=0;
    var browserList = null;
    var result = false;

    if (havePreviewTarget())
    {
        // Code to check if we were called from a shortcut key
        if (whichBrowser == 'primary' || whichBrowser == 'secondary')
        {
            // get the path of the selected browser
            if (whichBrowser == 'primary')
            {
                theBrowser = dw.getPrimaryBrowser();
            }
            else if (whichBrowser == 'secondary')
            {
                theBrowser = dw.getSecondaryBrowser();
            }

            // Match the path with the name of the corresponding browser
            // that appears in the menu.
            browserList = dw.getBrowserList();
            while(i < browserList.length)
            {
                if (browserList[i+1] == theBrowser)
                    theBrowser = browserList[i];
                i+=2;
            }
        }
        else
            theBrowser = whichBrowser;
        // Only launch the browser if we have a valid browser selected.
        if (theBrowser != "file:///" && typeof(theBrowser) != "undefined" &&
        theBrowser.length > 0)
        {
            if (dw.getFocus(true) == 'site')
            {
                // Only get the first item of the selection because
                // browseDocument() can't take an array.
                //dw.browseDocument(site.getSelection()[0],theBrowser);
```

```
                site.browseDocument(theBrowser);
            }
            else
                dw.browseDocument(dw.getDocumentPath('document'),theBrowser);
        }
        else
        {
            // Otherwise, F12 or Ctrl+F12 was pressed, ask if the user wants
            // to specify a primary or secondary browser now.
            if (whichBrowser == 'primary')
            {
                result = window.confirm(MSG_NoPrimaryBrowserDefined);
            }
            else if (whichBrowser == 'secondary')
            {
                result = window.confirm(MSG_NoSecondaryBrowserDefined);
            }

            // If the user clicked OK, show the prefs dialog with the browser panel.
            if (result)
                dw.showPreferencesDialog('browsers');
        }
    }
}
```

Die Funktion setzt zunächst die Variable `whichBrowser` auf den von Dreamweaver übergebenen Wert `arguments[0]`. Neben dem Festlegen anderer Standardwerte wird `result` auf den Standardwert `false` gesetzt.

Nach dem Initialisieren der Variablen ruft die Funktion `receiveArguments()` die benutzerdefinierte Funktion `havePreviewTarget()` auf und überprüft das Ergebnis. Wenn das Ergebnis des Tests „true“ ist, überprüft die Funktion, ob der Benutzer den Primär- bzw. Sekundärbrowser ausgewählt hat. Ist dies der Fall, wird der Variable `theBrowser` der Pfad der ausführbaren Datei zum Starten des Browsers (`dw.getPrimaryBrowser()` oder `dw.getSecondaryBrowser()`) zugewiesen. Die Funktion führt anschließend eine Schleife aus, mit der die von `dw.getBrowsersList()` zurückgegebene Browserliste überprüft wird. Wenn ein Pfad zum Browser in der Liste mit dem Primär- oder Sekundärbrowser übereinstimmt, 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 alle Bedingungen erfüllt sind und der Fokus auf dem Bedienfeld „Dateien“ liegt, ruft die Funktion `receiveArguments()` die Funktion `site.browseDocument()` auf, um den ausgewählten Browser mit den im Bedienfeld „Dateien“ ausgewählten Dateien aufzurufen. Wenn der Fokus nicht auf dem Bedienfeld „Dateien“ liegt, 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 einen Browser angeben kann.

API-Funktionen für Menübefehle

Die benutzerdefinierten Funktionen der API für Menübefehle sind nicht zwingend erforderlich.

canAcceptCommand()

Beschreibung
Legt fest, ob das Menüelement aktiviert oder abgeblendet ist.

Argumente
{arg1}, {arg2},...{argN}}

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige ID als einziges Argument übergeben. Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert als mindestens ein Argument an die Funktion `canAcceptCommand()` (sowie an die Funktionen „isCommandChecked()“ auf Seite 176, „receiveArguments()“ auf Seite 177 und „setMenuText()“ auf Seite 178) 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, andernfalls `false`.

commandButtons()

Beschreibung
Definiert die Schaltflächen, die rechts im Dialogfeld „Optionen“ angezeigt werden, sowie das Verhalten, wenn auf die Schaltflächen geklickt wird. Wenn diese Funktion nicht definiert ist, werden keine Schaltflächen angezeigt. Stattdessen wird der body-Bereich der Menübefehlsdatei im gesamten Dialogfeld angezeigt.

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 Verhalten beim Klicken auf die oberste Schaltfläche festgelegt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel
Im folgenden Beispiel werden mit der Funktion `commandButtons()` die Schaltflächen „OK“, „Abbrechen“ und „Hilfe“ definiert.

```
function commandButtons(){
    return new Array("OK" , "doCommand()" , "Cancel" , ¬
    "window.close()" , "Help" , "showHelp()");
}
```

getDynamicContent()

Beschreibung
Ruft den Inhalt des dynamischen Bereichs 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, in dem jeder String den Namen eines Menüelements und dessen eindeutige ID enthält. Die einzelnen Elemente sind jeweils durch ein Semikolon getrennt. Wenn die Funktion den Wert `null` zurückgibt, ändert sich das Menü nicht.

Beispiel
Im folgenden Beispiel gibt die Funktion `getDynamicContent()` ein Array mit vier Menüelementen („My Menu Item 1“, „My Menu Item 2“, „My Menu Item 3“ und „My Menu Item 4“) zurück.

```
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
Legt fest, ob neben dem Menüelement ein Häkchen angezeigt werden soll.

Argumente
{arg1}, {arg2},...{argN}

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige ID als einziges Argument übergeben. Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert als mindestens ein Argument an die Funktion `isCommandChecked()` (sowie an die Funktionen „canAcceptCommand()“ auf Seite 175, „receiveArguments()“ auf Seite 177 und „setMenuText()“ auf Seite 178) 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, 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 alle Argumente, die von einem Menüelement oder von der Funktion `dw.runCommand()` übergeben werden. Bei dynamischen Menüelementen wird die ID des dynamischen Menüelements verarbeitet.

Argumente

{arg1}, {arg2},...{argN}

Bei dynamischen Menüelementen wird die in der Funktion `getDynamicContents()` definierte eindeutige ID als einziges Argument übergeben. Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert als mindestens ein Argument an die Funktion `receiveArguments()` (sowie an die Funktionen „canAcceptCommand()“ auf Seite 175, „isCommandChecked()“ auf Seite 176 und „setMenuText()“ auf Seite 178) 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 keine Rückgabewerte.

Beispiel

```
function receiveArguments()
{
    var styleName = arguments[0];
    if (styleName == "(None)")
        dw.getDocumentDOM('document').applyCSSStyle('','');
    else
        dw.getDocumentDOM('document').applyCSSStyle('',styleName);
}
```

setMenuText()

Beschreibung
Legt den Text fest, der im Menü angezeigt werden soll.

Hinweis: *Setzen Sie diese Funktion nicht ein, wenn Sie „getDynamicContent()“ auf Seite 176 verwenden.*

Argumente
{arg1}, {arg2},...{argN}

Wenn das Attribut `arguments` für ein Tag des Typs `menuitem` definiert ist, wird der Attributwert als mindestens ein Argument an die Funktion `setMenuText()` (sowie an die Funktionen „canAcceptCommand()“ auf Seite 175, „isCommandChecked()“ auf Seite 176 und „receiveArguments()“ auf Seite 177) 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, wenn das Dialogfeld größer als 640 x 480 Pixel ist.*

Argumente
platform

Der Wert des Arguments *platform* ist entweder `"macintosh"` oder `"windows"`, abhängig von der Plattform des Benutzers.

Rückgabewerte
Dreamweaver erwartet einen String des Typs `"widthInPixels,heightInPixels"`.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da der Bereich für die Schaltflächen „OK“ und „Abbrechen“ nicht einbezogen ist. Wenn im Fenster mit den zurückgegebenen Abmessungen nicht alle Optionen angezeigt werden können, werden Bildlaufleisten eingeblendet.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Dialogfelds „Parameter“ auf 648 x 520 Pixel gesetzt:

```
function windowDimensions(){
    return "648,520";
}
```

Kapitel 11: Symbolleisten

Eine Symbolleiste für Adobe Dreamweaver können Sie erstellen, indem Sie einfach eine Datei erstellen, in der die Symbolleiste definiert wird, und diese Datei im Ordner „Configuration/Toolbars“ speichern.

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	Erstellen Sie diese Datei, um eine neue Symbolleiste zu erstellen.
Configuration/Toolbars/	*Bilddatei*.gif	Bild für das Symbolleistensteuerelement.
Configuration/Commands/	MyCommand.htm	Befehlsdatei, die mit dem Symbolleistenelement verknüpft ist.

Funktionsweise von Symbolleisten

Symbolleisten werden durch XML-Dateien und Bilddateien definiert, die im Unterordner „Toolbars“ des Dreamweaver-Hauptordners „Configuration“ gespeichert sind. Die Standardsymbolleisten von Dreamweaver sind in der Datei „toolbars.xml“ im Ordner „Configuration/Toolbars“ gespeichert. Wenn Dreamweaver gestartet wird, werden alle Symbolleistendateien geladen, die im Ordner „Toolbars“ gespeichert sind. Um neue Symbolleisten hinzuzufügen, müssen Sie nicht die Datei „toolbars.xml“ ändern, sondern können einfach eine Datei in den Ordner „Toolbars“ kopieren.

XML-Dateien für Symbolleisten definieren eine oder mehrere Symbolleisten und die zugehörigen Symbolleistenelemente. Eine Symbolleiste ist eine Liste von Elementen, z. B. Schaltflächen, Textfelder, Popupmenüs usw. Ein Symbolleistenelement stellt ein einzelnes Steuerelement dar, auf das Benutzer in einer Symbolleiste zugreifen können.

Mit einigen Typen von Symbolleistenelementen (z. B. Schaltflächen und Popupmenüs) sind Symbolbilder verknüpft. Symbolbilder werden im Ordner „Toolbars/images“ gespeichert. Die Bilder können jedes Format aufweisen, das in Dreamweaver dargestellt werden kann. Normalerweise handelt es sich jedoch um 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 für Symbolleisten sind im Ordner „Toolbars/MM“ gespeichert.

Da die API für Symbolleistenbefehle kompatibel ist mit der API für Menübefehle, können Symbolleistenelemente die Menübefehlsdateien wiederverwenden.

Im Gegensatz zu Menüs können Symbolleistenelemente unabhängig von den Symbolleisten definiert werden, in denen sie verwendet werden. Durch diese Flexibilität können Sie Symbolleistenelemente mithilfe des `itemref`-Tags in mehreren Symbolleisten verwenden.

Wenn eine Symbolleiste zum ersten Mal in Dreamweaver geladen wird, werden ihre Sichtbarkeit und Position durch die Symbolleistendefinition festgelegt. Dann werden Sichtbarkeit und Position in der Registrierung (Windows) bzw. in der Datei für Dreamweaver-Voreinstellungen (Macintosh) gespeichert und von dort aus wiederhergestellt.

Verhaltensweisen von Symbolleisten

Unter Windows verhalten sich Dreamweaver-Symbolleisten im Allgemeinen genau wie Windows-Symbolleisten. Dreamweaver-Symbolleisten zeichnen sich durch folgende Merkmale aus:

- 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.
- Symbolleisten behalten ihre feste Größe bei. Eine Symbolleiste wird nicht verkleinert, wenn ihr Container verkleinert wird oder andere Symbolleisten neben ihr platziert werden.
- Sie können Symbolleisten über das Menü „Ansicht“ > „Symbolleisten“ ein- oder ausblenden.
- Symbolleisten können einander nicht überlappen.
- Wenn Sie eine Symbolleiste ziehen, wird nur ihr Umriss angezeigt.

Auf Macintosh-Computern sind Symbolleisten immer mit dem Dokumentfenster verknüpft. Sie können im Menü ein- und ausgeblendet, jedoch nicht gezogen und abgelegt, neu angeordnet oder abgedockt werden.

Im Dreamweaver-Arbeitsbereich, in dem alle Dreamweaver-Dokumentfenster in einem einzigen übergeordneten Frame integriert sind, können Sie festlegen, ob Symbolleisten am Arbeitsbereichsfenster oder am Dokumentfenster andocken.

Für Symbolleisten, die am Dreamweaver-Arbeitsbereichsfenster andocken, gibt es jeweils nur eine Instanz. In diesem Fall werden die Symbolleisten für das Dokument immer im Vordergrund angezeigt. Im Dreamweaver-Arbeitsbereich können Sie Symbolleisten über, unter, links oder rechts der Einfügeleiste andocken. Symbolleisten, die mit dem Dreamweaver-Arbeitsbereichsfenster verknüpft sind, werden nicht automatisch deaktiviert, wenn kein Dokumentfenster vorhanden ist. Die Symbolleistenelemente legen fest, ob sie aktiviert sind, wenn kein Dokument geöffnet ist.

Wenn Symbolleisten immer an das Dokumentfenster angedockt sind, gibt es für jedes Fenster jeweils eine Symbolleiste. Symbolleisten, die mit einem Dokumentfenster verknüpft sind, deaktivieren sich selbst, wenn das entsprechende Fenster nicht mehr im Vordergrund angezeigt wird, und führen ihre Aktualisierungsprozeduren erneut aus, wenn das Fenster wieder im Vordergrund angezeigt wird.

Symbolleisten können nicht durch Ziehen und Ablegen zwischen dem Dokumentfenster und dem Dreamweaver-Arbeitsbereichsfenster verschoben werden.

Funktionsweise von Symbolleistenbefehlen

Wenn Dreamweaver eine Symbolleiste darstellt, werden folgende Aktionen ausgeführt:

1. Dreamweaver prüft für jedes Symbolleistenelement, ob das Attribut `file` vorhanden ist.
2. Wenn das Attribut `file` vorhanden ist, ruft Dreamweaver die Funktion `canAcceptCommand()` auf, um zu ermitteln, ob das Steuerelement im aktuellen Kontext des Dokuments aktiviert werden soll.

   Für das Textfeld „Dokumenttitel“ der Dreamweaver-Symbolleiste prüft die Funktion `canAcceptCommand()` beispielsweise, ob ein aktuelles DOM vorhanden ist und ob das aktuelle Dokument eine HTML-Datei ist. Wenn 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 gegebenenfalls angegebenen Attribute: `checked`, `command`, `DOMRequired`, `enabled`, `script`, `showif`, `update` und `value`.
4. Wenn das Attribut `file` nicht vorhanden ist, verarbeitet Dreamweaver die Attribute, die für das Symbolleistenelement festgelegt sind: `checked`, `command`, `DomRequired` usw.

Weitere Informationen zu bestimmten Attributen für Element-Tags finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 194.

5 Dreamweaver ruft die Funktion `getCurrentValue()` bei jedem Aktualisierungszyklus auf, wie dies durch das Attribut `update` festgelegt ist, um den für das Steuerelement anzuzeigenden Wert zu ermitteln.

6 Der Benutzer wählt ein Element auf der Symbolleiste aus.

7 Dreamweaver ruft die Funktion `receiveArguments()` auf, um alle Argumente zu verarbeiten, die durch das Attribut `arguments` des Symbolleistenelements angegeben sind.

Weitere Informationen zu den Funktionen in der API für Symbolleistenbefehle finden Sie unter „API-Funktionen für Symbolleistenbefehle“ auf Seite 199.

Einfache Befehlsdatei für eine Symbolleiste

Bei diesem einfachen Beispiel wird das Textfeldelement „Titel“ implementiert, wie es auf der Dokumentsymbolleiste von Dreamweaver angezeigt wird. Im Textfeldelement kann der Benutzer einen Namen für das aktuelle Dreamweaver-Dokument eingeben. Sie können dieses Symbolleistenbeispiel mit den folgenden Schritten implementieren.

Erstellen des Textfelds

Sie können in Dreamweaver eine Symbolleiste hinzufügen, indem Sie eine XML-Datei mit der Symbolleistendefinition im Ordner „Toolbars“ im Dreamweaver-Ordner „Configuration“ einfügen.

In der folgenden Abbildung ist das Textfeld „Titel“ dargestellt:

Das folgende `editcontrol`-Symbolleistenelement definiert ein Textfeld 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 mit dem Mauszeiger auf das Textfeld zeigt. Das Attribut `width` definiert die Größe des Felds in Pixel. Das Attribut `file` gibt an, dass die Datei „EditTitle.htm“ die JavaScript-Funktionen enthält, die auf das Textfeld angewendet werden. Die vollständige Definition der Dreamweaver-Dokumentsymbolleiste finden Sie in der Datei „toolbars.xml“ in der Deklaration der Hauptsymbolleiste (`id="DW_Toolbar_Main"`).

JavaScript-Code für das Textfeld

Wenn der Benutzer mit dem Textfeld interagiert, ruft Dreamweaver die Befehlsdatei „EditTitle.htm“ im Ordner „Toolbars/MM“ auf. Diese Datei enthält drei JavaScript-Funktionen, die mit dem Textfeld „Titel“ verwendet werden können. Dies sind `canAcceptCommand()`, `receiveArguments()` und `getCurrentValue()`.

canAcceptCommand(): Aktivieren des Symbolleistenelements

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 der Wert `true` zurückgegeben wird, 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(): Festlegen des Titels

Dreamweaver ruft die im Folgenden dargestellte Funktion `receiveArguments()` auf, wenn der Benutzer einen Wert im Textfeld „Titel“ eingibt und die Eingabetaste drückt oder wenn der Fokus nicht mehr auf dem Steuerelement liegt.

Die Funktion lautet:

```
function receiveArguments(newTitle)
{
    var dom = dw.getDocumentDOM();
    if (dom)
        dom.setTitle(newTitle);
}
```

Dreamweaver übergibt `newTitle`, d. h. den vom Benutzer eingegebenen Wert, an die Funktion `receiveArguments()`. Die Funktion `receiveArguments()` prüft zunächst, ob ein aktuelles DOM vorhanden ist. Ist dies der Fall, legt die Funktion `receiveArguments()` den neuen Dokumenttitel fest, indem sie `newTitle` an die Funktion `dom.setTitle()` übergibt.

getCurrentValue(): Abrufen des Titels

Bei jedem Aktualisierungszyklus – wie durch die standardmäßige Aktualisierungshäufigkeit der Ereignisprozedur `onEdit` festgelegt – ruft Dreamweaver die Funktion `getCurrentValue()` auf, um zu ermitteln, welcher Wert für das Steuerelement angezeigt werden soll. Die Aktualisierungshäufigkeit wird durch den Standardwert der Verarbeitungsprozedur `onEdit` festgelegt, da das Textfeldelement „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;
}
```

Bis der Benutzer einen 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()` den Wert für diesen Titel zurück. Dreamweaver zeigt den Wert dann als neuen Dokumenttitel an.

Die vollständige HTML-Datei mit den JavaScript-Funktionen für das Textfeld „Titel“ finden Sie in der Datei „EditTitle.htm“ im Ordner „Toolbars/MM“.

Der Ordner „MM“ ist für Adobe-Dateien reserviert. Erstellen Sie einen anderen Ordner im Ordner „Toolbars“ und speichern Sie Ihren JavaScript-Code in diesem Ordner.

Definitionsdatei für Symbolleisten

Bei einer Symbolleiste handelt es sich um eine Auflistung von Optionsschaltern, Kontrollkästchen, Bearbeitungsfeldern und anderen Symbolleistenelementen, die optional durch `separator`-Tags getrennt sind. Jedes Symbolleistenelement kann entweder ein Verweis auf ein Element (mit dem `itemref`-Tag), eine Trennlinie (mit dem `separator`-Tag) oder eine vollständige Definition des Symbolleistenelements (z. B. für ein Kontrollkästchen oder ein Bearbeitungsfeld) sein. Eine Beschreibung finden Sie unter „Symbolleistenelement-Tags“ auf Seite 188.

Jede Definitionsdatei für Symbolleisten beginnt mit den folgenden Deklarationen:

```
<?xml version="1.0" encoding="optional_encoding"?>
<!DOCTYPE toolbarset SYSTEM "-//Macromedia//DWExtension toolbar 5.0">
```

Wenn die Kodierung nicht angegeben wird, verwendet Dreamweaver die Standardkodierung des Betriebssystems.

Im Anschluss an die Deklarationen besteht die Datei aus einem einzigen `toolbarset`-Tag. Dieses wiederum umfasst eine beliebige Anzahl der folgenden Tags: `toolbar`, `itemref`, `separator`, `include` und *itemtype*. Dabei kann das *itemtype*-Tag einen der folgenden Werte haben: `button`, `checkbutton`, `radiobutton`, `menubutton`, `dropdown`, `combobox`, `editcontrol` oder `colorpicker`. Im folgenden Beispiel, einem gekürzten Auszug aus der Datei „toolbars.xml“, wird die Hierarchie der Tags in der Symbolleistendatei veranschaulicht. Im Beispiel sind Auslassungszeichen (. . .) für die Beschreibung der Attribute von Symbolleistenelementen angegeben.

```
<?xml version="1.0"?>
<!DOCTYPE toolbarset SYSTEM "-//Adobe//DWExtension toolbar 10.0">
<toolbarset>

<!-- main toolbar -->
    <toolbar id="DW_Toolbar_Main" label="Document">
        <radiobutton id="DW_CodeView" . . ./>
        <radiobutton id="DW_SplitView" . . ./>
        <radiobutton id="DW_DesignView" . . ./>
        <separator/>
        <checkbutton id="DW_LiveDebug" . . ./>
        <checkbutton 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>
```

Es folgen die Beschreibungen der einzelnen Symbolleisten-Tags.

<toolbar>

Beschreibung

Definiert eine Symbolleiste. Dreamweaver zeigt die Elemente und Trennlinien automatisch von links nach rechts in der angegebenen Reihenfolge an. Die Symbolleistendatei legt nicht den Abstand zwischen den einzelnen Elementen fest. Sie können jedoch die Breite einiger Elementtypen festlegen.

Attribute

`id, label, {container}, {initiallyVisible}, {initialPosition}, {relativeTo}`

- `id="unique_id"` Erforderlich. Ein Bezeichnerstring muss innerhalb einer Datei und in sämtlichen Dateien, die von dieser Datei importiert werden, eindeutig sein. Die JavaScript-API-Funktionen, die eine Symbolleiste ändern, verweisen durch deren ID auf den Bezeichnerstring. Weitere Informationen zu diesen Funktionen finden Sie im *Dreamweaver API-Referenzhandbuch*. Wenn zwei in derselben Datei enthaltene Symbolleisten dieselbe ID haben, zeigt Dreamweaver einen Fehler an.
- `label="string"` Erforderlich. Das label-Attribut legt den String für die Bezeichnung fest, die dem Benutzer in Dreamweaver angezeigt wird. Die Bezeichnung wird im Menü „Ansicht“ > „Symbolleisten“ und in der Titelleiste der Symbolleiste angezeigt, wenn diese abgedockt ist.
- `container="mainframe"` oder `"document"`. Der Standardwert ist `"mainframe"`. Gibt an, an welcher Position die Symbolleiste unter Windows im Dreamweaver-Arbeitsbereich angedockt ist. Wenn 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 angezeigt wird, wenn Dreamweaver sie zum ersten Mal aus dem Ordner „Toolbars“ lädt. Nach dem ersten Start steuert der Benutzer die Sichtbarkeit der Symbolleiste. Dreamweaver speichert den aktuellen Status in der Systemregistrierung (Windows) bzw. in der Datei für Dreamweaver-Voreinstellungen (Macintosh), wenn der Benutzer Dreamweaver beendet. Dreamweaver stellt die Einstellungen aus der Registrierung bzw. aus der Datei für Voreinstellungen wieder her, wenn die Anwendung wieder gestartet wird. Das Ein- und Ausblenden von Symbolleisten können Sie mit den Funktionen `dom.getToolbarVisibility()` und `dom.setToolbarVisibility()` festlegen (siehe dazu Beschreibung im *Dreamweaver API-Referenzhandbuch*). Wenn Sie das Attribut `initiallyVisible` nicht festlegen, wird standardmäßig `true` verwendet.
- `initialPosition="top"`, `"below"` oder `"floating"`. Gibt an, an welcher Stelle Dreamweaver anfangs die Symbolleiste relativ zu anderen Symbolleisten positioniert, wenn sie zum ersten Mal geladen wird. Die möglichen Werte für `intialPosition` sind in der folgenden Liste beschrieben:
- `top` Dies ist die Standardposition. Die Symbolleiste wird oben im Dokumentfenster angezeigt. Wenn `top` für mehrere Symbolleisten eines bestimmten Fenstertyps festgelegt ist, werden die Symbolleisten in der Reihenfolge angezeigt, in der Dreamweaver sie lädt. Wenn sich die Symbolleisten in unterschiedlichen Dateien befinden, ist diese Reihenfolge nicht immer vorhersehbar.
- `below` Die Symbolleiste wird am Anfang der Zeile direkt unterhalb der durch das `relativeTo`-Attribut angegebenen Symbolleiste angezeigt. Dreamweaver meldet einen Fehler, wenn die in `relativeTo` angegebene Symbolleiste nicht gefunden wird. Wenn bei mehreren Symbolleisten `below` relativ zur gleichen Symbolleiste festgelegt ist, werden die Symbolleisten in der Reihenfolge angezeigt, in der Dreamweaver sie lädt. Wenn sich die Symbolleisten 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 positioniert die Symbolleiste automatisch versetzt zu anderen schwebenden Symbolleisten. Auf dem Macintosh wird `floating` gleichbedeutend mit `top` behandelt.

  Wie das Attribut `initiallyVisible` wird das Attribut `initialPosition` nur angewendet, wenn Dreamweaver die Symbolleiste zum ersten Mal lädt. Anschließend wird die Position der Symbolleiste in der Registrierung bzw. in der Dreamweaver-Datei für Voreinstellungen gespeichert. Mithilfe der Funktion `dom.setToolbarPosition()` kann die Position der Symbolleiste zurückgesetzt werden. Weitere Informationen über die Funktion `dom.setToolbarPosition()` finden Sie im *Dreamweaver API-Referenzhandbuch*.

  Wenn Sie das Attribut `initialPosition` nicht angeben, positioniert Dreamweaver die Symbolleiste entsprechend der beim Laden entstandenen Reihenfolge.

- `relativeTo="toolbar_id"` Dieses Attribut ist erforderlich, wenn für das Attribut `initialPosition` der Wert `below` festgelegt ist. Andernfalls wird der Parameter ignoriert. Gibt die ID der Symbolleiste an, unter der diese Symbolleiste positioniert werden soll.

Inhalt

Das `toolbar`-Tag enthält die Tags `include`, `itemref` und `separator` sowie einzelne Elementdefinitionen, z. B. `button`, `combobox`, `dropdown` usw. Beschreibungen der Elementdefinitionen, die Sie angeben können, finden Sie unter „Symbolleistenelement-Tags“ auf Seite 188.

Container

Das `toolbarset`-Tag.

Beispiel

```
<toolbar id="MyDWedit_toolbar" label="Edit">
```

<include/>

Beschreibung

Lädt Symbolleistenelemente aus der angegebenen Datei, bevor der Ladevorgang der aktuellen Datei fortgesetzt 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 die rekursive <include>-Anweisung. Alle `toolbar`-Tags in der unter <include> angegebenen Datei werden übersprungen, Symbolleistenelemente dieser Symbolleisten können jedoch in der aktuellen Datei referenziert werden.

Attribute

- Der relativ zum Ordner „Toolbars“ angegebene `file`-Pfad der einzubindenden XML-Datei für die Symbolleiste.

Inhalt

Keine.

Container

Das `toolbar`-Tag oder das `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 Typen von Symbolleistenelementen, die Sie definieren können, finden Sie unter „Symbolleistenelement-Tags“ auf Seite 188.

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 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `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 zuvor definiert oder in die Datei aufgenommen wurde. Dreamweaver lässt keine vorwärts gerichteten Verweise zu. Wenn ein Symbolleistenelement-Tag auf eine nicht definierte ID verweist, meldet Dreamweaver einen Fehler und ignoriert den Verweis.
- `showIf="`*`script`*`"` Gibt an, dass dieses Element nur in der Symbolleiste angezeigt wird, wenn das angegebene Skript den Wert `true` zurückgibt. Sie können beispielsweise `showIf` verwenden, um bestimmte Schaltflächen nur in einer bestimmten Anwendung oder nur für Seiten in bestimmten serverseitigen Programmiersprachen wie Adobe ColdFusion, ASP oder JSP anzuzeigen. Wenn Sie `showIf` nicht angeben, wird das Element stets angezeigt. Diese Eigenschaft wird immer überprüft, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig vom Wert des `update`-Attributs. Dieses Attribut sollte sparsam verwendet werden. Das Attribut kann entweder in der Elementdefinition oder in einem Verweis auf das Element einer Symbolleiste verwendet werden. Wenn das `showIf`-Attribut sowohl in der Definition als auch im Verweis angegeben ist, zeigt Dreamweaver das Element nur an, wenn beide Bedingungen erfüllt sind. Das `showIf`-Attribut entspricht der Funktion `showIf()` in einer Befehlsdatei.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `toolbarset`-Tag.

Beispiel
```
<itemref id="strikeout_button">
```

<separator/>

Beschreibung
Fügt an der aktuellen Position eine Trennlinie in die Symbolleiste ein.

Attribute
{showIf}

- Das `showif`-Attribut gibt an, dass die Trennlinie nur in der Symbolleiste angezeigt wird, wenn das angegebene Skript den Wert `true` zurückgibt. Beispielsweise können Sie mit dem `showIf`-Attribut die Trennlinie nur in einer bestimmten Anwendung anzeigen oder nur, wenn die Seite einen bestimmten Dokumenttyp aufweist. Wenn das `showIf`-Attribut nicht definiert ist, wird die Trennlinie immer angezeigt.

Inhalt
Keine.

Container
Das `toolbar`-Tag.

Beispiel
```
<separator/>
```

Symbolleistenelement-Tags

Jeder Typ von Symbolleistenelementen verfügt über ein eigenes Tag und eine eigene Gruppe erforderlicher oder optionaler Attribute. Sie können `toolbar`-Elemente außerhalb oder innerhalb von Symbolleisten definieren. Normalerweise ist es besser, sie außerhalb von Symbolleisten zu definieren und innerhalb von Symbolleisten mithilfe des `itemref`-Tags auf sie zu verweisen.

Sie können folgende Elementtypen in Symbolleisten definieren.

<button>

Beschreibung
Wenn Sie auf diese Schaltfläche klicken, wird ein bestimmter Befehl ausgeführt. Die Schaltfläche sieht aus und verhält sich wie die Schaltfläche „Referenz“ in der Dreamweaver-Symbolleiste.

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 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `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"/>
```

<checkbutton>

Beschreibung
Ein Kontrollkästchen kann einen aktivierten oder deaktivierten Status haben und führt einen bestimmten Befehl aus, wenn es aktiviert wird. Ein aktiviertes Kontrollkästchen wird gedrückt und hervorgehoben dargestellt. Wenn es deaktiviert ist, wird es flach dargestellt. In Dreamweaver sind folgende Statusoptionen für Kontrollkästchen festgelegt: Mauszeiger über dem Kontrollkästchen, gedrücktes Kontrollkästchen, Mauszeiger über dem gedrückten Kontrollkästchen und deaktiviertes gedrücktes Kontrollkästchen. Die Verarbeitungsprozedur, die durch das `checked`-Attribut oder die Funktion `isCommandChecked()` angegeben wird, muss sicherstellen, dass der Status des Kontrollkästchens beim Klicken auf das Kontrollkästchen umgeschaltet wird.

Attribute
`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 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `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 deaktiviert ist. In Dreamweaver sind folgende Statusoptionen für Optionsschalter festgelegt: Mauszeiger über dem Optionsschalter, gedrückter Optionsschalter, Mauszeiger über dem gedrückten Optionsschalter und deaktivierter gedrückter Optionsschalter. In Dreamweaver müssen Optionsschalter sich nicht gegenseitig ausschließen. Die Verarbeitungsprozedur, die durch das `checked`-Attribut oder die Funktion `isCommandChecked()` angegeben 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 194.

Inhalt

Keine.

Container

Das `toolbar`-Tag oder das `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 angezeigt, das durch das `menuid`-Attribut angegeben ist. In Dreamweaver existieren für Menüschaltflächen die Statusoptionen „Mauszeiger über der Schaltfläche“ und „Gedrückt“. Dreamweaver erstellt den Menüpfeil nicht. Dabei handelt es sich um den Abwärtspfeil, der angibt, dass mit der Schaltfläche noch weitere Menüelemente verknüpft sind. Sie müssen diesen Pfeil in das Symbol einfügen. Beispiele für Menüschaltflächen in der Dokumentsymbolleiste von Dreamweaver sind die Schaltflächen „Dateiverwaltung“ und „Code-Navigation“.

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 194.

Inhalt

Keine.

Container

Das `toolbar`-Tag oder das `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 Steuerelement „Format“ im Eigenschafteninspektor für Text, mit der Ausnahme, dass es eine Standardgröße hat und damit größer ist als der Eigenschafteninspektor.

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 194.

Inhalt

Keine.

Container
Das `toolbar`-Tag oder das `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 einen Befehl ausführt, wenn Sie einen Eintrag auswählen oder der Benutzer im Textfeld Änderungen vornimmt und den Fokus wechselt. Das Menü sieht aus und verhält sich wie das Steuerelement „Schriftart“ im Eigenschafteninspektor für Text, mit der Ausnahme, dass es eine Standardgröße hat und damit größer ist als der Eigenschafteninspektor.

Attribute
`id, file, tooltip, enabled, value, command, {showiI}, {label}, {width}, {domRequired}, {update}, {arguments}`

Eine Beschreibung der einzelnen Attribute finden Sie unter „Attribute für Symbolleistenelement-Tags“ auf Seite 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `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 einen 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 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `toolbarset`-Tag.

Beispiel

```
<EDITCONTROL ID="DW_SetTitle"
    label="Title: "
    tooltip="Document Title"
    width="150"
    file="Toolbars/MM/EditTitle.htm"/>
```

<colorpicker>

Beschreibung
Eine Farbauswahl ist eine Farbpalette ohne verknüpftes Textfeld, die einen Befehl ausführt, wenn der Benutzer eine neue Farbe auswählt. Diese Palette sieht aus und verhält sich wie die Farbauswahl im Eigenschafteninspektor von Dreamweaver. Sie können das Standardsymbol 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 194.

Inhalt
Keine.

Container
Das `toolbar`-Tag oder das `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

Um das Aussehen und Verhalten von Symbolleistenelementen festzulegen, können Sie den Elementen Attribute und Befehle zuweisen. Sie können auch andere Symbolleistendateien einfügen und auf Symbolleistenelemente verweisen, die in anderen Symbolleisten definiert sind. Im Folgenden werden die Attribute für Symbolleistenelement-Tags erläutert.

id="unique_id"

Erforderlich. Das `id`-Attribut ist ein Bezeichner für das Symbolleistenelement. Das `id`-Attribut muss innerhalb der aktuellen Datei und in allen Dateien, die in der aktuellen Datei enthalten sind, eindeutig sein. Das `itemref`-Tag verweist mithilfe des `id`-Attributs auf ein Symbolleistenelement und fügt es in eine Symbolleiste ein.

Beispiel

```
<button id="DW_DocRerefresh" . . . >
```

showIf="script"

Optional. Dieses Attribut gibt an, dass das Element nur in der Symbolleiste angezeigt wird, wenn das Skript den Wert `true` zurückgibt. Sie können beispielsweise das Attribut `showIf` verwenden, um bestimmte Schaltflächen nur für Seiten in bestimmten serverseitigen Programmiersprachen wie Adobe ColdFusion, ASP oder JSP anzuzeigen. Wenn Sie `showIf` nicht angeben, wird das Element stets angezeigt.

Das `showIf`-Attribut wird immer überprüft, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig vom Wert des `update`-Attributs. Das `showIf`-Attribut sollte sparsam verwendet werden.

Sie können das `showIf`-Attribut in der Elementdefinition und in einem Verweis auf das Element in einem `itemref`-Tag angeben. Wenn das Attribut `showIf` sowohl in der Definition als auch im Verweis angegeben ist, wird das Element nur angezeigt, wenn beide Bedingungen erfüllt sind. Das `showIf`-Attribut entspricht der Funktion `showIf()` in einer Symbolleisten-Befehlsdatei. Wenn Sie sowohl das `showIf`-Attribut als auch die Funktion `showIf()` angeben, setzt die Funktion das Attribut außer Kraft.

Beispiel

```
showIf="dw.canLiveDebug()"
```

image="image_path"

Dieses Attribut ist erforderlich für Schaltflächen, Kontrollkästchen, Optionsschalter, Menüschaltflächen und Kombinationsfelder. Das `image`-Attribut ist für eine Farbauswahl optional und wird bei allen anderen Elementtypen ignoriert. Das `image`-Attribut legt den Pfad der auf der Schaltfläche angezeigten Symboldatei relativ zum Ordner „Configuration“ fest. Das Symbol kann jedes Format aufweisen, das in Dreamweaver dargestellt werden kann, ist jedoch normalerweise eine GIF- oder JPEG-Datei.

Wenn ein Symbol für eine Farbauswahl angegeben wird, ersetzt es das Farbauswahlsymbol vollständig. Wenn auch das Attribut `colorRect` angegeben ist, wird die aktuelle Farbe im festgelegten Rechteck über dem Symbol angezeigt.

Beispiel

```
image="Toolbars/images/MM/codenav.gif"
```

disabledImage="image_path"

Optional. Dreamweaver ignoriert das `disabledImage`-Attribut bei allen anderen Elementen außer Schaltflächen, Kontrollkästchen, Optionsschaltern, Menüschaltflächen, Farbauswahl-Steuerelementen und Kombinationsfeldern. Dieses Attribut gibt den Pfad der Symboldatei relativ zum Ordner „Configuration“ an, die Dreamweaver anzeigt, wenn die Schaltfläche deaktiviert ist. Wenn Sie das `disabledImage`-Attribut nicht angeben, zeigt Dreamweaver bei deaktivierter Schaltfläche das Bild an, das im `image`-Attribut festgelegt ist.

Beispiel

```
disabledImage="Toolbars/images/MM/codenav_dis.gif"
```

overImage="image_path"

Optional. Dreamweaver ignoriert das `overImage`-Attribut bei allen anderen Elementen außer Schaltflächen, Kontrollkästchen, Optionsschaltern, Menüschaltflächen, Farbauswahl-Steuerelementen und Kombinationsfeldern. Dieses Attribut gibt den Pfad der Symboldatei relativ zum Ordner „Configuration“ an, die Dreamweaver anzeigt, wenn der Benutzer den Mauszeiger über die Schaltfläche bewegt. Wenn Sie das Attribut `overImage` nicht angeben, zeichnet Dreamweaver lediglich einen Umriss um die Schaltfläche, wenn der Benutzer den Mauszeiger darüber bewegt. Darüber hinaus ändert sich die Schaltfläche nicht.

Beispiel

```
overImage="Toolbars/images/MM/codenav_ovr.gif"
```

tooltip="tooltip string"

Erforderlich. Dieses Attribut gibt den Identifizierungstext an (die sogenannte QuickInfo), der angezeigt wird, wenn der Mauszeiger über das Symbolleistenelement bewegt wird.

Beispiel

```
tooltip="Code Navigation"
```

label="label string"

Optional. Dieses Attribut gibt eine Bezeichnung an, die neben dem Element angezeigt wird. Dreamweaver fügt 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 Kombinationsfelder innerhalb der Schaltfläche und rechts neben dem Symbol.

Beispiel

```
label="Title: "
```

width="number"

Optional. Dieses Attribut wird auf die Elemente von Textfeldern, Popupmenüs und Kombinationsfeldern angewendet und gibt die Breite dieser Elemente in Pixel an. Wenn Sie das `width`-Attribut nicht angeben, verwendet Dreamweaver eine geeignete Standardbreite.

Beispiel

```
width="150"
```

menuID="menu_id"

Dieses Attribut ist für Menüschaltflächen und Kombinationsfeldern erforderlich, es sei denn, Sie legen die Funktion `getMenuID()` in einer verknüpften Befehlsdatei fest. Dreamweaver ignoriert das `menuID`-Attribut für andere Elementtypen. Dieses Attribut gibt die ID der Menüleiste mit dem Kontextmenü an, das geöffnet wird, wenn der Benutzer auf die Schaltfläche, die Menüschaltfläche oder das Kombinationsfeld klickt. Die ID stammt aus dem id-Attribut eines `menubar`-Tags in der Datei „menus.xml“.

Beispiel

```
menuID="DWCodeNavPopup"
```

colorRect="left top right bottom"

Dieses Attribut ist bei einer Farbauswahl optional, die über ein image-Attribut verfügt. Bei anderen Elementtypen und bei Farbauswahl-Steuerelementen ohne Angabe eines Bilds wird das `colorRect`-Attribut ignoriert. Wenn Sie das `colorRect`-Attribut angeben, zeigt Dreamweaver die aktuelle Farbe, die in der Farbauswahl ausgewählt ist, im Rechteck links oder über dem Symbol an. Wenn Sie das Attribut `colorRect` nicht angeben, zeigt Dreamweaver die aktuelle Farbe nicht im Bild an.

Beispiel

```
colorRect="0 12 16 16"
```

file="command_file_path"

Erforderlich für Popupmenüs und Kombinationsfelder. Bei anderen Elementtypen ist das `file`-Attribut optional. Das `file`-Attribut gibt relativ zum Ordner „Configuration“ den Pfad einer Befehlsdatei an, die JavaScript-Funktionen zum Füllen, Aktualisieren und Ausführen des Elements enthält. Das `file`-Attribut setzt die Attribute `enabled`, `checked`, `value`, `update`, `domRequired`, `menuID`, `showIf` und `command` außer Kraft. Wenn Sie mit dem `file`-Attribut eine Befehlsdatei angeben, ignoriert Dreamweaver im Allgemeinen alle entsprechenden Attribute, die im Tag angegeben sind. Weitere Informationen zu Befehlsdateien finden Sie unter „API-Funktionen für Symbolleistenbefehle“ auf Seite 199.

Beispiel

```
file="Toolbars/MM/EditTitle.htm"
```

domRequired="true" oder "false"

Optional. Wie bei Menüs gibt das `domRequired`-Attribut an, ob die Entwurfsansicht mit der Codeansicht synchronisiert werden soll, bevor Dreamweaver den verknüpften Befehl ausführt. Wenn Sie dieses Attribut nicht angeben, gilt der Standardwert `true`. Dieses Attribut entspricht der Funktion `isDOMRequired()` in einer Symbolleisten-Befehlsdatei.

Beispiel

```
domRequired="false"
```

enabled="script"

Optional. Wie bei Menüs gibt das Skript einen Wert zurück, der angibt, ob das Element aktiviert ist. Wenn Sie dieses Attribut nicht angeben, gilt der Standardwert „enabled“. Das `enabled`-Attribut entspricht der Funktion `canAcceptCommand()` in einer Symbolleisten-Befehlsdatei.

Beispiel

```
enabled="dw.getFocus() =='textView' || dw.getFocus() == 'html'"
```

checked="script"

Dieses Attribut ist für Kontrollkästchen und Optionsschalter erforderlich. Dreamweaver ignoriert das `checked`-Attribut für alle anderen Elementtypen. Wie bei Menüs gibt das Skript einen Wert zurück, der angibt, ob das Element aktiviert oder deaktiviert ist. Das `checked`-Attribut entspricht der Funktion `isCommandChecked()` in einer Symbolleisten-Befehlsdatei. Wenn Sie dieses Attribut nicht angeben, gilt der Standardwert „unchecked“.

Beispiel

```
checked="dw.getDocumentDOM() != null && dw.getDocumentDOM().getView() == 'code'"
```

value="script"

Dieses Attribut ist für Popupmenüs, Kombinationsfelder, Textfelder und Farbauswahl-Steuerelemente erforderlich. Dreamweaver ignoriert das `value`-Attribut für alle anderen Elementtypen.

Um zu ermitteln, welcher Wert für Popupmenüs und Kombinationsfelder angezeigt werden soll, ruft Dreamweaver zunächst für jedes Element im Menü die Funktion `isCommandchecked()` auf. Wenn die Funktion `isCommandchecked()` für eines der Elemente den Wert `true` zurückgibt, zeigt Dreamweaver den Wert des ersten Elements an. Wenn für kein Element der Wert `true` zurückgegeben wird oder die Funktion `isCommandChecked()` nicht definiert ist, ruft Dreamweaver die Funktion `getCurrentValue()` auf oder führt das Skript aus, das im `value`-Attribut angegeben ist. Wenn das Steuerelement ein Kombinationsfeld ist, zeigt Dreamweaver den zurückgegebenen Wert an. Wenn 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 Anzeigewert zurück. Bei Popupmenüs oder Kombinationsfeldern sollte es sich bei diesem 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 einer Farbauswahl sollte der Wert eine gültige Farbe sein, dies wird jedoch von Dreamweaver nicht überprüft.

Das value-Attribut entspricht der Funktion `getCurrentValue()` in einer Symbolleisten-Befehlsdatei.

update="update_frequency_list"

Optional. Dieses Attribut gibt an fest, wie oft die Verarbeitungsprozeduren für `enabled`, `checked`, `showif` und `value` ausgeführt werden sollen, um den angezeigten Status des Elements zu aktualisieren. Das `update`-Attribut entspricht der Funktion `getUpdateFrequency()` in einer Symbolleisten-Befehlsdatei.

Die Aktualisierungshäufigkeit für Symbolleistenelemente muss angegeben werden, da diese Elemente im Gegensatz zu Menüelementen immer sichtbar sind. Aus diesem Grund sollten Sie die geringstmögliche Häufigkeit auswählen und sicherstellen, dass die Verarbeitungsprozeduren für `enabled`, `checked` und `value` möglichst einfach sind.

Im Folgenden sind die möglichen Verarbeitungsprozeduren für *update_frequency_list* von der geringsten bis zur höchsten Aktualisierungshäufigkeit aufgeführt. Wenn Sie kein `update`-Attribut angeben, gilt für die Aktualisierungshäufigkeit der Standardwert `onEdit`. Sie können mehrere durch Kommas getrennte Aktualisierungshäufigkeiten angeben. Die Verarbeitungsprozeduren werden bei allen der folgenden Ereignisse 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 Leerlauf befindet. Dies kann sehr viel Zeit in Anspruch nehmen, da die Verarbeitungsprozeduren für `enabled/checked/showif/value` häufig ausgeführt werden. Dieser Wert 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 ausgelöst wurde und die Anwendung sich im Ruhezustand befindet. Es ist nicht sichergestellt, 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="script"

Dieses Attribut ist für alle Symbolleistenelemente außer Menüschaltflächen erforderlich. Dreamweaver ignoriert das `command`-Attribut für Menüschaltflächen. Gibt die JavaScript-Funktion an, die ausgeführt werden soll, wenn der Benutzer eine der folgenden Aktionen ausführt:

- Klicken auf eine Schaltfläche
- Auswählen eines Elements in einem Popupmenü oder Kombinationsfeld
- In einem Text- oder Kombinationsfeld: Drücken der Tabulatortaste oder der Eingabetaste bzw. Klicken auf eine Stelle außerhalb des Felds
- Auswählen einer Farbe in einem Farbauswahl-Steuerelement

Das `command`-Attribut entspricht der Funktion `receiveArguments()` in einer Symbolleisten-Befehlsdatei.

Beispiel

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

arguments="argument_list"

Optional. Dieses Attribut gibt die Liste der durch Kommas getrennten Argumente an, die an die Funktion `receiveArguments()` in einer Symbolleisten-Befehlsdatei übergeben werden. Wenn Sie das `arguments`-Attribut nicht angeben, übergibt Dreamweaver die ID des Symbolleistenelements. Darüber hinaus übergeben Popupmenüs, Kombinationsfelder, Textfelder und Farbauswahl-Steuerelemente ihren aktuellen Wert als erstes Argument vor allen Argumenten, die im `arguments`-Attribut angegeben sind, und vor der Element-ID, wenn keine Argumente angegeben sind.

Beispiel
In einer Symbolleiste mit den Schaltflächen „Rückgängig“ und „Wiederherstellen“ ruft jede der beiden Schaltflächen die Menübefehlsdatei „Edit_Clipboard.htm“ auf und übergibt ein Argument, das die gewünschte Aktion festlegt, z. B.:

```
<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"/>
```

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 Vorgehensweise ist dann notwendig, wenn für die Funktionen Argumente erforderlich sind, wie z. B. in der Befehlsprozedur für ein Textfeld. Dies ist der Fall bei Popupmenüs und Kombinationsfeldern.

Die Befehlsdatei-API für Symbolleistenelemente ist eine Erweiterung der Menübefehlsdatei-API. Sie können daher Menübefehlsdateien, unter Umständen mit einigen zusätzlichen Funktionen für Symbolleisten, direkt als Symbolleisten-Befehlsdateien wiederverwenden.

canAcceptCommand()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Ermittelt, ob das Symbolleistenelement aktiviert ist. Standardmäßig befindet sich ein Element im aktivierten Status. Deshalb sollten Sie diese Funktion nur definieren, wenn in mindestens einem Fall der Wert `false` zurückgegeben wird.

Argumente
Bei Popupmenüs, Kombinationsfeldern, Textfeldern und Farbauswahl-Steuerelementen ist das erste Argument der aktuelle Wert des Steuerelements. 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 in der Datei „toolbars.xml“ für dieses Symbolleistenelement das `arguments`-Attribut angeben, werden diese Argumente als Nächstes übergeben. Wenn Sie das `arguments`-Attribut nicht angegeben haben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Element aktiviert ist, andernfalls `false`.

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 Farbauswahl-Steuerelemente auf. Bei Popupmenüs sollte der aktuelle Wert eines der Menüelemente sein. Wenn sich der Wert nicht im Popupmenü befindet, wählt Dreamweaver das erste Element aus. Bei Kombinationsfeldern und Textfeldern kann dieser Wert ein beliebiger String sein, den die Funktion zurückgibt. Bei einer Farbauswahl sollte der Wert eine gültige Farbe sein, dies wird jedoch von Dreamweaver nicht überprüft. Diese Funktion entspricht dem `value`-Attribut.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String, der den aktuellen Anzeigewert enthält. Für die Farbauswahl enthält der String das RGB-Format der ausgewählten Farbe (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 an die Funktion`receiveArguments()` nicht den eigentlichen String, der im Menü angezeigt werden soll, sondern die ID.

Der Name `getDynamicContent()` ist eine Fehlbenennung, da diese Funktion auch dann verwendet werden sollte, wenn die Liste der Menüeinträge vorgegeben 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, kann dieselbe Befehlsdatei jedoch auch für ein Popupmenü in einer Symbolleiste verwendet werden. Für Symbolleistenelemente werden Unterstriche in den Strings eines zurückgegebenen Arrays ignoriert, damit Menübefehlsdateien wiederverwendet werden können. In der Menübefehlsdatei ignoriert Dreamweaver die Funktion `getDynamicContent()`, da das Menüelement nicht als dynamisch gekennzeichnet ist.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet ein Array von Strings, mit denen das Menü gefüllt 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
Nur gültig 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
Gibt an, wie häufig 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, da diese Elemente im Gegensatz zu Menüs immer angezeigt werden. Aus diesem Grund sollten Sie die geringstmögliche Häufigkeit auswählen und sicherstellen, dass die Verarbeitungsprozeduren für `enabled`, `checked` und `value` möglichst einfach sind.

Diese Funktion entspricht dem `update`-Attribut 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 197.

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 `checked`-Attribut in einem Symbolleistenelement-Tag.

Argumente
Bei Popupmenüs, Kombinationsfeldern, Textfeldern und Farbauswahl-Steuerelementen ist das erste Argument der aktuelle Wert des Steuerelements. 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 ermitteln, ob eine Übereinstimmung mit dem Textfeld besteht, führt Dreamweaver einen Vergleich mit dem Menü ohne Berücksichtigung von Groß- und Kleinschreibung durch.

Wenn Sie das `arguments`-Attribut angegeben haben, werden diese Argumente als Nächstes übergeben. Wenn Sie das `arguments`-Attribut nicht angeben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Element markiert ist, andernfalls `false`.

Beispiel

Im folgenden Beispiel wird ermittelt, welches Element, sofern 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

Gibt an, ob für den Symbolleistenbefehl ein gültiges DOM erforderlich ist. 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 dem Ausführen des verknüpften Befehls. Diese Funktion entspricht dem `domRequired`-Attribut in einem Symbolleistenelement-Tag.

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet einen booleschen Wert. `true`, wenn das DOM erforderlich ist, andernfalls `false`.

Beispiel

```
function isDOMRequired()
{
    return false;
}
```

receiveArguments()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Verarbeitet alle Argumente, die von einem Symbolleistenelement übergeben werden. Die Funktion `receiveArguments()` entspricht dem `command`-Attribut in einem Symbolleistenelement-Tag.

Argumente

Bei Popupmenüs, Kombinationsfeldern, Textfeldern und Farbauswahl-Steuerelementen ist das erste Argument der aktuelle Wert des Steuerelements. 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 ermitteln, ob eine Übereinstimmung mit dem Textfeld besteht, führt Dreamweaver einen Vergleich mit dem Popupmenü ohne Berücksichtigung von Groß- und Kleinschreibung durch.

Wenn Sie das `arguments`-Attribut angegeben haben, werden diese Argumente als Nächstes übergeben. Wenn Sie das `arguments`-Attribut nicht angegeben haben, übergibt Dreamweaver die ID des Elements.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
function receiveArguments(newTitle)
{
    var dom = dw.getDocumentDOM();
    if (dom)
        dom.setTitle(newTitle);
}
```

showIf()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Gibt an, dass ein Element nur in der Symbolleiste angezeigt wird, wenn die Funktion den Wert `true` zurückgibt. Beispielsweise können Sie mit der Funktion `showIf()` bestimmte Schaltflächen nur anzeigen lassen, wenn der Seite ein bestimmtes Servermodell zugewiesen ist. Wenn die Funktion `showif()` nicht definiert ist, wird das Element stets angezeigt. Die Funktion `showIf()` entspricht dem `showIf`-Attribut in einem Symbolleistenelement-Tag.

Die Funktion `showIf()` wird immer aufgerufen, wenn der Enabler des Elements ausgeführt wird, d. h. abhängig vom Wert, den die Funktion `getUpdateFrequency()` zurückgibt.

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Element angezeigt werden soll, andernfalls `false`.

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 unterstützt zwei Arten von Berichten: 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 die Datei „_foldername.txt“ vorhanden ist, verwendet Dreamweaver deren Inhalt als Kategorienamen. Wenn die Datei„ _foldername.txt“ nicht vorhanden ist, verwendet Dreamweaver den Ordnernamen als Kategorienamen.

Wenn der Benutzer im Dialogfeld „Berichte“ mehrere Site-Berichte auswählt, gibt Dreamweaver alle Ergebnisse im gleichen Ergebnisfenster auf der Registerkarte „Site-Berichte“ des Bedienfelds „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 Berichtinhalts.
Configuration/Reports/{Typ/}	*Berichtname*.htm	Ruft die entsprechenden JavaScript-Dateien auf. Definiert die Benutzeroberfläche für das Dialogfeld „Einstellungen“ des Berichts, sofern erforderlich.
Configuration/Reports/	Reports.js	Stellt allgemeine Funktionen zum Generieren von Berichten zur Verfügung.

Funktionsweise von Site-Berichten

1. Berichte werden über den Befehl „Site“ > „Berichte“ aufgerufen. Mit diesem Befehl wird ein Dialogfeld angezeigt, in dem der Benutzer Berichte auswählen kann, die für die angegebenen Ziele ausgeführt werden.
2. Der Benutzer legt im Popupmenü „Bericht über“ fest, für welche Dateien die ausgewählten Berichte ausgeführt 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“ auswählt, kann er über die Schaltfläche „Durchsuchen“ oder über ein Textfeld den gewünschten Ordner auswählen.
3. Der Benutzer kann Berichte, die Parameter enthalten, anpassen, indem er auf die Schaltfläche „Einstellungen“ klickt und Werte für die Parameter eingibt. Damit Benutzer Berichtparameter festlegen können, muss der jeweilige Bericht das Dialogfeld „Einstellungen“ enthalten. Dieses Dialogfeld ist optional. Nicht in allen Berichten muss der Benutzer Berichtparameter eingeben. 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:** Wenn ein Bericht die Prozedur `preventFileActivity` enthält, verhindert Dreamweaver, dass der Benutzer eine andere Dateiaktivität ausführen kann, während der Bericht ausgeführt wird.*

Dreamweaver löscht alle Elemente auf der Registerkarte „Site-Berichte“ des Bedienfelds „Ergebnisse“. Dreamweaver ruft für jeden Bericht die Funktion `beginReporting()` auf, bevor der Berichtsvorgang gestartet wird. Wenn ein Bericht für diese Funktion den Wert `false` zurückgibt, wird er aus dem Berichtslauf entfernt.

5 Alle Dateien werden mit der Funktion `processFile()` an die einzelnen im Dialogfeld „Berichte“ ausgewählten Berichte übergeben. Wenn der Bericht Informationen über eine Datei in der Ergebnisliste anzeigen soll, muss die Funktion `dw.resultsPalette.siteReports.addResultItem()` aufgerufen werden. Dieser Vorgang wird fortgesetzt, bis alle entsprechenden Dateien verarbeitet sind oder der Benutzer unten im Fenster auf die Schaltfläche „Stopp“ klickt. Dreamweaver zeigt die Namen aller bereits verarbeiteten Dateien und die Anzahl der noch zu verarbeitenden Dateien an.

6 Dreamweaver ruft für jeden Bericht die Funktion `endReporting()` auf, nachdem alle Dateien verarbeitet wurden und der Berichtsvorgang abgeschlossen ist.

Beispiel für einen einfachen Site-Bericht

In diesem Beispiel für eine einfache Erweiterung werden alle Bilder aufgelistet, auf die in einer bestimmten Datei, in einer gesamten Site, in ausgewählten Dateien oder in einem Ordner verwiesen wird. Der Bericht wird dann im Ergebnisfenster auf der Registerkarte „Site-Berichte“ angezeigt.

Zum Erstellen dieser Erweiterung legen Sie eine Berichtdefinition an und verfassen den JavaScript-Code.

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 JavaScript-Code für diesen Bericht enthält. Zusätzlich verweisen Sie auf die Datei „Reports.js“, die Bestandteil von Dreamweaver ist.

Erstellen der Berichtdefinition

Die Berichtdefinition gibt den Berichtnamen an, wie er im Dialogfeld „Berichte“ angezeigt wird, ruft die erforderlichen JavaScript-Dateien auf und definiert gegebenenfalls die Benutzeroberfläche des Dialogfelds „Einstellungen“.

1 Erstellen Sie im Ordner „Configuration/Reports/HTML Reports“ die Datei „List images.htm“.

2 Fügen Sie Folgendes hinzu, um den Berichtnamen anzugeben, der im Dialogfeld „Berichte“ im Titel der HTML-Seite angezeigt werden soll.

```
<html>
<head>
<title>List Images</title>
```

3 Fügen Sie am Ende der Datei das `script`-Tag ein und geben Sie im `src`-Attribut 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 im `src`-Attribut die Datei „List Images.js“ an, die Sie im Folgenden 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 „List Images.js“ im Ordner „Configuration/Reports/HTML Reports“.

Verfassen des JavaScript-Codes

Die Datei „Reports.js“ ist bereits in Dreamweaver enthalten. Sie können beliebige Funktionen aus der Datei „Reports.js“ aufrufen, müssen jedoch auch die JavaScript-Datei mit den Funktionen für Ihren benutzerdefinierten Site-Bericht erstellen.

1 Erstellen Sie im Ordner „Configuration/Reports/HTML Reports“ die Datei „List Images.js“ mit folgendem Inhalt:

```
// Function: configureSettings
// Description: Standard report API, used to initialize and load
//the default values. Does not initialize the UI.
//
function configureSettings() {
    return false;
}
// Function: processFile
// Description: Report command API called during file processing.
//
function processFile (fileURL) {
    if (!isHTMLType(fileURL)) //If the file isn't an HTML file
        return;                    //skip it.
    var curDOM = dw.getDocumentDOM(fileURL);    // Variable for DOM
    var tagList = curDOM.getElementsByTagName('img'); // Variable for img tags
    var imgfilename; // Variable for file name specified in img tag
    for (var i=0; i < tagList.length; i++) { // For img tag list
        imgfilename = tagList[i].getAttribute('src'); // Get image filename
        if (imgfilename != null) { // If a filename is specified
        // Print the appropriate icon, HTML filename,
        // image filename, and line number
        reportItem(REP_ITEM_CUSTOM, fileURL, imgfilename,
                curDOM.nodeToSourceViewOffsets(tagList[i])); }
    }
}
```

2 Speichern Sie die Datei unter dem Namen „List Images.js“ im Ordner „Configuration/Reports/HTML Reports“.

Eigenständige Berichte

Mit der API für das Ergebnisfenster können eigenständige Berichte erstellt werden. Eigenständige Berichte sind reguläre Befehle, die anstelle der API für Berichte direkt die API für das Ergebnisfenster 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 Ergebnisfenster, wenn der Benutzer einen neuen eigenständigen Bericht ausführt.

Pfad	Datei	Beschreibung
Configuration/Commands	*Befehlsname*.htm	Definiert die Benutzeroberfläche für das Dialogfeld, das angezeigt wird, wenn der Benutzer den Befehl auswählt. Die Datei enthält außerdem den JavaScript-Code oder einen Verweis auf eine JavaScript-Datei mit dem Code, der die zum Generieren des Berichts erforderlichen Aktionen ausführt.
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 bei diesem Vorgang sollten als Methoden dieser Objekte aufgerufen werden.
2. Der benutzerdefinierte Befehl 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 Ergebnisfenster 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()` aufgerufen wird, ruft Dreamweaver für jede Datei-URL in der Liste die Funktion `processFile()` auf. Definieren Sie die Funktion `processFile()` in dem eigenständigen Befehl. Dieser Funktion wird als einziges Argument die Datei-URL übergeben. 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 Ergebnisfenster, das mit dem eigenständigen Befehl erstellt wurde. Die Funktion `processFile()` kann auch die Funktion `stopProcessing()` des Ergebnisfenster-Objekts aufrufen, um die Verarbeitung der Dateiliste zu beenden.

Beispiel für einen einfachen eigenständigen Bericht

Die Erweiterung für einen einfachen eigenständigen Bericht listet alle Bilder auf, auf die in einer bestimmten Datei verwiesen wird, und zeigt den Bericht im Ergebnisfenster an.

Zum Erstellen dieser Erweiterung legen Sie ein Dialogfeld als Benutzeroberfläche an und verfassen den JavaScript-Code.

In diesem Beispiel werden zwei Dateien im Ordner „Configuration/Commands“ erstellt: die Datei „List images.htm“ zum Definieren der Benutzeroberfläche des Dialogfelds, das bei Auswahl des benutzerdefinierten Befehls angezeigt wird, und die Datei „Listimages.js“ mit dem für diesen Bericht spezifischen JavaScript-Code.

Erstellen der Benutzeroberfläche des Dialogfelds

Der body-Bereich der HTML-Datei gibt den Inhalt des Dialogfelds an, das angezeigt wird, wenn der Benutzer den benutzerdefinierten Befehl auswählt, und ruft die erforderlichen JavaScript-Dateien auf.

1. Erstellen Sie die Datei „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“.

Verfassen des JavaScript-Codes

Erstellen Sie die JavaScript-Datei, die alle Funktionen enthält, die spezifisch für Ihren eigenständigen Bericht sind.

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++)
    {
        // Get the name of the source file.
            imgfilename = tagList[i].getAttribute('src');
        // Get the character offset from the start of the file
        // to the start of the img tag.
        iOffset = curDOM.nodeToOffsets(curDOM.images[i]);
        // Based on the offset, figure out what line in the file
        // the img tag is on.
        iLineNumber = curDOM.getLineFromOffset(iOffset[0]);
        // As long as the src attribute specifies a file name,
            if (imgfilename != null)
        { // display the line number, and image path.
            resWin.addItem(resWin, "0", "Images in Current File", null, ¬
                null, null, [iLineNumber, imgfilename]);
            }
    }
    return;
}

// add buttons to dialog
function commandButtons()
{
    return new Array("OK", "stdaloneresultwin()", "Cancel", "window.close()");
}
```

2 Speichern Sie die Datei unter dem Namen „Listimages.js“ im Ordner „Configuration/Commands“.

API-Funktionen für Berichte

Die einzige erforderliche API-Funktion 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 Befehl für Berichte verarbeitet die Datei, ohne sie zu ändern, und verwendet die Funktionen `dw.ResultsPalette.SiteReports()`, `addResultItem()` oder `resWin.addItem()`, um Informationen über die Datei zurückzugeben. Dreamweaver gibt das DOM jeder Datei 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 keine Rückgabewerte.

beginReporting()

Verfügbarkeit
Dreamweaver 4.

Beschreibung
Diese Funktion wird am Anfang des Berichtsvorgangs noch vor dem Ausführen des ersten Berichts aufgerufen. Wenn der Befehl für diese Funktion den Wert `false` zurückgibt, wird er aus dem Berichtsvorgang entfernt.

Argumente
target

Das Argument *target* ist ein String, der das Ziel der Berichtssitzung angibt. Zulässige Werte sind `"CurrentDoc"`, `"CurrentSite"`, `"CurrentSiteSelection"` (für die ausgewählten Dateien in einer Site) oder `"Folder:+`*Pfad zum vom Benutzer ausgewählten Ordner"* (z. B. `"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 ist.

endReporting()

Verfügbarkeit
Dreamweaver 4.

Beschreibung
Diese Funktion wird am Ende des Berichtsvorgangs aufgerufen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

commandButtons()

Verfügbarkeit
Dreamweaver 4.

Beschreibung
Definiert die Schaltflächen, die auf der rechten Seite des Dialogfelds „Optionen“ angezeigt werden sollen, und das Verhalten beim Klicken auf die Schaltflächen. 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 Verhalten beim Klicken auf die oberste Schaltfläche festgelegt wird. Mit den restlichen Elementen werden weitere Schaltflächen in der gleichen Weise definiert.

Beispiel
Mit der folgenden Instanz der Funktion `commandButtons()` werden die Schaltflächen „OK“, „Abbrechen“ und „Hilfe“ definiert.

```
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 ist.

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: *Definieren Sie diese Funktion nur, 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 Plattform des Benutzers.

Rückgabewerte
Dreamweaver erwartet einen String des Typs `"widthInPixels,heightInPixels"`.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da der Bereich für die Schaltflächen „OK“ und „Abbrechen“ nicht einbezogen ist. Wenn im Fenster mit den zurückgegebenen Abmessungen nicht alle Optionen angezeigt werden können, werden Bildlaufleisten eingeblendet.

Beispiel
Im folgenden Beispiel für die Funktion `windowDimensions()` werden die Abmessungen des Dialogfelds „Parameter“ auf 648 x 520 Pixel festgelegt.

```
function windowDimensions(){
    return "648,520";
}
```

Kapitel 13: Tag-Bibliotheken und Tag-Editoren

Dreamweaver speichert Informationen zu den einzelnen Tags einschließlich aller Tag-Attribute in mehreren Unterordnern des Ordners „Configuration/TagLibraries“. Bei der Bearbeitung von Tags verwenden der Tag-Editor und die Tag-Auswahl die in diesem Ordner gespeicherten Informationen.

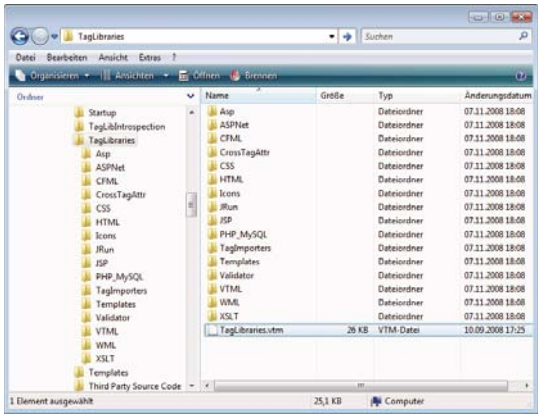
Dreamweaver enthält Editoren für die folgenden Sprachen: HTML, ASP.NET, CFML, JRun und JSP. Sie können die mit Dreamweaver gelieferten Tag-Editoren anpassen oder neue Tag-Editoren erstellen. Zudem können Sie den Tag-Bibliotheken neue Tags hinzufügen.

Bevor Sie benutzerdefinierte Tag-Editoren erstellen, sollten Sie mit der Struktur einer Tag-Bibliothek vertraut sein. In der folgenden Tabelle sind die Dateien zum Erstellen von Tag-Bibliotheken aufgeführt:

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.

Dateiformat von Tag-Bibliotheken

Eine Tag-Bibliothek besteht aus einer einzelnen Stammdatei mit dem Namen „TagLibraries.vtm“, in der alle installierten Tags aufgelistet sind, und je einer VTML-Datei für jedes Tag in der Tag-Bibliothek. Die Datei „TagLibraries.vtm“ dient als Inhaltsverzeichnis. Sie enthält Verweise auf die VTML-Dateien der einzelnen Tags. In der folgenden Abbildung ist dargestellt, wie die VTML-Dateien in Dreamweaver nach Markup-Sprachen strukturiert sind:



Benutzern von Macromedia HomeSite von Adobe ist die VTML-Dateistruktur möglicherweise vertraut. In Dreamweaver werden VTML-Dateien jedoch auf etwas andere Weise als in HomeSite verwendet. Der wichtigste Unterschied besteht darin, dass Dreamweaver über einen eigenen HTML-Renderer zum Anzeigen von Benutzeroberflächen für Erweiterungen verfügt. Die VTML-Dateien werden daher nicht bei der Darstellung grafischer Benutzeroberflächen verwendet.

Im folgenden Beispiel ist 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 `taglibrary`-Tag werden Tags in einer Tag-Bibliothek zusammengefasst. 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 Java Server 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` aufgeführt:

Attribut	**Beschreibung**	**Obligatorisch/optional**
`taglibrary.name`	Mit diesem Namen wird die Tag-Bibliothek in der Benutzeroberfläche angegeben.	Obligatorisch
`taglibrary.doctypes`	Gibt die Dokumenttypen an, für die diese Bibliothek aktiv ist. Wenn die Bibliothek aktiv ist, werden im Menü „Codehinweise“ Tags der Bibliothek angezeigt. Aufgrund möglicher Namenskonflikte können nicht alle Tag-Bibliotheken gleichzeitig aktiv sein (HTML- und WML-Dateien sind beispielsweise nicht kompatibel).	Obligatorisch
`taglibrary.prefix`	Wenn dieses Attribut angegeben ist, weisen Tags in der Tag-Bibliothek das Format `taglibrary.prefix+ tagref.name` auf. Wenn z. B. `taglibrary.prefix` den Wert `"<jrun:"` und `tagref.name` den Wert `"if"` hat, gilt für das Tag das Format `"<jrun:if"`. Dies kann für ein bestimmtes Tag außer Kraft gesetzt werden.	Optional

Attribut	Beschreibung	Obligatorisch/optional
`taglibrary.servermodel`	Wenn die Tags in der Tag-Bibliothek auf einem Anwendungsserver ausgeführt werden, gibt das Attribut `servermodel` das Servermodell des Tags an. Bei Client-seitigen Tags (d. h. nicht serverbasierten Tags) wird das Attribut `servermodel` nicht angegeben. Das Attribut `servermodel` wird auch für „Zielbrowser überprüfen“ verwendet.	Optional
`taglibrary.id`	Dies kann ein beliebiger String sein, der sich von den `taglibrary.ID`-Attributen anderer Tag-Bibliotheken in der Datei unterscheidet. Der Extension Manager verwendet das ID-Attribut, damit die MXP-Dateien einen neuen `taglibrary`-Wert und die `tags`-Dateien in die Datei „TagLibraries.vtm“ einfügen können.	Optional
`taglibrary.tagchooser`	Ein relativer Pfad zur Datei „TagChooser.xml“, die mit dieser Tag-Bibliothek verknüpft ist.	Optional

In der folgenden Tabelle sind die `tagref`-Attribute aufgeführt:

Attribut	Beschreibung	Obligatorisch/optional
`tagref.name`	Mit diesem Namen wird das Tag in der Benutzeroberfläche angegeben.	Obligatorisch
`tagref.prefix`	Gibt an, wie das Tag in der Codeansicht angezeigt wird. Wenn das Attribut `tagref.prefix` verwendet wird, legt es das Präfix des aktuellen Tags fest. Wenn das Attribut definiert ist, setzt es den für `taglibrary.prefix` angegebenen Wert außer Kraft.	Optional
`tagref.file`	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 „taglibrary.prefix“ definiert?	Wurde das Attribut „tagref.prefix“ definiert?	Ergebnis für das Tag-Präfix
Nein	Nein	`'<' + tagref.name`
Ja	Nein	`taglibrary.prefix + tagref.name`
Nein	Ja	`tagref.prefix`
Ja	Ja	`tagref.prefix`

Zur Definition von Tags verwendet Dreamweaver eine abgeänderte Version des VTML-Dateiformats. Im folgenden Beispiel sind alle Elemente angegeben, die Dreamweaver zur Definition eines 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"/>
    <attributes>
        <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 sind die Attribute aufgeführt, mit denen Tags definiert werden:

Attribut	**Beschreibung**	**Obligatorisch/optional**
`tag.bind`	Wird im 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 Tag-Namen 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. Wenn nicht zwischen Groß- und Kleinschreibung unterschieden wird, 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 `casesensitive`-Attribut nicht angegeben wird, 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 ein 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 zu erzwingen. Mit `<tag name="foo" endtag="xml" tagtype="empty">` wird z. B. 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. Dieses Attribut hat den Wert `No` für Tags wie `SCRIPT` und `STYLE`, die keinen HTML-Inhalt haben.	Optional
`tagformat.nlbeforetag`	Legt fest, ob vor dem Tag ein Zeilenumbruchzeichen eingefügt wird. Der Wert `0` bedeutet „Nein“ und der Wert `1` „Ja“.	Optional
`tagformat.nlbeforecontents`	Gibt an, wie viele Zeichen für Zeilenumbrüche vor dem Inhalt dieses Tags eingefügt werden sollen.	Optional

Attribut	Beschreibung	Obligatorisch/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 dem Tag ein Zeilenumbruchzeichen eingefügt wird. Der Wert `0` bedeutet „Nein“ und der Wert `1` „Ja“.	Optional
`attrib.name`	Der Name des Attributs, wie er im Quellcode dargestellt wird.	Obligatorisch
`attrib.type`	Wenn das Attribut `attrib.type` nicht angegeben wird, hat es den Wert `"text"`. Folgende Werte sind möglich: `TEXT` (freier Textinhalt), `ENUMERATED` (Liste mit Aufzählungswerten), `COLOR` (Farbwert als Name oder Hexadezimalzahl), `FONT` (Schriftname oder Schriftfamilie), `STYLE` (CSS-Stilattribut), `CSSSTYLE` (CSS-Klassenname), `CSSID` (CSS-Klassen-ID), `FILEPATH` (vollständiger Dateipfad), `DIRECTORY` (Ordnerpfad), `FILENAME` (nur der Dateiname), `RELATIVEPATH` (relative Pfadangabe), `FLAG` (ON/OFF-Attribut ohne Wert).	Optional
`attrib.casesensitive`	Gibt an, ob beim Attributnamen zwischen Groß- und Kleinschreibung unterschieden wird. Wenn das Attribut `CASESENSITIVE` nicht angegeben ist, wird beim Attributnamen nicht zwischen Groß- und Kleinschreibung unterschieden.	Optional

Hinweis: *In älteren Dreamweaver-Versionen (vor Dreamweaver MX) wurden die Tag-Informationen in der Datei „Configuration/TagAttributeList.txt“ gespeichert.*

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 der Tag-Bibliothek hinzufügen. Dazu können Sie das Dialogfeld „Tag-Bibliothek-Editor“ verwenden oder eine Dreamweaver-Erweiterung installieren, die als MXP-Datei gepackt ist .

„TagChooser.xml“-Dateien

Die „TagChooser.xml“-Dateien enthalten die Metadaten zum Strukturieren der Tag-Gruppen, die in der Tag-Auswahl angezeigt werden. Alle mit Dreamweaver installierten Tags sind in einer funktionellen Gruppe gespeichert und stehen 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 umstrukturieren, 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.

Wenn das Attribut `taglibrary.tagchooser` nicht vorhanden ist, wird in der Tag-Auswahl die Struktur aus der Datei „TagLibraries.vtm“ angezeigt.

„TagChooser.xml“-Dateien sind im Ordner „Configuration/TagLibraries/*Tag-Bibliotheksname*“ gespeichert. Im folgenden Beispiel ist die Struktur der „TagChooser.xml“-Dateien dargestellt:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<tclibrary 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"/>
            ... more elements to display in the list view ...
        </category>
        ... more subcategories ...
    </category>
    ... more categories ...
</tclibrary>
```

In der folgenden Tabelle sind die Tags aufgeführt, 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 der Tag-Bibliothek umschließt.	Obligatorisch
`tclibrary.name`	Der Wert wird im 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 `desc` nicht vorhanden ist, stammen die Informationen für „Tag-Info“ aus dem Bedienfeld „Referenz“. Austauschbar mit `tclibrary.reference`.	Optional (`desc` und `reference` 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 `tclibrary.desc`.	Optional (`desc` und `reference` schließen sich gegenseitig aus).

Das Tag `category` steht für alle anderen Knoten in der Strukturansicht unterhalb des Knotens `tclibrary`, wie in der folgenden Tabelle dargestellt:

Tag	**Beschreibung**	**Obligatorisch/optional**
`category.name`	Der Wert wird im Strukturansichtsknoten angezeigt.	Obligatorisch
`category.desc`	Der Wert ist ein HTML-String, der im Bereich „Tag-Info“ des Dialogfelds „Tag-Auswahl“ angezeigt wird. Wenn weder `desc` noch `reference attr` angegeben wird, ist der Bereich „Tag-Info“ leer.	Optional (`desc` und `reference` schließen sich gegenseitig aus).
`category.reference`	Der Wert beschreibt Sprache, Thema und Unterthema für die Anzeige im Bereich „Tag-Info“.	Optional (`desc` und `reference` 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 `category.id`-Attributen 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 im Code platziert wird, oder ein Parameter, der an grafische Dialogfelder übergeben wird.	Obligatorisch
`element.desc`	Der Wert ist ein HTML-String und wird im integrierten Bedienfeld „Referenz“ angezeigt. Wenn dieses Attribut nicht angegeben wird, zeigt das `reference`-Attribut den Referenzinhalt im integrierten Bedienfeld „Referenz“ an.	Optional (`desc` und `reference` schließen sich gegenseitig aus).
`element.reference`	Bis zu drei durch Kommas 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 `element` obligatorisch. Für die Tags `category` und `tclibrary` ist er dagegen optional. Der dritte String ist optional.	Optional (`desc` und `reference` schließen sich gegenseitig aus).
`element.id`	Ein beliebiger String, der sich von den `element.id`-Attributen anderer Elemente in dieser Datei unterscheidet.	Optional

Einfaches Beispiel für das Erstellen neuer Tag-Editoren

In den Beispielen in diesem Abschnitt werden die erforderlichen Schritte zum Erstellen eines neuen Tag-Editors veranschaulicht. Dazu wird das hypothetische Adobe ColdFusion-Tag `cfweather` zum Extrahieren der aktuellen Temperatur aus einer Wetterdatenbank verwendet.

Die Attribute für `cfweather` sind in der folgenden Tabelle aufgeführt:

Attribut	Beschreibung
`zip`	Fünfstellige Postleitzahl
`tempaturescale`	Temperaturskala (Celsius oder Fahrenheit)

Sie erstellen diesen neuen Tag-Editor, indem Sie das Tag registrieren, eine Tag-Definition erstellen, eine Benutzeroberfläche für den Tag-Editor definieren und dann der Tag-Auswahl ein Tag hinzufügen.

Registrieren des Tags in der Tag-Bibliothek

Damit Dreamweaver das neue Tag erkennen kann, muss es in der Datei „TagLibraries.vtm“ im Ordner „Configuration/TagLibraries“ angegeben sein. Wenn der Benutzer jedoch eine Mehrbenutzerplattform wie Windows XP, Windows 2000, Windows NT oder Mac OS X verwendet, befindet sich eine weitere Datei des Typs „TagLibraries.vtm“ im Konfigurationsordner des jeweiligen Benutzers. Diese Datei muss aktualisiert werden, da sie die Instanz ist, die Dreamweaver sucht und analysiert.

Der Speicherort des Benutzerordners „Configuration“ auf dem Computer hängt jeweils vom Betriebssystem ab.

Unter Windows 2000 und Windows XP:

```
<drive>:\Documents and Settings\<username>\Application Data\Adobe\\xc2
Dreamweaver CS5\Configuration
```

Hinweis: *Dieser Ordner befindet sich unter Windows XP möglicherweise in einem versteckten Ordner.*

Unter Mac OS X:

```
<drive>:Users:<username>:Library:Application Support:Adobe:¬
Dreamweaver CS5:Configuration
```

Wenn Dreamweaver die Datei „TagLibraries.vtm“ nicht im Konfigurationsordner des Benutzers finden kann, wird sie im Konfigurationsordner von Dreamweaver gesucht.

Hinweis: *Wenn Sie auf Mehrbenutzerplattformen die Version von „TagLibraries.vtm“ im Dreamweaver-Ordner „Configuration“ bearbeiten, haben die Änderungen keine Auswirkungen auf Dreamweaver. Dreamweaver analysiert die Datei „TagLibraries.vtm“ im Konfigurationsordner des Benutzers, nicht die Datei im Konfigurationsordner von Dreamweaver.*

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.

1. Öffnen Sie die Datei „TagLibraries.vtm“ in einem Texteditor.
2. Navigieren Sie durch die vorhandenen Tag-Bibliotheken, bis Sie die CFML-Tags finden.
3. Fügen Sie ein neues Tag-Referenzelement hinzu, wie im folgenden Beispiel dargestellt:

```
<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“.

Erstellen einer Tag-Definitionsdatei (VTML-Datei)

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.

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.

Erstellen einer Benutzeroberfläche für den Tag-Editor

1. Speichern Sie die Datei „cfweather.htm“ im Ordner „Configuration/TagLibraries/CFML“:

```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//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;    // tempaurelist control (initialized in initializeUI())
var theUIObjects;          // array of UI objects used by common API functions
/*************************************************************/

// inspectTag() API function defined (required by all tag editors)
function inspectTag(tagNodeObj)
{
    // call into a common library version of inspectTagCommon defined
    // in tagDialogCmns.js (note that it's been included)
    // For more information about this function, look at the comments
    // for inspectTagCommon in tagDialogCmn.js
    tagDialog.inspectTagCommon(tagNodeObj, theUIObjects);
}
function applyTag(tagNodeObj)
{
    // call into a common library version of applyTagCommon defined
    // in tagDialogCmns.js (note that it's been included)
    // For more information about this function, look at the comments
    // for applyTagCommon in tagDialogCmn.js
    tagDialog.applyTagCommon(tagNodeObj, theUIObjects);
}
function initializeUI()
{
    // define two arrays for the values and display captions for the list
    control
    var theTempatureScaleCap = new Array("celsius","fahrenheit");
    var theTempatureScaleVal = new Array("celsius","fahrenheit");

    // instantiate a new list control
    TEMPATURESCALELIST = new ListControl("thetempaturescale");

    // add the tempaturescalelist dropdown list control to the uiobjects
    theUIObjects0= new Array(TEMPATURESCALELIST);

    // call common populateDropDownList function defined in tagDialogCmn.js to
    // populate the tempaturescale list control
    tagDialog.populateDropDownList(TEMPATURESCALELIST, theTempatureScaleCap,
    theTempatureScaleVal, 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/> 
            </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>
```

Nun überprüfen Sie, ob der Tag-Editor funktioniert.

2 Starten Sie Dreamweaver.

3 Geben Sie in der Codeansicht **cfweather** ein.

4 Klicken Sie mit der rechten Maustaste auf das Tag.

5 Wählen Sie im Kontextmenü „Tag bearbeiten: `cfweather`“ aus.

Wenn nun der Tag-Editor gestartet wird, wurde das Tag erfolgreich erstellt.

Hinzufügen eines Tags zur Tag-Auswahl

1 Bearbeiten Sie die Datei „TagChooser.xml“ im Ordner „Configuration/TagLibraries/CFML“ wie im folgenden Beispiel dargestellt, indem Sie eine neue Kategorie mit dem Namen „Third Party Tags“ hinzufügen, die das `cfweather`-Tag enthält:

```
<category name="Third Party Tags" icon="icons/Elements.gif" reference='CFML'>
    <element name="cfweather" value='cfweather zip="" temperaturescale="fahrenheit">' />
</category>
```

Hinweis: *Auf Mehrbenutzerplattformen befindet sich die Datei „TagChooser.xml“ auch im Konfigurationsordner des Benutzers. Weitere Informationen zu Mehrbenutzerplattformen finden Sie unter* *„Registrieren des Tags in der Tag-Bibliothek“ auf Seite 222.*

Als Nächstes überprüfen Sie, ob das `cfweather`-Tag nun in der Tag-Auswahl angezeigt wird.

2 Wählen Sie „Einfügen“ > „Tag“ aus.

3 Erweitern Sie die Gruppe „CFML-Tags“.

4 Wählen Sie am Ende der Tag-Auswahl die Gruppe „Third Party Tags“ aus. Das Tag `cfweather` wird im Listenfeld rechts angezeigt.

5 Wählen Sie `cfweather` aus und klicken Sie auf die Schaltfläche „Einfügen“.

Nun wird der Tag-Editor angezeigt.

API-Funktionen für den Tag-Editor

Zum Erstellen eines neuen Tag-Editors müssen Sie eine Implementierung für die Funktionen `inspectTag()`, `validateTag()` und `applyTag()` bereitstellen. Ein Implementierungsbeispiel finden Sie unter „Erstellen einer Benutzeroberfläche für den Tag-Editor“ auf Seite 223.

inspectTag()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Die Funktion wird beim ersten Anzeigen des Tag-Editors aufgerufen. Als Argument erhält die Funktion das Tag, das der Benutzer bearbeitet und das als `dom`-Objekt vorliegt. Die Funktion extrahiert Attributwerte aus dem bearbeiteten Tag 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 ein Benutzer auf einen Knoten in der Strukturansicht oder auf „OK“ klickt, überprüft die Funktion die Gültigkeit der Daten in den derzeit angezeigten HTML-Formularelementen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn die Eingaben für die HTML-Formularelemente gültig sind. `false`, wenn die Eingabewerte nicht gültig sind.

Beispiel
Der Benutzer erstellt eine Tabelle und gibt dabei für die Anzahl der Tabellenzeilen eine negative Ganzzahl ein. Die Funktion `validateTag()` erkennt, dass die Eingabe ungültig 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 (d. h. 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
Wenn der Benutzer im Beispiel `cfweather` im folgenden Code die Postleitzahl von 94065 in 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. Er ist unverzichtbar zum Definieren, Prüfen und Ändern des Namens, der Größe, der Darstellung und anderer Auswahlattribute. Er wird zudem zum Starten interner und externer Editoren für das jeweils ausgewählte Element verwendet.

Die folgende Tabelle enthält die Dateien zum Erstellen von Eigenschafteninspektoren:

Pfad	**Datei**	**Beschreibung**
Configuration/Inspectors/	*Name_des_Eigenschafteninspektors*.htm	Definiert die Benutzeroberfläche 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.

Dateien für Eigenschafteninspektoren

Dreamweaver verfügt über mehrere integrierte Benutzeroberflächen für den Eigenschafteninspektor, mit denen Sie die Eigenschaften für viele HTML-Standard-Tags festlegen können. Da diese integrierten Inspektoren Teil des Basiscodes von Dreamweaver sind, befinden sich keine entsprechenden Dateien für Eigenschafteninspektoren im Ordner „Configuration“. Durch benutzerdefinierte Dateien für Eigenschafteninspektoren können Sie diese integrierten Benutzeroberflächen jedoch ersetzen oder neue Benutzeroberflächen zum Prüfen benutzerdefinierter Tags erstellen. Benutzerdefinierte Dateien für Eigenschafteninspektoren befinden sich im Ordner „Configuration/Inspectors“ im Dreamweaver-Anwendungsordner.

Die HTML-Dateien für Eigenschafteninspektoren müssen zusätzlich zum doctype-Kommentar direkt vor dem öffnenden HTML-Tag einen Kommentar enthalten. Der Kommentar ist im folgenden Beispiel dargestellt:

```
<!-- tag:serverModel:tagNameOrKeyword,priority:1to10,selection:¬ exactOrWithin,hline,vline,
serverModel-->
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//pi">
```

Dieser Kommentar enthält folgende Elemente:

- Das Element *serverModel* gibt an, dass Dreamweaver diesen Eigenschafteninspektor nur laden darf, wenn das angegebene Servermodell aktiv ist.
- Das Element *tagNameOrKeyword* enthält das zu prüfende Tag oder eines der folgenden Schlüsselwörter: *COMMENT* (für Kommentare), *LOCKED* (für gesperrte Bereiche) oder *ASP* (für ASP-Tags).
- Das Element 1to10 stellt die Priorität der Datei für den Eigenschafteninspektor dar. 1 gibt an, dass dieser Eigenschafteninspektor nur verwendet wird, wenn kein anderer Eigenschafteninspektor diese Auswahl prüfen kann. 10 gibt an, dass dieser Eigenschafteninspektor Vorrang vor allen anderen für diese Auswahl hat.
- Das Element *exactOrWithin* gibt an, ob die aktuelle Auswahl innerhalb eines Tags liegen kann (within) oder genau das Tag enthalten muss (exact).
- Das optionale Element *hline* legt fest, dass in der erweiterten Darstellung eine waagerechte graue Linie zwischen der oberen und unteren Hälfte des Inspektors angezeigt werden soll.

- Das optionale Element *vline* gibt an, dass eine senkrechte graue Linie zwischen dem Tag-Namensfeld und den restlichen Eigenschaften im Inspektor angezeigt werden soll.
- Das optionale Element *serverModel* gibt das Servermodell des Eigenschafteninspektors an. Das Servermodell des Eigenschafteninspektors muss mit dem Servermodell des Dokuments übereinstimmen. Andernfalls verwendet Dreamweaver den Eigenschafteninspektor nicht, um die Eigenschaften der aktuellen Auswahl anzuzeigen. Angenommen, das Servermodell eines Dokuments ist z. B. Adobe ColdFusion. Das Servermodell des Eigenschafteninspektors ist jedoch ASP. Dreamweaver verwendet in diesem Fall nicht diesen Eigenschafteninspektor für eine Auswahl in dem Dokument.

Der folgende Kommentar eignet sich beispielsweise für einen Inspektor, der das Tag `happy` prüfen soll:

```
<!-- tag:happy, priority:8,selection:exact,hline,vline,serverModel:ASP -->
```

In einigen Fällen ist es erforderlich anzugeben, dass Ihre Erweiterung nur das Dreamweaver-Erweiterungsrendering verwenden soll und nicht die frühere Rendering-Engine. Dies ist möglich, indem Sie die folgende Zeile unmittelbar vor dem Tag-Kommentar einfügen, wie im folgenden Beispiel dargestellt:

```
<!--DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//pi"-->
```

Der body-Bereich einer Datei für Eigenschafteninspektoren enthält ein HTML-Formular. In Dreamweaver wird der Formularinhalt nicht in einem Dialogfeld angezeigt. Stattdessen werden mit dem Formular die Eingabebereiche und das Layout des Eigenschafteninspektors definiert.

Der head-Bereich einer Datei für Eigenschafteninspektoren enthält JavaScript-Funktionen oder einen Verweis auf die JavaScript-Dateien.

Funktionsweise von Dateien für Eigenschafteninspektoren

Beim Start liest Dreamweaver die erste Zeile aller HTM- und HTML-Dateien aus dem Ordner „Configuration/Inspectors“ und sucht dabei nach dem Kommentar-String für Typ, Priorität und Auswahltyp eines 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 setzt, werden die folgenden Ereignisse ausgelöst:

1. Dreamweaver sucht nach Inspektoren mit dem Auswahltyp `within`.
2. Wenn 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 für jeden gefundenen Inspektor die Funktion `canInspectSelection()` auf. Wenn diese Funktion den Wert `false` zurückgibt, wird der Inspektor nicht zur Prüfung des ausgewählten Bereichs herangezogen.
4. Wenn nach dem Aufrufen der Funktion `canInspectSelection()` mehrere potenzielle Inspektoren vorhanden sind, sortiert Dreamweaver die Inspektoren nach Priorität.
5. Wenn mehrere potenzielle Inspektoren die gleiche Priorität haben, wählt Dreamweaver einen Inspektor in alphabetischer Reihenfolge der Namen aus.
6. Der ausgewählte Inspektor wird im schwebenden Bedienfeld des Eigenschafteninspektors angezeigt. Wenn in der Datei für Eigenschafteninspektoren die Funktion `displayHelp()` definiert ist, wird in der rechten oberen Ecke des Inspektors ein kleines Fragezeichen (?) angezeigt.

7 Dreamweaver ruft die Funktion `inspectSelection()` auf, um Informationen zur aktuellen Auswahl zu sammeln und die Felder des Inspektors entsprechend zu füllen.

8 Die den Feldern des Eigenschafteninspektors zugeordneten Ereignisprozeduren werden ausgeführt, wenn der Benutzer sie auslöst. (So kann es beispielsweise ein `onBlur`-Ereignis geben, das die Funktion `setAttribute()` aufruft, um ein Attribut für den vom Benutzer eingegebenen Wert festzulegen.)

Einfaches Beispiel für einen Eigenschafteninspektor

Der folgende Eigenschafteninspektor prüft das `marquee`-Tag, das nur in Microsoft Internet Explorer verfügbar ist. In diesem Beispiel kann im Eigenschafteninspektor der Wert für das Attribut `direction` festgelegt werden. Verwenden Sie dieses Beispiel als Modell, wenn Sie die Werte der anderen Attribute des `marquee`-Tags festlegen möchten.

Um diese Erweiterung zu erstellen, müssen Sie die Benutzeroberfläche definieren, den JavaScript-Code verfassen, ein Bild erstellen und die Erweiterung testen.

Erstellen der Benutzeroberfläche

Sie erstellen eine HTML-Datei mit einem Formular, das im Eigenschafteninspektor angezeigt wird.

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 dem öffnenden und dem schließenden `body`-Tag ein, um den Inhalt anzugeben, der im Eigenschafteninspektor angezeigt wird.

```
<!-- Specify the image that will appear in the Property inspector -->
<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>

<!-- If your form fields are in different AP elements, you must ¬
    create a separate form inside each AP element and reference it as ¬
    shown in the inspectSelection() and setInterjectionTag() functions. -->

<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>
        </TR>
    </TABLE>
</FORM>
</SPAN>
```

5 Speichern Sie die Datei unter dem Namen „marquee.htm“ im Ordner „Configuration/Inspectors“.

Verfassen des JavaScript-Codes

Sie müssen JavaScript-Funktionen hinzufügen, um die Prüfmöglichkeit der Auswahl sicherzustellen, die Auswahl zu prüfen und die entsprechenden Werte im Eigenschafteninspektor einzugeben.

1 Erstellen Sie eine neue, leere Datei.

2 Um festzulegen, dass der Eigenschafteninspektor angezeigt wird, wenn die Auswahl das `marquee`-Tag enthält, fügen Sie 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(){
    // Get the DOM of the current document.
    var theDOM = dw.getDocumentDOM();
    // Get the selected node.
    var theObj = theDOM.getSelectedNode();

    // Get the value of the DIRECTION attribute on the MARQUEE tag.
    var theDirection = theObj.getAttribute('direction');

    // Initialize a variable for the DIRECTION attribute to -1.
    // This is used to store the menu index that corresponds to
    // the value of the attribute.
    // var typeIndex = -1;
    var directionIndex = -1;

    // If there was a DIRECTION attribute...
    if (theDirection){
        // If the value of DIRECTION is "left", set typeIndex to 0.
        if (theDirection.toLowerCase() == "left"){
            directionIndex = 0;
        // If the value of DIRECTION is "right", set typeIndex to 1.
            }else if (theDirection.toLowerCase() == "right"){
                directionIndex = 1;
                }
            }
    // If the value of the DIRECTION attribute was "left"
    // or "right", choose the corresponding
    // option from the pop-up menu in the interface.
    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(){
    // Get the DOM of the current document.
    var theDOM = dw.getDocumentDOM();
    // Get the selected node.
    var theObj = theDOM.getSelectedNode();

    // Get the index of the selected option in the pop-up menu
    // in the interface.
    var directionIndex = ¬
        document.topLayer.document.topLayerForm.marqDirection.selectedIndex;
    // Get the value of the selected option in the pop-up menu
    // in the interface.
    var theDirection = ¬
        document.topLayer.document.topLayerForm.marqDirection.¬
        options[directionIndex].value;

    // Set the value of the direction attribute to theDirection.
    theObj.setAttribute('direction',theDirection);
}
```

5 Speichern Sie die Datei unter dem Namen „marquee.js“ im Ordner „Configuration/Inspectors“.

Erstellen des Bilds

Sie haben die Möglichkeit, ein Bild zu erstellen, das im Eigenschafteninspektor angezeigt werden soll.

1 Erstellen Sie eine Grafik mit einer Breite von 36 Pixel und einer Höhe von 36 Pixel.

2 Speichern Sie das Bild unter dem Namen „marquee.gif“ im Ordner „Configuration/Inspectors“.

Im Allgemeinen können Sie Bilder für Eigenschafteninspektoren in jedem beliebigen von Dreamweaver unterstützten Format speichern.

Testen des Eigenschafteninspektors

Abschließend können Sie den Eigenschafteninspektor testen.

1 Starten Sie Dreamweaver neu.

2 Erstellen Sie eine neue HTML-Seite oder öffnen Sie eine vorhandene HTML-Seite.

3 Fügen Sie Folgendes in den body-Bereich der Seite ein:

```
<MARQUEE></MARQUEE>
```

4 Markieren Sie den soeben eingefügten Text.

Der Eigenschafteninspektor, den Sie für das `marquee`-Tag erstellt haben, wird angezeigt.

5 Geben Sie im Eigenschafteninspektor einen Wert für das direction-Attribut ein.

Das Tag auf der Seite enthält nun das direction-Attribut und den im Eigenschafteninspektor eingegebenen Wert.

API-Funktionen für den Eigenschafteninspektor

Zwei der API-Funktionen für den Eigenschafteninspektor (`canInspectSelection()` und `inspectSelection()`) sind zwingend 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 abzurufen. (Weitere Informationen zu* `dom.getSelectedNode()` *finden Sie im Dreamweaver API-Referenzhandbuch.)*

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn mit dem Inspektor die aktuelle Auswahl geprüft werden kann, andernfalls `false`.

Beispiel

Die folgende Instanz der Funktion `canInspectSelection()` gibt den Wert `true` zurück, wenn die Auswahl das Attribut `CLASSID` enthält und der Attributwert `"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"` lautet (die Klassen-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 diese Funktion definiert ist, wird in der rechten oberen Ecke des Eigenschafteninspektors ein 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 für die `displayHelp()`-Funktion 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

maxOrMin

- Das Argument *maxOrMin* ist entweder `max` oder `min`, je nachdem, ob der Inspektor in der erweiterten oder reduzierten Darstellung angezeigt wird.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Im folgenden Beispiel für die Funktion `inspectSelection()` wird der Wert des Attributs `content` abgerufen und damit ein Formularfeld mit dem Namen `keywords` gefüllt.

```
function inspectSelection(){
    var dom = dreamweaver.getDocumentDOM();
    var theObj = dom.getSelectedNode();
    document.forms[0].keywords.value = theObj.getAttribute("content");
}
```

Kapitel 15: Schwebende Bedienfelder

Sie können schwebende Bedienfelder oder Inspektoren erstellen, ohne an die Größen- und Layoutbeschränkungen von Eigenschafteninspektoren gebunden zu sein.

In der Regel sollten Sie die Eigenschaften der aktuellen Auswahl mit einem benutzerdefinierten Eigenschafteninspektor festlegen. Mit benutzerdefinierten schwebenden Bedienfeldern können Sie jedoch Informationen zum gesamten Dokument oder zu mehreren Auswahlbereichen flexibler und umfassender anzeigen.

Sie können benutzerdefinierte Bedienfelder erstellen und dem Menü „Fenster“ hinzufügen. Weitere Informationen zum Hinzufügen von Elementen zum Menüsystem finden Sie unter „Menüs und Menübefehle“ auf Seite 153.

In der folgenden Tabelle sind die Dateien zum Erstellen von schwebenden Bedienfeldern aufgeführt:

Pfad	Datei	Beschreibung
Configuration/Floaters/	*panelname*.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

Die Dateien für benutzerdefinierte schwebende Bedienfelder sind HTML-Dateien. Sie befinden sich im Ordner „Configuration/Floaters“ des Anwendungsordners. Der `body`-Bereich einer Datei für ein schwebendes Bedienfeld enthält ein HTML-Formular. An Formularelemente angefügte Ereignisprozeduren können JavaScript-Code aufrufen, mit dem das aktuelle Dokument beliebig bearbeitet werden kann.

Dreamweaver enthält mehrere schwebende Bedienfelder, die Sie über das Menü „Fenster“ aufrufen können. (Diese integrierten Bedienfelder sind Bestandteil des Kernprogramms von Dreamweaver. Es befinden sich keine entsprechenden Dateien im Ordner „Configuration/Floaters“.)

Benutzerdefinierte schwebende Bedienfelder können wie die in Dreamweaver integrierten schwebenden Bedienfelder verschoben, in der Größe geändert und in Gruppen zusammengefasst werden. Benutzerdefinierte schwebende Bedienfelder unterscheiden sich jedoch in folgender Hinsicht von den in Dreamweaver integrierten schwebenden Bedienfeldern:

- Benutzerdefinierte schwebende Bedienfelder werden immer grau angezeigt. Das Festlegen des Attributs `bgcolor` im `body`-Tag hat keine Auswirkung auf die Farbe.
- Alle benutzerdefinierten schwebenden Bedienfelder werden entweder vor dem Dokumentfenster angezeigt oder befinden sich hinter dem Dokumentfenster, wenn sie inaktiv sind. Wo sie angezeigt werden, hängt von der Einstellung „Alle anderen Fenster“ in den Voreinstellungen für schwebende Bedienfelder ab.

Dateien für schwebende Bedienfelder unterscheiden sich in einigen Aspekten von anderen Erweiterungen. Im Gegensatz zu anderen 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 die schwebenden Bedienfelder beim Beenden von Dreamweaver nicht sichtbar waren, erstellen Sie einen Verweis mit einer der folgenden Funktionen, um die Dateien zu laden, die die Bedienfelder definieren.

- `dreamweaver.getFloaterVisibility()`

- `dreamweaver.setFloaterVisibility()`
- `dreamweaver.toggleFloater()`

Weitere Informationen zu diesen 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 Aktionen ausgeführt:

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()` im *Dreamweaver API-Referenzhandbuch*. Wenn die Datei „floaterName.htm“ nicht gefunden wird, sucht Dreamweaver nach der Datei „floaterName.html“. Wenn auch diese Suche ergebnislos bleibt, wird kein Vorgang durchgeführt.
2. Beim ersten Laden der Datei für ein schwebendes Bedienfeld wird die Funktion `initialPosition()` aufgerufen, sofern sie definiert ist, um die Standardposition des schwebenden Bedienfelds auf dem Bildschirm festzulegen. Zudem wird die Funktion `initialTabs()` aufgerufen, sofern sie definiert ist, um die Standard-Registerkartengruppierung des schwebenden Bedienfelds festzulegen.
3. Die Funktionen `selectionChanged()` und `documentEdited()` werden aufgerufen, da sich aller Wahrscheinlichkeit nach Änderungen ergeben haben, während das schwebende Bedienfeld ausgeblendet war.
4. Wenn das schwebende Bedienfeld sichtbar ist, werden folgende Aktionen ausgeführt:
   - Bei einer Änderung der Auswahl wird die Funktion `selectionChanged()` aufgerufen, sofern sie definiert ist.
   - Wenn der Benutzer Änderungen am Dokument vornimmt, wird die Funktion `documentEdited()` aufgerufen, sofern sie definiert ist.
   - Die den Felder des schwebenden Bedienfelds zugeordneten Ereignisprozeduren werden ausgeführt, wenn der Benutzer sie auslöst. (Beim Klicken auf eine Schaltfläche mit einer `onClick`-Ereignisprozedur, die die Funktion `dw.getDocumentDOM().body.innerHTML=''` ausführt, wird beispielsweise der gesamte Bereich zwischen dem öffnenden und dem schließenden `body`-Tag im Dokument gelöscht.)

     Schwebende Bedienfelder unterstützen zwei spezielle Ereignisse für das `body`-Tag: `onShow()` und `onHide()`.
5. Beim Beenden von Dreamweaver werden die aktuelle Sichtbarkeit, Position und Registerkartengruppierung 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 mit der zuletzt gültigen Registerkartengruppierung angezeigt.

Einfaches Beispiel für schwebende Bedienfelder

In diesem Beispiel erstellen Sie die Erweiterung „Script Editor“ zum Erstellen von schwebenden Bedienfeldern, in denen der einer ausgewählten Skriptmarkierung zugrunde liegende JavaScript-Code in der Entwurfsansicht angezeigt wird. Die Erweiterung „Script Editor“ zeigt den JavaScript-Code im Element `textarea` eines HTML-Formulars an, das im schwebenden Bedienfeld `scriptlayer` definiert ist. Wenn Sie den ausgewählten Code im schwebenden Bedienfeld ändern, ruft die Erweiterung die Funktion `updateScript()` auf, um die Änderungen zu speichern. Wenn Sie beim Aufrufen von „Script Editor“ keine Skriptmarkierung ausgewählt haben, zeigt die Erweiterung den Text `(no script selected)` im schwebenden Bedienfeld `blanklayer` an.

Zum Erstellen dieser Erweiterung erstellen Sie schwebende Bedienfelder, programmieren JavaScript-Code und erstellen ein Menüelement.

Erstellen der schwebenden Bedienfelder

Oben in der HTML-Datei dieser Erweiterung sind die Header-Standardinformationen für das Dokument sowie ein `title`-Tag angegeben, das die Bezeichnung „Script Editor“ in die Titelleiste der schwebenden Bedienfelder einfügt.

Erstellen des Headers der HTML-Datei

1 Erstellen Sie ein neues, leeres Dokument.

2 Geben Sie folgenden Code ein:

```
<!doctype html public "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Script Editor</title>
<script language="JavaScript">
```

Die Erweiterung definiert zwei schwebende Bedienfelder, die entweder den Text `(no script selected)` anzeigen, wenn der Benutzer keine Skriptmarkierung ausgewählt hat, oder den JavaScript-Code, der der ausgewählten Skriptmarkierung zugrunde liegt. Mit dem folgenden Code werden die beiden schwebenden Bedienfelder bzw. absolut positionierten Elemente (AP-Elemente) `blanklayer` und `scriptlayer` definiert.

Erstellen der beiden schwebenden Bedienfelder

1 Fügen Sie den folgenden Code nach dem Header in die HTML-Datei ein:

```
<body>
<div id="blanklayer" style="position:absolute; width:422px; ¬
height:181px; z-index:1; left: 8px; top: 11px; ¬
visibility: hidden">
<center>
<br>
<br>
<br>
<br>
<br>
(no script selected)
</center>
</div>

<div id="scriptlayer" style="position:absolute; width:422px; ¬
height:181px; z-index:1; left: 8px; top: 11px; ¬
visibility: visible">
<form name="theForm">
<textarea name="scriptCode" cols="80" rows="20" wrap="VIRTUAL" ¬
onBlur="updateScript()"></textarea>
</form>
</div>

</body>
</html>
```

2 Speichern Sie die Datei unter dem Namen „scriptEditor.htm“ im Ordner „Configuration/Floaters“.

In beiden `div`-Tags wird mit dem Attribut `style` die Position (`absolute`), Größe (`width:422px` und `height:181px`) und die `visibility`-Standardeinstellung (`visible`) der schwebenden Bedienfelder angegeben. Das Bedienfeld `blanklayer` verwendet das Attribut `center` und mehrere Zeilenumbruch-Tags (`br`), um den Text in der Mitte des Bedienfelds zu positionieren. Das Bedienfeld `scriptlayer` erstellt ein Formular mit einem einzelnen `textarea`-Tag, um den ausgewählten JavaScript-Code anzuzeigen. Mit dem `textarea`-Tag wird darüber hinaus festgelegt, dass beim Auslösen eines `onBlur`-Ereignisses nach einer Änderung des ausgewählten Codes die Funktion `updateScript()` aufgerufen wird, um den geänderten Text wieder in das Dokument zu schreiben.

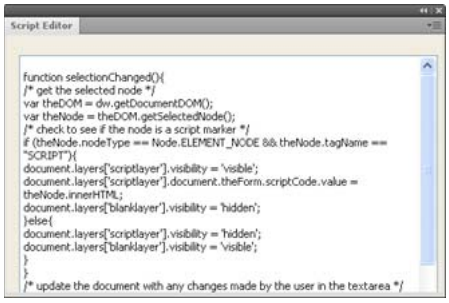
Programmieren des JavaScript-Codes

Der JavaScript-Code für die Erweiterung „Script Editor“ besteht aus der von Dreamweaver aufgerufenen Funktion für schwebende Bedienfelder `selectionchanged()` und der benutzerdefinierten Funktion `updateScript()`.

selectionChanged(): Skriptmarkierung ausgewählt?

Die Funktion `selectionChanged()` ermittelt, ob in der Entwurfsansicht eine Skriptmarkierung ausgewählt wurde. Eine Skriptmarkierung wird in der Entwurfsansicht angezeigt, wenn eine JavaScript-Routine im body-Bereich eines Dokuments vorhanden ist und wenn im Bereich „Unsichtbare Elemente“ des Dialogfelds „Voreinstellungen“ die Option „Skripts“ ausgewählt ist. In der folgenden Abbildung ist eine Skriptmarkierung dargestellt:

Die Funktion `selectionChanged()` ruft zunächst 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 und das Bedienfeld `scriptlayer` ausgeblendet ist. Das schwebende Bedienfeld `blanklayer` zeigt den Text `(no script selected)` wie in der folgenden Abbildung dargestellt an:

Hinzufügen der selectionChanged()-Funktion

1 Öffnen Sie die Datei „scriptEditor.htm“ im Ordner „Configuration/Floaters“.

2 Geben Sie im Header der Datei den folgenden Code ein.

```
function selectionChanged(){
    /* get the selected node */
    var theDOM = dw.getDocumentDOM();
    var theNode = theDOM.getSelectedNode();

    /* check to see if the node is a script marker */
    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, wenn ein `onBlur`-Ereignis im `textarea`-Element des Bedienfelds `scriptlayer` ausgelöst wird. Das Formularelement `textarea` enthält den JavaScript-Code. Das `onBlur`-Ereignis wird ausgelöst, wenn `textarea` den Eingabefokus verliert.

1 Öffnen Sie die Datei „scriptEditor.htm“ im Ordner „Configuration/Floaters“.

2 Geben Sie im Header der Datei den folgenden Code ein.

```
/* update the document with any changes made by
    the user in the textarea */

    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.

Erstellen eines Menüelements

Es genügt nicht, den Code für „Script Editor“ im Ordner „Configuration/Floaters“ zu speichern. Sie müssen zudem die Funktion `dw.setFloaterVisibility('scriptEditor',true)` oder die Funktion `dw.toggleFloater('scriptEditor')` aufrufen, um das schwebende Bedienfeld zu laden und sichtbar zu machen. Die Erweiterung „Script Editor“ kann am besten über das Menü „Fenster“ abgerufen werden, 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 (siehe folgende Abbildung).

Wenn Sie für das aktuelle Dokument zunächst eine Skriptmarkierung 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 der Skriptmarkierung zugrunde liegt. Wenn Sie das Menüelement auswählen, ohne zuvor eine Skriptmarkierung auszuwählen, zeigt Dreamweaver das Bedienfeld `blanklayer` mit dem Text `(no script selected)` an.

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 folgenden Code in eine neue Zeile ein:

```
<menuitem name="Script Editor" enabled="true" ¬
command="dw.toggleFloater('scriptEditor')"¬
checked="dw.getFloaterVisibility('scriptEditor')" />
```

4 Speichern Sie die Datei.

API-Funktionen für schwebende Bedienfelder

Alle benutzerdefinierten 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, wird jeweils in der Beschreibung der einzelnen Funktionen angegeben.

displayHelp()

Beschreibung
Wenn diese Funktion definiert ist, wird im Dialogfeld unter den Schaltflächen „OK“ und „Abbrechen“ die Schaltfläche „Hilfe“ angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche „Hilfe“ klickt.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// the following instance of displayHelp() opens
// in a browser a file that explains how to use
// the extension.
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/superDuperHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

documentEdited()

Beschreibung
Diese Funktion wird aufgerufen, wenn das schwebende Bedienfeld sichtbar wird und nachdem die aktuellen Bearbeitungsvorgänge abgeschlossen sind (d. h. vor dem Aufrufen dieser Funktion werden möglicherweise mehrere Bearbeitungsvorgänge durchgeführt). Diese Funktion sollte nur definiert werden, wenn das schwebende Bedienfeld Änderungen am Dokument verfolgen soll.

Hinweis: *Da* `documentEdited()` *die Systemleistung beeinträchtigt, sollte diese Funktion nur definiert werden, wenn sie unbedingt erforderlich ist.*

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel
Im folgenden Beispiel für die Funktion `documentEdited()` wird das Dokument nach AP-Elementen durchsucht und ein Textfeld aktualisiert, in dem die Anzahl der AP-Elemente im Dokument angezeigt wird.

```
function documentEdited(){
    /* create a list of all the AP elements in the document */
    var theDOM = dw.getDocumentDOM();
    var layersInDoc = theDOM.getElementsByTagName("layer");
    var layerCount = layersInDoc.length;
    /* update the numOfLayers field with the new layercount */
    document.theForm.numOfLayers.value = layerCount;
}
```

getDockingSide()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Gibt die Positionen an, an denen ein schwebendes Bedienfeld andocken kann. Die Funktion gibt einen String zurück, der eine Kombination der Wörter "`left`", "`right`", "`top`" und "`bottom`" enthält. Wenn die Bezeichnung im String angegeben ist, können Sie ein schwebendes Bedienfeld an dieser Seite andocken. Wenn die Funktion fehlt, ist ein Andocken schwebender Bedienfelder an keiner Seite möglich.

Mithilfe dieser Funktion können Sie verhindern, dass Bedienfelder an einer bestimmten Seite des Dreamweaver-Arbeitsbereichs oder aneinander andocken.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String mit den Wörtern "`left`", "`right`", "`top`" und "`bottom`" oder eine Kombination dieser Werte, mit dem festgelegt wird, an welcher Position das schwebende Bedienfeld angedockt werden kann.

Beispiel

```
getDockingSide()
{
    return dock_side = "left top";
}
```

initialPosition()

Beschreibung
Legt die Anfangsposition des schwebenden Bedienfelds beim ersten Aufruf fest. Wenn diese Funktion nicht definiert wurde, ist die Standardposition die Mitte des Bildschirms.

Argumente
platform

- Das Argument *platform* hat je nach Benutzerplattform entweder den Wert "`Mac`" oder "`Win`".

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 unter Windows 420 Pixel vom linken und 20 Pixel vom oberen Bildschirmrand entfernt angezeigt wird. Für den Macintosh soll das Bedienfeld 390 Pixel vom linken und 20 Pixel vom oberen Bildschirmrand entfernt angezeigt werden.

```
function initialPosition(platform){
    var initPos = "420,20";
    if (platform == "macintosh"){
        initPos = "390,20";
    }
    return initPos;
}
```

initialTabs()

Beschreibung

Legt fest, welche anderen schwebenden Bedienfelder zusammengefasst sind, wenn dieses schwebende Bedienfeld zum ersten Mal angezeigt wird. Wenn eines der aufgeführten Bedienfelder bereits zuvor angezeigt wurde, wird es nicht in die Registerkartengruppe aufgenommen. Um sicherzustellen, dass zwei benutzerdefinierte schwebende Bedienfelder zusammengefasst werden, sollte jedes Bedienfeld mit der Funktion `initialTabs()` jeweils auf das andere verweisen.

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet einen String der Form `"floaterName1,floaterName2,...floaterNameN"`.

Beispiel

Im folgenden Beispiel für die Funktion `initialTabs()` wird angegeben, dass das schwebende Bedienfeld beim ersten Aufruf mit dem schwebenden Bedienfeld „scriptEditor“ zusammengefasst wird.

```
function initialTabs(){
    return "scriptEditor";
}
```

isATarget()

Verfügbarkeit

Dreamweaver MX (nur Windows).

Beschreibung

Gibt an, ob andere schwebende Bedienfelder an diesem schwebenden Bedienfeld andocken können. Wenn die Funktion `isATarget()` nicht definiert ist, verhindert Dreamweaver, dass andere Bedienfelder an dieses Bedienfeld angedockt werden. Dreamweaver ruft diese Funktion auf, wenn der Benutzer versucht, dieses Bedienfeld mit anderen Bedienfeldern zusammenzufassen.

Argumente

Keine.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert: `true`, wenn andere schwebende Bedienfelder an dieses Bedienfeld angedockt werden können, andernfalls `false`.

Beispiel
```
isATarget()
{
    return true;
}
```

isAvailableInCodeView()

Beschreibung
Legt fest, ob das schwebende Bedienfeld in der Codeansicht aktiviert werden soll. Wenn diese Funktion nicht definiert wurde, ist das schwebende Bedienfeld in der Codeansicht deaktiviert.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert: `true`, wenn das schwebende Bedienfeld in der Codeansicht aktiviert werden soll, andernfalls `false`.

isResizable()

Verfügbarkeit
Dreamweaver 4.

Beschreibung
Legt fest, ob der Benutzer die Größe eines schwebenden Bedienfelds ändern kann. Wenn die Funktion nicht definiert ist oder den Wert `true` zurückgibt, kann der Benutzer die Größe des schwebenden Bedienfelds ändern. Wenn die Funktion den Wert `false` zurückgibt, kann der Benutzer die Größe des schwebenden Bedienfelds nicht anpassen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert: `true`, wenn der Benutzer die Größe des schwebenden Bedienfelds ändern kann, andernfalls `false`.

Beispiel
Im folgenden Beispiel kann der Benutzer die Größe des schwebenden Bedienfelds nicht ändern:

```
function isResizable()
{
    return false;
}
```

selectionChanged()

Beschreibung
Wird aufgerufen, wenn das schwebende Bedienfeld sichtbar wird und sich die Auswahl ändert (wenn der Fokus zu einem neuen Dokument wechselt oder die Einfügemarke an eine andere Stelle im aktuellen Dokument verschoben wird). Diese Funktion sollte nur definiert werden, wenn das schwebende Bedienfeld die Auswahl verfolgen soll.

Hinweis: *Da `selectionChanged()` die Systemleistung beeinträchtigt, sollte diese Funktion nur definiert werden, wenn sie unbedingt erforderlich ist.*

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel
Im folgenden Beispiel für `selectionChanged()` wird abhängig davon, ob es sich bei der Auswahl um eine Skriptmarkierung oder ein anderes Element handelt, im schwebenden Bedienfeld das entsprechende AP-Element angezeigt. Wenn die Auswahl eine Skriptmarkierung ist, wird in Dreamweaver das AP-Element `scriptlayer` angezeigt. Andernfalls wird das AP-Element `blanklayer` angezeigt.

```
function selectionChanged(){
    /* get the selected node */
        var theDOM = dw.getDocumentDOM();
        var theNode = theDOM.getSelectedNode();
    /* check to see if the node is a script marker */
        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';
        }
}
```

Programmleistung

Wenn die Funktionen `selectionChanged()` oder `documentEdited()` in benutzerdefinierten schwebenden Bedienfeldern deklariert sind, hat dies möglicherweise negative Auswirkungen auf die Leistung von Dreamweaver. Beachten Sie, dass die Funktionen `documentEdited()` und `selectionChanged()` nach jedem Tastendruck und jedem Mausklick aufgerufen werden, wenn in Dreamweaver länger als eine Zehntelsekunde keine Bearbeitung durchgeführt wird. Beim Testen eines schwebenden Bedienfelds sollten Sie auf jeden Fall verschiedene Szenarios und auch große HTML-Dokumente von mindestens 100 KB verwenden, um die Auswirkung auf die Systemleistung zu prüfen.

Um Leistungseinbußen zu vermeiden, sollten Sie die Funktion `setTimeout()` verwenden. Wie für Browser erwartet die Funktion `setTimeout()` zwei Argumente: den aufzurufenden JavaScript-Code und die Wartezeit vor dem Aufruf in Millisekunden.

Mithilfe der Methode `setTimeout()` können Sie Pausen in die Verarbeitung integrieren. Während dieser Pausen kann der Benutzer weiterhin interaktiv in der Anwendung arbeiten. Sie müssen diese Pausen explizit integrieren, da während der Skriptverarbeitung 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 programmiert, in dem Informationen zu den einzelnen AP-Elementen des Dokuments angezeigt werden. Mithilfe der `setTimeout()`-Methode wird nach dem Verarbeiten jedes AP-Elements eine Pause von einer halben Sekunde eingefügt.

```
/* create a flag that specifies whether an edit is being processed, and set it to false.*/
document.running = false;
/* this function called when document is edited */
function documentEdited(){
    /* create a list of all the AP elements to be processed */
    var dom = dw.getDocumentDOM();
    document.layers = dom.getElementsByTagName("layer");
    document.numLayers = document.layers.length;
    document.numProcessed = 0;
    /* set a timer to call processLayer(); if we didn't get
     * to finish processing the previous edit, then the timer
     * is already set. */
    if (document.running = false){
        setTimeout("processLayer()", 500);
    }
    /* set the processing flag to true */
    document.running = true;
}
/* process one AP element*/
function processLayer(){
    /* display information for the next unprocessed AP element.
        displayLayer() is a function you would write to
        perform the "magic".0*/
     displayLayer(document.layers[document.numProcessed]);
    /* if there's more work to do, set a timeout to process
     * the next layer.0.If we're finished, set the document.running
     * flag to false. */
    document.numProcessed = document.numProcessed + 1;
    if (document.numProcessed < document.numLayers){
        setTimeout("processLayer()", 500);
    }else{
        document.running = false;
    }
}
```

Kapitel 16: Verhalten

Der Begriff *Verhalten* bezieht sich auf die Kombination eines Ereignisses und einer Aktion. `onClick`, `onLoad` und `onSubmit` sind Beispiele für Ereignisse. „Plug-In überprüfen“, „Gehe zu URL“ und „Bild austauschen“ sind dagegen Beispiele für Aktionen. 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 Adobe Dreamweaver

Der body-Bereich einer Aktionsdatei enthält in der Regel ein HTML-Formular. Das HTML-Formular nimmt Parameter für die Aktion an (z. B. Parameter, mit denen angegeben wird, welche absolut positionierten Elemente angezeigt oder ausgeblendet werden sollen). Der head-Bereich einer Aktionsdatei enthält JavaScript-Funktionen, mit denen Formulareingaben aus dem body-Inhalt verarbeitet werden. Die Funktionen steuern zudem Funktionen, Argumente und Ereignisprozeduren, die in das Dokument eines Benutzers eingefügt werden.

Programmieren Sie Verhaltensaktionen, wenn Sie Funktionen auch Benutzern zur Verfügung stellen möchten oder wenn Sie dieselbe JavaScript-Funktion mehrmals einfügen möchten. Ändern Sie dabei jeweils die Parameter.

Hinweis: *Es ist nicht möglich, VBScript-Funktionen mithilfe von Verhalten direkt einzufügen. Sie können eine VBScript-Funktion jedoch indirekt einfügen. Bearbeiten Sie dazu das Dokumentobjektmodell (DOM) in der Funktion* `applyBehavior()`.

Im Folgenden sind die Dateien zum Erstellen von Verhaltensaktionen aufgeführt.

Pfad	Datei	Beschreibung
Configuration/Behaviors/Actions/	*Verhaltensaktion*.htm	Der body-Bereich der Datei enthält ein HTML-Formular für die Parameter der Aktion. Der head-Bereich der Datei enthält die JavaScript-Funktionen.

Hinweis: *Informationen zu Serververhalten mit Funktionen für Webanwendungen finden Sie unter „Serververhalten“ auf Seite 262.*

Funktionsweise von Verhalten

Wenn ein Benutzer ein HTML-Element in einem Dreamweaver-Dokument auswählt und im Bedienfeld „Verhalten“ auf die Schaltfläche mit dem Pluszeichen (+) klickt, werden die folgenden Ereignisse ausgelöst:

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.

   Wenn für diese Funktion der Wert `false` zurückgegeben wurde, wird die Aktion im Popupmenü „Aktionen“ abgeblendet angezeigt. (Die Aktion „Shockwave steuern“ wird beispielsweise abgeblendet, wenn das Dokument des Benutzers keine SWF-Dateien enthält.) Wenn eine Liste mit Ereignissen zurückgegeben wird, vergleicht Dreamweaver jedes Ereignis mit den gültigen Ereignissen des ausgewählten HTML-Elements und des Zielbrowsers, bis eine Übereinstimmung gefunden wird. 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 im Popupmenü „Aktionen“ eine Aktion aus.

3 Dreamweaver ruft die Funktion `windowDimensions()` auf, um die Größe des Dialogfelds „Parameter“ festzulegen. Wenn die Funktion `windowDimensions()` nicht definiert ist, wird die Größe automatisch festgelegt.

   Unabhängig vom Inhalt des `body`-Elements wird stets ein Dialogfeld angezeigt, das auf der rechten Seite die 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 mit den einzelnen Formularfeldern verknüpften Ereignisprozeduren aus, wenn 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 Dokument des Benutzers 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. Anschließend ruft Dreamweaver die Funktion `inspectBehavior()` in der ausgewählten Aktionsdatei auf, die die Felder mit den vom Benutzer zuvor eingegebenen Daten füllt.

Einfügen mehrerer Funktionen in die Datei des Benutzers

Aktionen können mehrere Funktionen, d. h. die Hauptverhaltensfunktion sowie eine beliebige Anzahl von Hilfsfunktionen, in den head-Bereich einfügen. Mehrere Verhalten können zudem Hilfsfunktionen gemeinsam verwenden, sofern die Funktionsdefinition in allen Aktionsdateien genau gleich ist. Eine Möglichkeit um sicherzustellen, dass gemeinsam verwendete Funktionen identisch sind, besteht darin, die einzelnen Hilfsfunktionen in einer externen JavaScript-Datei zu speichern und mithilfe von `<SCRIPT SRC="externe_Datei.js">` in die entsprechenden Aktionsdateien einzufügen.

Wenn der Benutzer ein Verhalten löscht, versucht Dreamweaver, alle nicht verwendeten Hilfsfunktionen zu entfernen, die mit diesem Verhalten verknüpft sind. Wird eine Hilfsfunktion auch von anderen Verhalten verwendet, wird diese nicht gelöscht. Da der Algorithmus zum Löschen von Hilfsfunktionen sehr konservativ ausgelegt ist, wird gelegentlich eine nicht verwendete Hilfsfunktion im Dokument des Benutzers nicht gelöscht.

Aktionen mit obligatorischem Rückgabewert

In einigen Fällen muss eine Ereignisprozedur einen Rückgabewert haben (z. B. `onMouseOver="window.status='This is a link'; return true"`). Wenn Dreamweaver jedoch die Aktion `"return behaviorName(args)"` in die Ereignisprozedur einfügt, werden Verhalten weiter unten 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 `return document.MM_returnValue` am Ende der Liste der Aktionen in die Ereignisprozedur eingefügt. In der Datei „Validate Form.js“ finden Sie ein Beispiel, in dem die Variable `MM_returnValue` verwendet wird. Die Datei befindet sich im Unterordner „Configuration/Behaviors/Actions“ des Dreamweaver-Anwendungsordners.

Einfaches Beispiel für Verhalten

Mit dem folgenden Beispiel werden die Funktionsweise und das Erstellen von Verhalten veranschaulicht. Im Unterordner „Configuration/Behaviors/Actions“ des Dreamweaver-Anwendungsordners befinden sich Beispiele, die jedoch teilweise sehr komplex sind. Dieses Beispiel ist einfacher und besser geeignet, das Erstellen von Verhalten zu erläutern. Beginnen Sie mit der einfachen Aktionsdatei „Call JavaScript.htm“ (zusammen mit der Datei „Call JavaScript.js“, die alle JavaScript-Funktionen enthält).

Zum Erstellen eines Verhaltens erstellen Sie eine Erweiterung und die aufzurufenden HTML-Dateien und testen dann das Verhalten.

Erstellen der Verhaltenserweiterung

Der folgende Code ist ein relativ einfaches Beispiel. Mit diesem Code wird der Browser geprüft. Es wird eine Seite geöffnet, wenn es sich beim Browser um Netscape Navigator handelt, und eine andere Seite, wenn als Browser Microsoft Internet Explorer verwendet wird. Dieser Code kann problemlos erweitert werden, damit auch andere Browser wie Opera und WebTV erkannt werden, und kann so bearbeitet werden, dass eine andere Aktion als das Wechseln zu URLs ausgeführt wird.

1 Erstellen Sie eine neue, leere Datei.

2 Fügen Sie folgenden Code in die Datei ein:

```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//dialog">
<html>
<head>
<title>behavior "Check Browser Brand"</title>
<meta http-equiv="Content-Type" content="text/html">
<script language="JavaScript">

// The function that will be inserted into the
// HEAD of the user's document
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;
}

// Return the name of the function to be inserted into
// the HEAD of the user's document
function behaviorFunction(){
        return "checkBrowserBrand";
}

// Create the function call that will be inserted
// with the event handler
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."
    }
}

// Extract the arguments from the function call
// in the event handler and repopulate the
// parameters form
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;
}

//***************** LOCAL FUNCTIONS ******************

// Put the pointer in the first text field
// and select the contents, if any
function initializeUI(){
    document.theForm.nsURL.focus();
    document.theForm.nsURL.select();
}

// Let the user browse to the Navigator and
// IE URLs
function browseForURLs(whichButton){
    var theURL = dreamweaver.browseForFileURL();
    if (whichButton == "nsURL"){
        document.theForm.nsURL.value = theURL;
    }else{
        document.theForm.ieURL.value = theURL;
    }
}

//*************** END OF JAVASCRIPT *****************
</script>
</head>
<body>
```

```
<form method="post" action="" name="theForm">
<table border="0" cellpadding="8">
<tr>
<td nowrap="nowrap">  Go to this URL if the browser is ¬
    Netscape Navigator:<br>
<input type="text" name="nsURL" size="50" value="">  
<input type="button" name="nsBrowse" value="Browse..." ¬
    onClick="browseForURLs('nsURL')"></td>
</tr>
<tr>
<td nowrap="nowrap">  Go to this URL is the browser is ¬
    Microsoft Internet Explorer:<br>
<input type="text" name="ieURL" size="50" value="">  
<input type="button" name="ieBrowse" value="Browse..." ¬
    onClick="browseForURLs('ieURL')"></td>
</tr>
</table>
</form>
</body>
</html>
```

3 Speichern Sie die Datei unter dem Namen „BrowserDependentURL.htm“ im Ordner „Configuration/Behaviors/Actions“.

Erstellen der aufzurufenden HTML-Dateien (veraltet)

Erstellen Sie die für Internet Explorer aufzurufende HTML-Datei sowie die für Netscape Navigator aufzurufende HTML-Datei.

1 Erstellen Sie eine 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 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 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 im Bedienfeld „Verhalten“ auf die Schaltfläche mit dem Pluszeichen (+) und wählen Sie das Verhalten „Browser überprüfen“ aus.

9 Klicken Sie auf die Schaltfläche „Durchsuchen“ neben der Option „Gehe zu URL“ und wählen Sie die Datei „netscapecontent.htm“ aus, wenn Netscape Navigator als Browser verwendet wird. Wählen Sie für Internet Explorer die Datei „iecontent.htm“ aus.

10 Klicken Sie auf „OK“.

Dreamweaver fügt den angegebenen JavaScript-Code in der Datei „whichbrowser.htm“ ein. 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('netscaptecontent.htm','iecontent.htm')">
</body>
</html>
```

Testen des Verhaltens

1 Öffnen Sie die Datei „whichbrowser.htm“ in Ihrem Browser.

2 Je nach dem verwendeten Browser wird entweder „iecontent.htm“ oder „netscapecontent.htm“ angezeigt.

API-Funktionen für Verhalten

Zwei API-Funktionen für Verhalten sind obligatorisch (`applyBehavior()` und `behaviorFunction()`). Die anderen Funktionen sind optional.

applyBehavior()

Beschreibung
Diese Funktion fügt in das Benutzerdokument eine Ereignisprozedur ein, die die von `behaviorFunction()` eingefügte Funktion aufruft. Die Funktion `applyBehavior()` kann im Benutzerdokument auch weitere Bearbeitungsschritte ausführen, darf das Objekt jedoch nicht löschen, auf das das Verhalten angewendet wird bzw. das die Aktion empfängt.

Beim Programmieren der Funktion `applyBehavior()` müssen Sie festlegen, wie das Dokument des Benutzers bearbeitet werden soll. Sie können beispielsweise angeben, dass Code in die script-Tags im body-Bereich des Dokuments eingefügt wird. Sie können dies mithilfe der Standard-APIs für DOM-Bearbeitungen angeben.

Argumente
uniqueName

Das Argument ist eine ID, die unter allen Instanzen aller Verhalten im Dokument des Benutzers eindeutig ist. Das Argument wird im Format *functionNameInteger* angegeben, 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 dem zugehörigen Attribut `NAME` einen eindeutigen Wert zuweisen möchten.

Rückgabewerte

Dreamweaver erwartet in der Regel einen String mit dem Funktionsaufruf, der in das Benutzerdokument eingefügt werden soll, 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. Wenn der String leer ist (`return "";`), wird keine Fehlermeldung ausgegeben. Wenn der String jedoch weder leer noch ein Funktionsaufruf ist, wird ein Dialogfeld mit dem Text *Die Eingabe für dieses Verhalten ist ungültig:* und dem von `applyBehavior()` zurückgegebenen String angezeigt. Wenn der Rückgabewert `null` ist (`return;`), gibt Dreamweaver an, dass ein Fehler aufgetreten ist, zeigt jedoch keine spezifischen Informationen an.

Hinweis: *Vor den Anführungszeichen (`""`) innerhalb des zurückgegebenen Strings muss ein umgekehrter Schrägstrich (\) eingegeben werden, damit der JavaScript-Interpreter keine Fehler ausgibt.*

Beispiel

Im folgenden Beispiel für die Funktion `applyBehavior()` werden ein Aufruf an die Funktion `MM_openBrWindow()` und benutzerdefinierte Parameter (Höhe und Breite des Fensters, angezeigte Bildlaufleisten, Symbolleisten und andere Funktion sowie die URL, die im Fenster geöffnet werden soll) zurückgegeben.

```
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 diese nicht bereits vorhanden sind – 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 und darf daher nicht von Benutzereingaben abhängen. Die Funktionen werden nur einmal eingefügt, unabhängig davon, wie oft die Aktion auf Elemente im Dokument angewendet wird.

Hinweis: *Vor den Anführungszeichen (`""`) innerhalb des zurückgegebenen Strings muss ein umgekehrter Schrägstrich (\) eingegeben werden, damit der JavaScript-Interpreter keine Fehler ausgibt.*

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 Verhalten von Dreamweaver verwendet wird.

```
function MM_popupMsg(theMsg){ //v1.0
    alert(theMsg);
}

function behaviorFunction(){
    return "MM_popupMsg";
}
```

canAcceptBehavior()

Beschreibung
Die Funktion legt fest, ob die Aktion für das ausgewählte HTML-Element zulässig ist, und gibt das Standardereignis an, das die Aktion auslösen soll. Sie kann auch überprüfen, ob bestimmte Objekte (z. B. SWF-Dateien) im Benutzerdokument vorhanden sind, und die Aktion nicht zulassen, wenn 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 zulässig ist, jedoch keine bevorzugten Ereignisse festgelegt sind.

- Eine Liste der bevorzugten Ereignisse für diese Aktion (in absteigender Reihenfolge). 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 248.
- 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

Im folgenden Beispiel für die Funktion `canAcceptBehavior()` wird eine Liste bevorzugter Ereignisse für das Verhalten für den Fall zurückgegeben, dass das Dokument über benannte Bilder verfügt.

```
function canAcceptBehavior(){
    var theDOM = dreamweaver.getDocumentDOM();
    // Get an array of all images in the document
    var allImages = theDOM.getElementsByTagName('IMG');
    if (allImages.length > 0){
        return "onMouseOver, onClick, onMouseDown";
    }else{
        return false;
    }
}
```

displayHelp()

Beschreibung

Wenn diese Funktion definiert ist, wird unter den Schaltflächen „OK“ und „Abbrechen“ im Dialogfeld „Parameter“ die Schaltfläche „Hilfe“ angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche „Hilfe“ klickt.

Argumente

Keine.

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// the following instance of displayHelp() opens
// in a browser a file that explains how to use
// the extension.
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/superDuperHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

deleteBehavior()

Beschreibung

Diese Funktion macht die durch die Funktion `applyBehavior()` vorgenommenen Änderungen rückgängig.

Hinweis: *Dreamweaver löscht automatisch die mit einem Verhalten verknüpfte Funktionsdeklaration und Ereignisprozedur, wenn der Benutzer das Verhalten im Bedienfeld „Verhalten" löscht. Daher ist es nur nötig, die Funktion* `deleteBehavior()` *zu definieren, wenn die Funktion* `applyBehavior()` *zusätzliche Bearbeitungsvorgänge am Benutzerdokument durchführt (wenn sie z. B. ein Tag einfügt).*

Argumente
applyBehaviorString

Dieses Argument ist der String, der von der Funktion `applyBehavior()` zurückgegeben wird.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

identifyBehaviorArguments()

Beschreibung
Diese Funktion erkennt Argumente aus einem Verhaltensfunktionsaufruf als Navigationshyperlinks, abhängige Dateien, URLs, Verweise im Stil von Netscape Navigator 4.0 oder Objektnamen. Auf diese Weise können URLs in Verhalten aktualisiert werden, wenn der Benutzer das Dokument in einem anderen Verzeichnis speichert, und die verknüpften Dateien können in der Sitemap aufgeführt und beim Hoch- und Herunterladen auf 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. Folgende Argumenttypen sind zulässig:

- 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 einen Objektverweis im Stil von Netscape Navigator 4.0 handelt.
- Der Argumenttyp `IE4.0ref` gibt an, dass es sich bei dem Argument um einen Objektverweis 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 wurde in Dreamweaver 3 hinzugefügt.
- Der Argumenttyp `other` gibt an, dass das Argument keinem der oben aufgeführten Typen angehört.

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 (z. B. „Ebenen ein-/ausblenden“). Bei dieser Variante 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 daher auch 4, 8 oder 12 Argumente haben, jedoch niemals 10 Argumente.

```
function identifyBehaviorArguments(fnCallStr) {
    var listOfArgTypes;
    var itemArray = dreamweaver.getTokens(fnCallStr, '(),');

    // The array of items returned by getTokens() includes the
    // function name, so the number of *arguments* in the array
    // is the length of the array minus one. Divide by 4 to get the
    // number of groups of arguments.
    var numArgGroups = ((itemArray.length - 1)/4);
    // For each group of arguments
    for (i=0; i < numArgGroups; i++){

    // Add a comma and "NS4.0ref,IE4.0ref,other,dep" (because this
    // hypothetical behavior function has a minimum of four
    // arguments the Netscape object reference, the IE object
    // reference, a dependent URL, and perhaps a property value
    // such as "show" or "hide") to the existing list of argument
    // types, or if no list yet exists, add only
    // "NS4.0ref,IE4.0ref,other,dep"
    var listOfArgTypes += ((listOfArgTypes)?",":"") + ¬
    "NS4.0ref,IE4.0ref,other,dep";
    }
}
```

inspectBehavior()

Beschreibung

Diese Funktion überprüft 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, in dieser Funktion weitere Informationen über das Benutzerdokument abzurufen (z. B. 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. Siehe dazu das folgende Beispiel:*

```
function inspectBehavior(enteredStr){
    if(enteredStr){
    //do your work here
    }
}
```

Rückgabewerte

Dreamweaver erwartet keine Rückgabewerte.

Beispiel

Die folgende Instanz der Funktion `inspectBehavior()` aus der Datei „Display Status Message.htm“ trägt im Meldungsfeld des Dialogfelds „Parameter“ die Meldung ein, 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 = ¬
        unescQuotes(msgStr.substring(startStr,endStr));
    }
}
```

Hinweis: *Weitere Informationen zur Funktion `unescQuotes()` 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, wenn das Dialogfeld „Parameter“ größer als 640 x 480 Pixel sein soll.*

Argumente

`platform`

Der Wert des Arguments *`platform`* ist entweder `macintosh` oder `windows`, je nach der verwendeten Plattform.

Rückgabewerte

Dreamweaver erwartet einen String des Typs `widthInPixels,heightInPixels`.

Die zurückgegebenen Abmessungen sind kleiner als die für das gesamte Dialogfeld, da der Bereich für die Schaltflächen „OK“ und „Abbrechen“ nicht einbezogen ist. Wenn im Fenster mit den zurückgegebenen Abmessungen nicht alle Optionen angezeigt werden können, werden Bildlaufleisten eingeblendet.

Beispiel

Im folgenden Beispiel für `windowDimensions()` werden die Abmessungen des Dialogfelds „Parameter“ auf 648 x 520 Pixel gesetzt:

```
function windowDimensions(){
    return "648,520";
}
```

Kapitel 17: Serververhalten

Adobe® Dreamweaver® stellt Benutzern eine Oberfläche zur Verfügung, in der Dokumenten Serververhalten zum Ausführen serverseitiger Aufgaben hinzugefügt werden können. Dazu zählen beispielsweise:

- Filtern von Datensätzen anhand benutzerdefinierter Kriterien
- Navigieren durch Datensätze
- Verknüpfen von Ergebnislisten mit Detailseiten
- Einfügen von Datensätzen in einen Ergebnissatz

Beim Verwenden von Dreamweaver fügen Sie gelegentlich denselben Laufzeitcode wiederholt in die Dokumente ein. Erstellen Sie in solchen Fällen eine Erweiterung, um das Aktualisieren von Dokumenten mit häufig verwendeten Codeblöcken zu automatisieren. Ausführliche Informationen zum Implementieren benutzerdefinierter Serververhalten mit dem Serververhaltengenerator finden Sie unter „Benutzerdefinierte Serververhalten hinzufügen“ im Handbuch *Erste Schritte mit Dreamweaver*. Im Kapitel zu den Serververhalten finden Sie Informationen zum Verwenden der unterstützenden Serververhaltendateien und der Funktionen, die für die Interaktion mit vorhandenen Serververhalten verfügbar sind. Informationen zu den einzelnen Funktionen finden Sie unter „Serververhalten-Funktionen“ und „Extension Data Manager-Funktionen“ im *Dreamweaver API-Referenzhandbuch*. Dreamweaver unterstützt derzeit Serververhaltenserweiterungen, die Laufzeitcode für die folgenden Servermodelle hinzufügen: ASP/JavaScript, ASP/VBScript, ColdFusion und PHP/MySQL.

Begriffserklärungen im Zusammenhang mit Serververhalten

Im Zusammenhang mit Serververhalten werden häufig die folgenden Begriffe verwendet:

Serververhaltenserweiterung Die Serververhaltenserweiterung ist die Schnittstelle zwischen serverseitigem Code und Dreamweaver. Eine Serververhaltenserweiterung besteht aus JavaScript, HTML und Extension Data Markup Language (EDML), d. h. XML-Code, der eigens für Erweiterungsdaten erstellt wurde. Beispieldateien, die nach Servermodell angeordnet sind, finden Sie in Ihrem Installationsverzeichnis im Ordner „Configuration/ServerBehaviors“. Wenn Sie Skriptcode für eine Erweiterung programmieren, weisen 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.

Serververhalteninstanz Wenn Dreamweaver Codeblöcke in das Dokument eines Benutzers einfügt, stellt der eingefügte Code eine Instanz des Serververhaltens dar. Die meisten Serververhalten kann der Benutzer mehrmals anwenden, sodass mehrere Serververhalteninstanzen vorliegen können. Alle Serververhalteninstanzen sind im Bedienfeld „Serververhalten“ der Dreamweaver-Benutzeroberfläche aufgeführt.

Laufzeitcode Laufzeitcode ist die Gruppe der Codeblöcke, die einem Dokument beim Anwenden des Serververhaltens hinzugefügt werden. Diese Codeblöcke enthalten normalerweise auch serverseitigen Code, z. B. ASP-Skriptcode, der in `<% ... %>`-Tags eingeschlossen ist.

Mitglieder Ihre Serververhaltenserweiterung fügt Codeblöcke in das Dokument des Benutzers ein. Ein Codeblock ist ein einzelner, zusammenhängender Block mit Skriptcode. Dies kann z. B. ein serverseitiges Tag, ein HTML-Tag oder ein Attribut sein, mit dem einer Webseite serverseitige Funktionen hinzugefügt werden. Eine EDML-Datei definiert jeden Codeblock als Mitglied. Alle Mitglieder für ein bestimmtes Serververhalten bilden eine Mitgliedergruppe.

Hinweis: *Weitere Informationen über Mitglieder, Mitgliedergruppen und den Aufbau von Dreamweaver EDML-Dateien finden Sie unter „EDML (Extension Data Markup Language)“ auf Seite 263.*

Dreamweaver-Architektur

Wenn Sie mit dem Serververhaltengenerator eine Dreamweaver-spezifische Erweiterung erstellen, legt Dreamweaver mehrere Dateien an (EDML- und HTML-Skriptdateien), die das Einfügen von Serververhaltencode in ein Dreamweaver-Dokument unterstützen. Einige Verhalten verweisen zudem auf JavaScript-Dateien und ermöglichen so weitere Funktionalitäten. Die Architektur vereinfacht Ihre API-Implementierung und behandelt Ihren Laufzeitcode unabhängig von dessen Bereitstellung in Dreamweaver. In diesem Abschnitt finden Sie Informationen zum Bearbeiten dieser Dateien.

Serververhaltenordner und -dateien

Die Benutzeroberfläche für alle Serververhalten befindet sich im Ordner „Configuration/ServerBehaviors/`Servermodellname`“, wobei „Servermodellname“ einer der folgenden Servertypen ist: ASP_Js (JavaScript), ASP_Vbs (VBScript), ColdFusion, PHP_MySQL oder Shared (servermodellübergreifende Implementierungen).

EDML (Extension Data Markup Language)

Bei Verwendung des Serververhaltengenerators erstellt Dreamweaver zwei EDML-Dateien: eine EDML-Gruppendatei und eine EDML-Mitgliederdatei mit den Namen, die Sie im Serververhaltengenerator angegeben haben. Die Gruppendatei enthält die relevanten Mitglieder, d. h. die Codeblöcke, und die Gruppen definieren die Mitglieder, aus denen sich ein Serververhalten zusammensetzt.

Gruppendateien

Gruppendateien enthalten eine Liste der Mitglieder, während Mitgliederdateien alle servermodellspezifischen Codedaten enthalten. Mitgliederdateien können von mehreren Erweiterungen verwendet werden. Daher verweisen mehrere Gruppendateien unter Umständen auf dieselbe Mitgliederdatei.

Das folgende Beispiel bietet eine allgemeine Übersicht über die EDML-Gruppendatei eines Serververhaltens. Eine vollständige Liste der Elemente und Attribute finden Sie unter „EDML-Gruppendatei-Tags“ auf Seite 277.

```
<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="identifier" />
    </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 Mitgliederdatei 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 sein, damit der Autor eines Dreamweaver-Dokuments darauf zugreifen kann. Mehrere Gruppendateien können eine einzelne Mitgliederdatei verwenden.

Die EDML-Datei „moveTo_declareParam.edml“ enthält beispielsweise 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, werden u. a. detaillierte Informationen zur Position, an der der Code eingefügt werden soll, zum Aufbau des Codes und zu den Parametern benötigt, die vom Dreamweaver-Autor bzw. durch die Datenstrukturen zur Laufzeit ersetzt werden. Jede EDML-Mitgliederdatei enthält diese Details für jeden Codeblock. Die Mitgliederdatei beschreibt folgende Daten:

- Der Code und der Einfügeort der eindeutigen Instanz werden durch die Parameter des `insertText`-Tags definiert, z. B.:

  ```
  <insertText location="aboveHTML+80">
  ```

- Im folgenden Beispiel ist angegeben, wie mit dem `searchPatterns`-Tag Instanzen erkannt werden, die sich bereits auf der Seite befinden:

  ```
  <searchPatterns whereToSearch="directive">
      <searchPattern><![CDATA[/var\s*MM_paramName/]]></searchPattern>
  </searchPatterns>
  ```

  Im Block-Tag `searchPatterns` enthält jedes `searchPattern`-Tag ein Suchmuster zum Suchen von Laufzeitcode-Instanzen und zum Extrahieren bestimmter Parameter. Weitere Informationen finden Sie unter „Serververhaltentechniken“ auf Seite 301.

Skriptdatei

Für jedes Serververhalten wird auch eine HTML-Datei mit den Funktionen und Links zu den Skripts erstellt, mit denen die Integration des zugehörigen Codes in die Dreamweaver-Benutzeroberfläche verwaltet wird. Informationen zu den bearbeitbaren Funktionen in dieser Datei finden Sie unter „Implementierungsfunktionen für Serververhalten“ auf Seite 273.

Einfaches Beispiel für ein Serververhalten

In diesem Beispiel werden die Schritte zum Erstellen eines neuen Serververhaltens, die von Dreamweaver generierten Dateien sowie das Bearbeiten dieser Dateien erläutert. Ausführliche Informationen zum Verwenden des Serververhaltengenerators finden Sie unter „Benutzerdefinierte Serververhalten hinzufügen“ im Handbuch *Erste Schritte mit Dreamweaver*. Im Beispiel wird durch einen ASP-Server die Meldung „Hello World“ angezeigt. Dieses Verhalten hat nur ein Mitglied (ein einzelnes ASP-Tag) und es werden keine Änderungen oder sonstigen Bearbeitungen auf der Seite vorgenommen.

Zum Erstellen des Verhaltens erstellen Sie das dynamische Seitendokument, definieren das neue Serververhalten und dann den einzufügenden Code.

Erstellen des dynamischen Seitendokuments

1 Wählen Sie in Dreamweaver die Menüoption „Datei“ > „Neu“ aus.

2 Wählen Sie im Dialogfeld „Neues Dokument“ die Option „Kategorie: Dynamische Seite“ und dann den Seitentyp „ASP JavaScript“ aus.

3 Klicken Sie auf „Erstellen“.

Definieren des neuen Serververhaltens

Hinweis: *Wenn das Bedienfeld „Serververhalten“ nicht geöffnet und sichtbar ist, wählen Sie „Fenster“ > „Serververhalten“ aus.*

1 Klicken Sie im Bedienfeld „Serververhalten“ auf die Schaltfläche mit dem Pluszeichen (+) und wählen Sie die Option „Neues Serververhalten“ aus.

2 Wählen Sie im Dialogfeld „Neues Serververhalten“ die Optionen „Dokumenttyp:ASP JavaScript“ und „Name: Hello World“ aus. (Lassen Sie das Kontrollkästchen „Vorhandenes Serververhalten kopieren“ deaktiviert.)

3 Klicken Sie auf „OK“.

Definieren des einzufügenden Codes

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** (u. U. wird dies von Dreamweaver automatisch vorgenommen).

3 Klicken Sie auf „OK“.

4 Geben Sie im Textfeld „Codeblock“ Folgendes ein: **<% Response.Write("Hello World") %>**.

5 Wählen Sie im Popupmenü „Code einfügen“ die Option „Relativ zur Auswahl“ aus, 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“ aus.

7 Klicken Sie auf „OK“.

Nun wird im Bedienfeld „Serververhalten“ im Menü mit dem Pluszeichen (+) das neue Serververhalten in der Popupliste angezeigt. Zudem enthält das Unterverzeichnis „Configuration/ServerBehaviors/ASP_Js“ im Installationsverzeichnis für Ihre Dreamweaver-Dateien nun die folgenden drei Dateien:

- Gruppendatei: „Hello World.edml“
- Mitgliederdatei: „Hello World_block1.edml“
- Skriptdatei: „Hello World.htm“

Hinweis: *Wenn Sie in einer Umgebung mit mehreren Benutzern arbeiten, sind diese Dateien im Ordner „Anwendungsdaten“ abgelegt.*

Szenarios mit Aufruf der API-Funktionen für Serververhalten

Die API-Funktionen für Serververhalten werden in folgenden Fällen aufgerufen:

- Die Funktion `findServerBehaviors()` wird aufgerufen, wenn ein Dokument geöffnet wird. Sie wird erneut aufgerufen, wenn ein Mitglied bearbeitet wird. Die Funktion 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 mit dem Objekt.
- Wenn sie implementiert ist, ruft Dreamweaver die Funktion `analyzeServerBehavior()` für jede Verhalteninstanz auf, die im Dokument des Benutzers gefunden wird, nachdem alle `findServerBehaviors()`-Funktionen aufgerufen wurden.

  Wenn die Funktion `findServerBehaviors()` ein Verhaltenobjekt erstellt, werden in der Regel die vier Eigenschaften (`incomplete`, `participants`, `selectedNode` und `title`) festgelegt. In manchen Fällen ist es jedoch einfacher, das Festlegen einiger Eigenschaften so lange zu verzögern, bis alle anderen Serververhalten zugehörige Instanzen finden. Das Verhalten „Zum nächsten Datensatz verschieben“ hat beispielsweise 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 zugehörigen Instanzen findet.

  Beim Aufruf der Funktion `analyzeServerBehavior()` des Verhaltens „Zum nächsten Datensatz verschieben“ wird ein Array zurückgegeben, das alle Serververhaltenobjekte im Dokument enthält. Die Funktion kann im Array nach dem zugehörigen Datensatzobjekt suchen.

  Manchmal wird während einer Analyse ein einzelnes Tag im Dokument des Benutzers von mehreren Verhalten als jeweils zugehörige Instanz identifiziert. So kann z. B. die Funktion `findServerBehaviors()` für das Verhalten „Dynamisches Attribut“ eine Instanz dieses Verhaltens erkennen, die im Dokument des Benutzers mit einem `input`-Tag verknüpft ist. Gleichzeitig kann es sein, dass die Funktion `findServerBehaviors()` für das Verhalten „Dynamisches Textfeld“ dasselbe `input`-Tag überprüft und eine Instanz des Verhaltens „Dynamisches Textfeld“ findet. Im Bedienfeld „Serververhalten“ werden als Folge sowohl „Dynamisches Attribut“ als auch „Dynamisches Textfeld“ angezeigt. Um dieses Problem zu beheben, müssen die `analyzeServerBehavior()`-Funktionen alle bis auf eines dieser Serververhalten löschen.

  Hierzu kann durch die Funktion `analyzeServerBehavior()` die Eigenschaft `deleted` jedes Serververhaltens auf `true` gesetzt werden. Wenn die Eigenschaft `deleted` auch dann noch den Wert `true` hat, nachdem der Aufruf der `analyzeServerBehavior()`-Funktionen abgeschlossen ist, wird das Verhalten aus der Liste gelöscht.
- Wenn der Benutzer im Bedienfeld „Serververhalten“ auf die Schaltfläche mit dem Pluszeichen (+) klickt, wird das Popupmenü angezeigt.

  Dreamweaver sucht zunächst in dem Ordner, in dem sich die Serververhalten befinden, nach der Datei „ServerBehaviors.xml“, um den Inhalt des Menüs zu ermitteln. Die Datei „ServerBehaviors.xml“ verweist auf die HTML-Dateien, die im Menü angezeigt werden sollen.

  Wenn die HTML-Datei, auf die verwiesen wird, ein <title>-Tag enthält, wird im Menü der Inhalt des <title>-Tags angezeigt. Wenn die Datei „ServerBehaviors/ASP_Js/GetRecords.htm“ beispielsweise das Tag `<title>Get More Records</title>` enthält, wird im Menü *Get More Records* angezeigt.

  Wenn die Datei kein <title>-Tag enthält, wird im Menü der Dateiname angezeigt. Wenn beispielsweise die Datei „GetRecords.htm“ kein <title>-Tag enthält, wird im Menü „GetRecords“ angezeigt.

Wenn die Datei „ServerBehaviors.xml“ nicht vorhanden ist oder der Ordner eine oder mehrere HTML-Dateien enthält, die nicht in „ServerBehaviors.xml“ aufgeführt sind, sucht Dreamweaver in jeder Datei nach einem <title>-Tag und verwendet dieses oder den Dateinamen im Menü.

Wenn 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 ein Dialogfeld angezeigt. Wenn der Benutzer auf „OK“ klickt, wird die Funktion `applyServerBehavior()` aufgerufen.
- Wenn der Benutzer auf ein vorhandenes Serververhalten doppelklickt, um es zu bearbeiten, zeigt Dreamweaver das Dialogfeld an, führt ggf. die `onLoad`-Prozedur für das `body`-Tag aus und ruft dann die Funktion `inspectServerBehavior()` auf. Die Funktion `inspectServerBehavior()` füllt die Formularelemente mit den aktuellen Argumentwerten. Wenn der Benutzer auf „OK“ klickt, ruft Dreamweaver die Funktion `applyServerBehavior()` erneut auf.
- Wenn der Benutzer auf die Schaltfläche mit dem Minuszeichen (–) klickt, 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 darstellt, an die entsprechende `copyServerBehavior()`-Funktion. Die Funktion `copyServerBehavior()` fügt dem Serververhaltenobjekt alle zusätzlichen Eigenschaften hinzu, die erforderlich sind, um es später einzufügen.

  Nachdem die Funktion `copyServerBehavior()` diesen Vorgang beendet hat, konvertiert Dreamweaver das Serververhaltenobjekt in ein Format, in dem es in die Zwischenablage kopiert werden kann. Dreamweaver löscht während der Konvertierung des Objekts alle Eigenschaften, die auf Objekte verweisen. Alle Eigenschaften des Objekts, die keine Zahl, keinen booleschen Wert und keinen String darstellen, gehen verloren.

  Wenn der Benutzer den Befehl „Einfügen“ auswählt, entpackt Dreamweaver den Inhalt der Zwischenablage und generiert ein neues Serververhaltenobjekt. Das neu erstellte Objekt ist identisch mit dem Original, verfügt jedoch über keine Eigenschaften, die auf Objekte verweisen. Dreamweaver übergibt das neue Serververhaltenobjekt an die Funktion `pasteServerBehavior()`. Die Funktion `pasteServerBehavior()` fügt das Verhalten dem Dokument des Benutzers hinzu. Nachdem die Funktion `pasteServerBehavior()` beendet ist, ruft Dreamweaver die Funktion `findServerBehaviors()` auf, um eine neue Liste aller Serververhalten im Dokument des Benutzers abzurufen.

  Benutzer können Verhalten von einem Dokument in ein anderes kopieren. Die Funktionen `copyServerBehavior()` und `pasteServerBehavior()` dürfen sich beim Austausch von Informationen nur auf die Eigenschaften des Serververhaltenobjekts stützen.

API für Serververhalten

Sie können Serververhalten mit den folgenden API-Funktionen verwalten.

analyzeServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Ermöglicht Serververhalten, die zugehörigen `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“ ruft Dreamweaver beispielsweise die Funktion `analyzeServerBehavior()` in der Datei „DynamicText.htm“ (oder „DynamicText.js“) auf.

Ein Einsatzzweck der Funktion `analyzeServerBehavior()` ist es, das Festlegen aller verfügbaren Eigenschaften (`incomplete`, `participants`, `selectedNode` und `title`) für das Verhaltenobjekt abzuschließen. 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 mehrere Verhalten auf dasselbe Tag im Dokument des Benutzers verweisen. In diesem Fall werden mit der `deleted`-Eigenschaft alle Verhalten bis auf eines aus dem Array gelöscht.

Angenommen, die Serververhalten „Recordset1“, „DynamicText1“ und „DynamicText2“ befinden sich auf einer Seite. Für beide DynamicText-Serververhalten ist es erforderlich, dass „Recordset1“ auf der Seite vorhanden ist. Nachdem die Serververhalten mit der Funktion `findServerBehaviors()` gefunden wurden, ruft Dreamweaver für die drei Serververhalten die Funktion `analyzeServerBehavior()` auf. Wenn die Funktion `analyzeServerBehavior()` für „DynamicText1“ aufgerufen wird, durchsucht die Funktion das Array aller Serververhaltenobjekte auf der Seite nach dem Serververhaltenobjekt, das zu „Recordset1“ gehört. Wenn für „Recordset1“ kein Serververhaltenobjekt gefunden wird, wird die `incomplete`-Eigenschaft auf `true` gesetzt. Durch ein daraufhin im Bedienfeld „Serververhalten“ angezeigtes Ausrufezeichen wird der Benutzer informiert, dass ein Problem vorliegt. Wenn die Funktion `analyzeServerBehavior()` für „DynamicText2“ aufgerufen wird, sucht die Funktion ebenfalls nach dem Objekt für „Recordset1“. Da „Recordset1“ in diesem Beispiel nicht von anderen Serververhalten abhängig ist, muss es die Funktion `analyzeServerBehavior()` nicht definieren.

Argumente
serverBehavior, {*serverBehaviorArray*}

- Das Argument *serverBehavior* ist ein JavaScript-Objekt, das das zu analysierende Verhalten angibt.
- Das Argument {*serverBehaviorArray*} ist ein Array von JavaScript-Objekten, das alle auf der Seite gefundenen Serververhalten darstellt.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

applyServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Liest Werte aus 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. Beim erfolgreichen Abschluss dieser Funktion wird das Dialogfeld „Serververhalten“ geschlossen. Wenn diese Funktion fehlschlägt, wird eine Fehlermeldung angezeigt, ohne dass das Dialogfeld „Serververhalten“ geschlossen wird. Diese Funktion kann das Dokument eines Benutzers bearbeiten.

Weitere Informationen finden Sie unter „dwscripts.applySB()“ auf Seite 273.

Argumente
serverBehavior

Das JavaScript-Objekt *serverBehavior* stellt das Serververhalten dar. Es ist erforderlich, um ein vorhandenes 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
Legt fest, ob ein Verhalten angewendet werden kann. Dreamweaver ruft diese Funktion auf, bevor das Dialogfeld „Serververhalten“ angezeigt wird. Wenn die Funktion den Wert `true` zurückgibt, wird das Dialogfeld „Serververhalten“ angezeigt. Wenn die Funktion den Wert `false` zurückgibt, wird das Dialogfeld „Serververhalten“ nicht angezeigt und das Hinzufügen eines Serververhaltens wird abgebrochen.

Argumente
serverBehavior

Das JavaScript-Objekt *serverBehavior* stellt das Verhalten dar. Es ist erforderlich, um ein vorhandenes 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 angewendet werden kann, andernfalls `false`.

copyServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Das Implementieren der Funktion `copyServerBehavior()` ist optional. Benutzer können Instanzen des angegebenen Serververhaltens kopieren. Im folgenden Beispiel wird diese Funktion für Datensatzgruppen implementiert. Wenn ein Benutzer eine Datensatzgruppe im Bedienfeld „Serververhalten“ oder „Datenbindung“ auswählt, wird das Verhalten mit dem Befehl „Kopieren“ in die Zwischenablage kopiert. Mit dem Befehl „Ausschneiden“ wird das Verhalten ausgeschnitten und in der Zwischenablage abgelegt. Für Serververhalten, die diese Funktion nicht implementieren, sind die Befehle „Kopieren“ und „Ausschneiden“ nicht verfügbar. Weitere Informationen finden Sie unter „Szenarios mit Aufruf der API-Funktionen für Serververhalten“ auf Seite 266.

Die Funktion `copyServerBehavior()` darf zum Austauschen von Informationen mit der Funktion `pasteServerBehavior()` nur Verhaltenobjekteigenschaften 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 müssen.

Hinweis: *Die Funktion „pasteServerBehavior()“ muss ebenfalls implementiert werden, um dem Benutzer die Möglichkeit zu geben, das Verhalten in ein Dreamweaver-Dokument einzufügen.*

Argumente
serverBehavior

- Das JavaScript-Objekt *serverBehavior* stellt das Verhalten dar.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Verhalten erfolgreich in die Zwischenablage kopiert wurde, andernfalls `false`.

deleteServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Entfernt das Verhalten aus dem Dokument des Benutzers. Diese Funktion wird aufgerufen, wenn der 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 274.

Argumente
serverBehavior

- Das JavaScript-Objekt *serverBehavior* stellt das Verhalten dar.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

displayHelp()

Beschreibung
Wenn diese Funktion definiert ist, wird im Dialogfeld unter den Schaltflächen „OK“ und „Abbrechen“ die Schaltfläche „Hilfe“ angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche „Hilfe“ klickt.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// the following instance of displayHelp() opens
// in a browser a file that explains how to use
// the extension.
function displayHelp(){
    var myHelpFile = dw.getConfigurationPath() +
        '/ExtensionsHelp/superDuperHelp.htm';
    dw.browseDocument(myHelpFile);
}
```

findServerBehaviors()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Durchsucht das Dokument des Benutzers nach zugehörigen Instanzen. 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 273 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 Verhalteninstanzen, die auf der Seite gefunden wurden.

inspectServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Bestimmt die Einstellungen des Dialogfelds „Serververhalten“ anhand des angegebenen Verhaltenobjekts. In Dreamweaver wird die Funktion `inspectServerBehavior()` aufgerufen, wenn ein Benutzer das Dialogfeld „Serververhalten“ öffnet. Dreamweaver ruft diese Funktion nur auf, wenn ein Benutzer ein vorhandenes Verhalten bearbeitet.

Argumente
serverBehavior

Das Argument *serverBehavior* ist ein JavaScript-Objekt, das das Verhalten darstellt. Es handelt sich um das gleiche Objekt, das von `findServerBehaviors()` zurückgegeben wird.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

pasteServerBehavior()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Wenn diese Funktion implementiert ist, können Benutzer Instanzen des angegebenen Serververhaltens mit der Funktion `pasteServerBehavior()` einfügen. Wenn der Benutzer das Serververhalten einfügt, strukturiert Dreamweaver die Daten aus der Zwischenablage und generiert ein neues Verhaltenobjekt. Das neue Objekt ist mit dem ursprünglichen Objekt identisch, enthält jedoch keine Verweiseigenschaften mehr. Dreamweaver übergibt das neue Verhaltenobjekt an die Funktion `pasteServerBehavior()`. Die Funktion `pasteServerBehavior()` ermittelt anhand der Eigenschaften des Verhaltenobjekts, welcher Inhalt dem Dokument des Benutzers hinzugefügt werden soll. Die Funktion `pasteServerBehavior()` fügt dann das Verhalten dem Dokument des Benutzers hinzu. Nachdem die Funktion `pasteServerBehavior()` abgeschlossen ist, ruft Dreamweaver die `findServerBehaviors()`-Funktionen auf, um eine neue Liste aller Serververhalten im Dokument des Benutzers abzurufen.

Das Implementieren der Funktion `pasteServerBehavior()` ist optional. Weitere Informationen finden Sie unter „Szenarios mit Aufruf der API-Funktionen für Serververhalten“ auf Seite 266.

Hinweis: *Wenn Sie diese Funktion implementieren, müssen Sie auch die Funktion „copyServerBehavior()“ implementieren.*

Argumente
behavior

Das JavaScript-Objekt *behavior* repräsentiert das Verhalten.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Verhalten erfolgreich aus der Zwischenablage eingefügt wurde, andernfalls `false`.

Implementierungsfunktionen für Serververhalten

Diese Funktionen können innerhalb der HTML-Skriptdateien oder in den darin angegebenen JavaScript-Dateien hinzugefügt oder bearbeitet werden.

dwscripts.findSBs()

Verfügbarkeit
Dreamweaver MX (diese Funktion ersetzt die Funktion `findSBs()` aus früheren Versionen von Dreamweaver).

Beschreibung
Findet alle Instanzen eines Serververhaltens und alle Mitglieder auf der aktuellen Seite. Es werden die Eigenschaften „title“, „type“, die Arrays „participants“, „weights“ und „types“, der `selectedNode`-Wert und das incomplete-Flag festgelegt. Diese Funktion erstellt zudem ein Parameterobjekt, das eine Liste mit benutzerdefinierbaren Eigenschaften wie „recordset“, „name“ und „column name“ enthält. Sie können dieses Array mit der Funktion `findServerBehaviors()` zurückgeben lassen.

Argumente
serverBehaviorTitle

Das Argument *serverBehaviorTitle* ist ein optionaler Titelstring, der verwendet wird, wenn im EDML-Titel kein Titel angegeben ist (hilfreich für die Lokalisierung).

Rückgabewerte
Dreamweaver erwartet ein Array von JavaScript-Objekten, in denen die erforderlichen Eigenschaften definiert wurden. Gibt ein leeres Array zurück, wenn auf der Seite keine Instanzen eines Serververhaltens vorhanden sind.

Beispiel
Mit dem Code im folgenden Beispiel wird nach allen Instanzen eines bestimmten Serververhaltens im aktuellen Benutzerdokument gesucht:

```
function findServerBehaviors() {
    allMySBs = dwscripts.findSBs();
    return allMySBs;
}
```

dwscripts.applySB()

Verfügbarkeit
Dreamweaver MX (diese Funktion ersetzt die Funktion `applySB()` aus früheren Versionen von Dreamweaver).

Beschreibung
Fügt Laufzeitcode für das Serververhalten ein oder aktualisiert ihn. Wenn das *sbObj*-Argument den Wert `null` hat, fügt die Funktion neuen Laufzeitcode ein. Andernfalls wird der vorhandene Laufzeitcode aktualisiert, der durch das Objekt *sbObj* angegeben ist. Benutzereigenschaften müssen als Eigenschaften in einem JavaScript-Objekt festgelegt und als *paramObj* übergeben werden. Diese Einstellungen sollten für alle Argumenten vorliegen, die im EDML-Einfügetext als `@@paramName@@` deklariert sind.

Argumente
paramObj, sbObj

- Das Argument *paramObj* ist das Objekt, das die Benutzereinstellungen enthält.
- Das Argument *sbObj* ist das vorherige Serververhaltenobjekt, wenn Sie ein vorhandenes Serververhalten aktualisieren. Andernfalls hat es den Wert `null`.

Rückgabewerte
Dreamweaver erwartet einen booleschen Wert. `true`, wenn das Serververhalten erfolgreich in das Benutzerdokument eingefügt wurde, andernfalls `false`.

Beispiel
Im folgenden Beispiel füllen Sie das Objekt `paramObj` mit den Benutzereingaben und rufen dann `dwscripts.applySB()` auf, wobei Sie die Eingabe und Ihr Serververhalten (`sbObj`) übergeben.

```
function applyServerBehaviors(sbObj) {
    // get all UI values here...
    paramObj = new Object();
    paramObj.rs= rsName.value;
    paramObj.col = colName.value;
    paramObj.url = urlPath.value;
    paramObj.form__tag = formObj;
     dwscripts.applySB(paramObj, sbObj);
}
```

dwscripts.deleteSB()

Verfügbarkeit
Dreamweaver MX (diese Funktion ersetzt die Funktion `deleteSB()` aus früheren Versionen von Dreamweaver).

Beschreibung
Löscht alle Mitglieder der Serververhalteninstanz `sbObj`. 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 Serververhalteninstanzen angehören („reference count“ > 1).

Argumente
sbObj

- Das Argument *sbObj* 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 `sbObj` gelöscht. Ausgenommen sind Mitglieder, die vom `delete`-Tag der EDML-Datei geschützt werden.

```
function deleteServerBehavior(sbObj) {
    dwscripts.deleteSB(sbObj);
}
```

EDML-Dateien

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 mit Dreamweaver MX eingeführt. Wenn Sie frühere Serververhalten verwenden, finden Sie Informationen in den Handbüchern „Dreamweaver erweitern" der Vorversionen.*

Reguläre Ausdrücke

Sie müssen mit regulären Ausdrücken vertraut sein, die in JavaScript 1.5 implementiert sind. Darüber hinaus müssen Sie wissen, wann sie in EDML-Serververhaltendateien verwendet werden können. Reguläre Ausdrücke können beispielsweise nicht in `quickSearch`-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 (Metazeichen), um Text zu klassifizieren, aufzugliedern und nach vordefinierten Regeln zu verarbeiten. Reguläre Ausdrücke sind ein leistungsstarkes Hilfsmittel zum Durchsuchen und Bearbeiten von Text, da 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 in EDML-Serververhaltendateien von Dreamweaver mithilfe von regulären Ausdrücken Argumente im Laufzeitcode gesucht und deren Werte extrahiert werden. 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 EDML-Serververhaltendateien verwendet werden. Siehe dazu folgende Tabelle:

Regulärer Ausdruck	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, das der Parser das nachfolgende Zeichen nicht als Metazeichen behandelt.
/ ... /i	Bei der Suche nach der Metasequenz die Groß-/Kleinschreibung ignorieren.
(...)	Innerhalb der Metasequenz einen Teilausdruck in Klammern erstellen.
\s*	Nach Leerräumen suchen.

Das EDML-Tag `<searchPatterns whereToSearch="directive">` gibt an, dass Laufzeitcode durchsucht werden muss. Jedes `<searchPattern>...</searchPattern>`-Subtag definiert ein im Laufzeitcode zu suchendes Muster. Im Beispiel „Redirect If Empty" gibt es zwei Muster.

Verfassen Sie im folgenden Beispiel zum Extrahieren von Argumentwerten aus `<% if (@@rs@@.EOF) Response.Redirect("@@new__url@@"); %>` 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 Vorgang wird das Dokument des Benutzers durchsucht und bei Übereinstimmungen werden die jeweiligen Argumentwerte extrahiert. Der erste Teilausdruck in Klammern (`\w+`) extrahiert den Wert für `rs`. Der zweite Teilausdruck (`[^\r\n]*`) extrahiert den Wert für `new_url`.

Hinweis: *Die Zeichenfolge „[^\r\n]*“ steht für alle Zeichen, die kein Zeilenvorschubzeichen sind (auf Macintosh- und Windows-Systemen).*

EDML-Struktur

Sie sollten für Ihre Serververhaltengruppe 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, funktioniert die Gruppendatei `updateRecord.edml` mit der Mitgliederdatei `updateRecord_init.edml`. Wenn eine Mitgliederdatei von mehreren Serververhaltengruppen verwendet wird, müssen Sie eindeutige, beschreibende Namen vergeben.

Hinweis: *Der EDML-Namespace wird unabhängig von der Ordnerstruktur gemeinsam genutzt. Achten Sie darauf, eindeutige Dateinamen zu verwenden. 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 Ihr Laufzeitcode vom `CDATA`-Tag eingeschlossen sein. Das `CDATA`-Tag repräsentiert Zeichendaten, d. h. jeglichen Text, der kein EDML-Markup ist. Wenn Sie das `CDATA`-Tag verwenden, interpretiert der EDML-Parser den enthaltenen Text nicht als Markup, sondern als Klartextblock. Die als `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 folgendes Beispiel):

```
<insertText>Hello, World</insertText>
```

Wenn Sie jedoch Inhalte mit Tags einfügen (z. B. `<img src='foo.gif'>`), kann der EDML-Parser diese nicht immer richtig auswerten. In diesem Fall empfiehlt es sich, die Inhalte in eine `CDATA`-Struktur aufzunehmen:

```
<insertText><![CDATA[<img src='foo.gif'>]]></insertText>
```

Der ASP-Laufzeitcode wird in ein `CDATA`-Tag eingeschlossen, z. B.:

```
<![CDATA[
    <% if (@@rs@@.EOF) Response.Redirect("@@new__url@@"); %>
]]
```

Aufgrund des `CDATA`-Tags werden die ASP-Tags `<%= %>` und die weiteren Inhalte nicht verarbeitet. Stattdessen erhält der Extension Data Manager (EDM) folgenden, nicht interpretierten Text:

```
<% if (Recordset1.EOF) Response.Redirect("http://www.Adobe.com"); %>
```

Bei den folgenden Beispielen für EDML-Definitionen sind die empfohlenen Positionen für `CDATA`-Tags angegeben.

EDML-Gruppendatei-Tags

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, für welche Version der Dreamweaver-Serververhaltenverarbeitung das aktuelle Serververhalten bestimmt ist. Für Dreamweaver CS3 ist dies die Versionsnummer 9. Ist keine Version angegeben, wird von Dreamweaver Version 7 ausgegangen. Bei der vorliegenden Version von Dreamweaver haben alle Gruppen und Mitglieder, die mit dem Serververhaltengenerator erstellt werden, das Versionsattribut 9.0. Die Gruppenversion dieses Attributs hat derzeit keine Auswirkung.

Übergeordnet
`group`

Typ
Attribut.

Erforderlich
Nein.

serverBehavior

Beschreibung
Das Attribut `serverBehavior` gibt an, welches Serververhalten die Gruppe verwenden kann. Wird ein beliebiger `quickSearch`-String der Gruppenmitglieder im Dokument gefunden, bewirkt das vom Attribut `serverBehavior` angegebene 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 Serververhalten-HTML-Datei im Ordner „Configuration/ServerBehaviors“:

```
<group serverBehavior="redirectIfEmpty.htm">
```

dataSource

Beschreibung
Diese erweiterte Funktion unterstützt neue Datenquellen, die zu Dreamweaver hinzugefügt 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 erweitert, um einen neuen Datenquellentyp (z. B. ein COM-Objekt) zu unterstützen, können Sie für eine abweichende Implementierung von „Repeat Region“ die neue Datenquelle in einer weiteren Gruppendatei deklarieren (z. B. 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/DataSources“, z. B.:

```
<group serverBehavior="Repeat Region.htm" dataSource="myCOMdataSource.htm">
```

Diese Gruppe definiert eine neue Implementierung des Serververhaltens „Repeat Region“, die bei Auswahl der Datenquelle COM 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 Gruppendateien ein Serververhalten angewendet, kann die richtige Gruppendatei durch Übergabe eines `subType`-Werts festgelegt werden. Angewendet wird dann die Gruppe mit dem angegebenen Wert `subType`.

Übergeordnet
`group`

Typ
Attribut.

Erforderlich
Nein.

Wert
Der Wert ist ein eindeutiger String, der festlegt, welche Gruppe angewendet wird, z. B.:

```
<group serverBehavior="myServerBehavior.htm" subType="longVersion">
```

Dieses Gruppenattribut definiert die Langform des Subtyps `myServerBehavior`. Sie können auch eine Version mit dem Attribut `subType="shortVersion"` definieren. 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 Serververhaltens 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, z B.:

```
<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 Serververhalteninstanzen werden nach dem ausgewählten Mitglied sortiert. Legen Sie daher das Attribut `selectParticipant` auch dann fest, wenn das Mitglied nicht sichtbar ist.

Übergeordnet
`groupParticipants`

Typ
Attribut.

Erforderlich
Nein.

Wert
Der Wert von *participantName* ist der genaue Name einer Mitgliederdatei (ohne die „.edml“-Erweiterung), die als Gruppenmitglied aufgeführt wird, z. B.: Weitere Informationen finden Sie unter „name“ auf Seite 281.

```
<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 `groupParticipant`-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 Dateinamenerweiterung „.edml“) übereinstimmen.

Übergeordnet
`groupParticipant`

Typ
Attribut.

Erforderlich
Ja.

Wert
Der Wert ist der genaue Name einer Mitgliederdatei (ohne die Dateinamenerweiterung „.edml“), z. B.:

```
<groupParticipant name="redirectIfEmpty_init">
```

Dieses Beispiel bezieht sich auf die Datei „redirectIfEmpty_init.edml“.

partType

Beschreibung
Dieses Attribut gibt den Typ des Mitglieds an.

Übergeordnet
`groupParticipant`

Typ
Attribut.

Erforderlich
Nein.

Werte
identifier, *member*, *option*, *multiple*, *data*

- Der Wert *identifier* kennzeichnet ein Mitglied, das für die gesamte Gruppe steht. Wenn dieses Mitglied im Dokument gefunden wird, gilt die Gruppe als vorhanden. Dies ist der Standardwert, wenn das Attribut `partType` nicht angegeben ist.
- Der Wert *member* kennzeichnet ein normales Mitglied einer Gruppe. Wenn nur dieses Mitglied 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 das Mitglied 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 verwendete Argumente werden beim Gruppieren von Mitgliedern nicht berücksichtigt, da sie möglicherweise abweichende Werte aufweisen.
- Der Wert *data* kennzeichnet ein besonderes Mitglied, das von Programmierern als Speicher für zusätzliche Gruppendaten verwendet wird. Ansonsten wird dieses Mitglied ignoriert.

EDML-Mitgliederdateien

Die folgenden Tags und Attribute sind für EDML-Mitgliederdateien gültig.

<participant>

Beschreibung
Dieses Tag enthält alle Spezifikationen für ein einzelnes Mitglied.

Übergeordnet
Keine.

Typ
Block-Tag.

Erforderlich
Ja.

<participant>-Attribute

Die folgenden Elemente sind gültige Attribute des <participant>-Tags.

version

Beschreibung
Dieses Attribut definiert, für welche Version der Dreamweaver-Serververhaltenverarbeitung das aktuelle Serververhalten bestimmt ist. Für Dreamweaver CS3 ist dies die Versionsnummer 9. Wenn keine Version angegeben ist, wird von Dreamweaver Version 7 ausgegangen. Bei der vorliegenden Version von Dreamweaver haben alle mit dem Serververhaltengenerator erstellten Gruppen und Mitglieder das Versionsattribut 0.0.

Hinweis: *Das Mitglieder-Versionsattribut überschreibt das Gruppen-Versionsattribut, wenn es sich von diesem unterscheidet. Die Mitgliederdatei verwendet jedoch das Gruppen-Versionsattribut, wenn für das Mitglied kein Versionsattribut angegeben ist.*

Bei Mitgliederdateien legt dieses Attribut fest, ob Codeblöcke zusammengeführt werden sollen. Bei Mitgliedern ohne dieses Attribut oder Mitgliedern, bei denen dieses Attribut auf den Wert 4 oder niedriger 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 zur Leistungssteigerung 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 für *searchString* ist ein Stringliteral, das auf der Seite vorhanden ist, wenn das Mitglied existiert. Der String muss möglichst eindeutig sein, um die Leistung zu optimieren, er muss jedoch nicht völlig eindeutig sein. Die Groß-/Kleinschreibung spielt keine Rolle. Achten Sie jedoch auf verzichtbare Leerräume, die vom Benutzer geändert werden können, z. B.:

```
<quickSearch>Response.Redirect</quickSearch>
```

Ist das Tag `quickSearch` leer, gilt dies als Übereinstimmung. Für die genauere Suche werden dann die in den `searchPattern`-Tags definierten regulären Ausdrücke verwendet. Diese Methode ist sinnvoll, wenn zur zuverlässigen 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 sein.

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 beeinflusst, sollte dafür ein `CDATA`-Tag verwendet werden, z. B.:

```
<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 bedingten und wiederholenden 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, an welcher Position 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 287).

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 `HTML`-Tags 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 standardmäßig 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, z. B.:

  ```
  <insert location="aboveHTML+60">
  ```

  Wurde keine Gewichtung angegeben, wird ihr intern der Wert `100` zugewiesen und die Datensatzgruppe unterhalb aller explizit gewichteten Mitglieder eingefügt, z. B.:

  ```
  <insert location="aboveHTML">
  ```

- Der Wert *belowHTML `[+weight]`* entspricht dem Wert `aboveHTML`. Diese Mitglieder werden jedoch unterhalb des schließenden `/HTML`-Tags 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 `body`-Tags eingefügt.
- Der Wert *replaceSelection* ersetzt die aktuelle Auswahl durch den Text. Liegt keine Auswahl vor, wird der Text am Ende des `body`-Tags eingefügt.
- Der Wert *wrapSelection* berücksichtigt die aktuelle Auswahl und fügt vor der Auswahl ein Block-Tag und nach der Auswahl das entsprechende Schluss-Tag ein.
- Der Wert *afterSelection* fügt den Text nach der aktuellen Auswahl oder nach der Einfügemarke ein. Liegt keine Auswahl vor, wird der Text am Ende des `body`-Tags eingefügt.
- Der Wert *beforeNode* fügt den Text vor einem Knoten (einer bestimmten Position im DOM) ein. Wird eine Funktion wie `dwscripts.applySB()` aufgerufen, um den Einfügevorgang durchzuführen, muss der Zeiger auf den Knoten als ein *paramObj*-Parameter übergeben werden. Der benutzerdefinierbare Name dieses Parameters muss über das Attribut `nodeParamName` angegeben werden (siehe „nodeParamName“ auf Seite 286).

  Zusammenfassend gilt: Wenn die Position das Wort `node` enthält, müssen Sie das Tag `nodeParamName` deklarieren.

- Der Wert *replaceNode* ersetzt einen Knoten durch den Text.

- Der Wert *afterNode* fügt den Text nach einem Knoten ein.
- Der Wert *firstChildOfNode* fügt den Text als erstes untergeordnetes Element eines Block-Tags ein. Hiermit können Sie beispielsweise Inhalte am Anfang eines `FORM`-Tags einfügen.
- Der Wert *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 hilfreich beim Hinzufügen von verborgenen Formularfeldern.
- *nodeAttribute[+attribute]* legt ein Attribut eines Tag-Knotens fest. Sofern das Attribut nicht bereits existiert, wird es erstellt.

  Mit `<insert location="nodeAttribute+ACTION" nodeParamName="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 einem Tag ein optionales Attribut hinzufügen. Auf diese Weise können Sie das `INPUT`-Tag eines Benutzers ändern von `<input type="checkbox">` in `<input type="checkbox" <%if(foo)Reponse.Write("CHECKED")%>>`.

  Hinweis: *Beim location="nodeAttribute"-Attributwert 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 Namen des Parameters, 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 vom Benutzer angegebener Name für den mit dem Parameterobjekt an die Funktion `dwscripts.applySB()` übergebenen Knotenparameter. Angenommen, Sie verwenden zum Einfügen von Text in ein Formular einen Parameter mit dem Namen *form_tag*. In Ihrer Serververhaltenfunktion `applyServerBehavior()` können Sie dann mit dem Parameter *form_tag* das zu aktualisierende Formular genau bezeichnen, z. B.:

```
function applyServerBehavior(ssRec) {
    var paramObj = new Object();
    paramObj.rs = getRecordsetName();
    paramObj.form_tag = getFormNode();
    dwscripts.applySB(paramObj, sbObj);
}
```

Sie können den Knotenparameter `form_tag` in Ihrer EDML-Datei festlegen, z. B.:

```
<insertText location="lastChildOfNode" nodeParamName="form_tag">
    <![CDATA[<input type="hidden" name="MY_DATA">]]>
</insertText>
```

Der Text wird als Wert `lastChildOfNode` eingefügt und der ausgewählte Knoten mithilfe der `form_tag`-Eigenschaft des Parameterobjekts übergeben.

<searchPatterns>

Beschreibung
Dieses Tag enthält Informationen zum Suchen von Mitgliedertext im Dokument sowie eine Liste von Suchmustern, die bei der Suche nach einem Mitglied eingesetzt werden. Sind mehrere Suchmuster definiert, müssen diese 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, an welcher Position nach dem Mitgliedertext gesucht werden soll. Dieses Attribut hängt von der Einfügeposition ab. Beide Werte sind daher sorgfältig zu wählen (siehe „location“ auf Seite 285).

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, z. B.:

  ```
  <searchPatterns whereToSearch="tag+FORM">
  ```

  In diesem Beispiel werden nur `form`-Tags durchsucht. Standardmäßig wird der gesamte `outerHTML`-Knoten durchsucht. Geben Sie den Typ von `INPUT`-Tags nach einem Schrägstrich (/) an. Um beispielsweise nach allen Schaltflächen „Senden“ zu suchen, geben Sie folgenden Code ein:

  ```
  <searchPatterns whereToSearch="tag+INPUT/SUBMIT">.
  ```

- Der Wert *tag+** durchsucht den Inhalt eines beliebigen Tags, z. B.:

  ```
  <searchPatterns whereToSearch="tag+*">
  ```

  In diesem Beispiel werden alle Tags durchsucht.

- Der Wert *comment* durchsucht nur die HTML-Kommentare `<! ... >`, z. B.:

  ```
  <searchPatterns whereToSearch="comment">
  ```

  Im folgenden Beispiel werden Tags wie `<!-- my comment here -->` durchsucht.

- Der Wert *text* durchsucht nur reine Textabschnitte, z. B.:

  ```
  <searchPatterns whereToSearch="text">
      <searchPattern>XYZ</searchPattern>
  </searchPatterns>
  ```

  In diesem Beispiel wird ein Textknoten gesucht, der den Text `XYZ` enthält.

<searchPattern>

Beschreibung

Dieses Tag ist ein Muster, mit dem Mitgliedertext identifiziert und daraus Parameterwerte 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 vorkommen.

Weitere Informationen zur Verwendung des `searchPattern`-Tags finden Sie unter „Suchen von Serververhalten“ auf Seite 301.

Übergeordnet

`searchPatterns`

Typ

Block-Tag.

Erforderlich

Ja.

Werte

searchString, */regularExpression/*, <*empty*>

- Der Wert von *searchString* ist ein einfacher Suchstring, bei dem die Groß- und Kleinschreibung unterschieden wird. Er kann nicht zum Extrahieren von Parametern verwendet werden.
- Der Wert von */regularExpression/* ist ein Suchmuster mit regulären Ausdrücken.
- Der Wert <*empty*> wird verwendet, wenn kein Muster angegeben ist. Er 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` zu.

Hinweis: *Es ist sehr wichtig, dass der reguläre Ausdruck mit einem Schrägstrich (/) beginnt und endet. Andernfalls wird der Ausdruck als Suchstringliteral verwendet. Regulären Ausdrücken kann die Modifizieroption `i` nachgestellt werden, um anzugeben, dass die Groß- und Kleinschreibung nicht beachtet werden soll (z. B. `/pattern/i`). VBScript berücksichtigt beispielsweise die Groß- und Kleinschreibung nicht. Es sollte daher `/pattern/i` verwendet werden. JavaScript berücksichtigt die Groß- und 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, z. B.:

```
<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 `searchPattern`-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 Ausdrücke in Klammern verwendet, jedoch keine Parameter angegeben, können in der Liste der Parameternamen zusätzliche Kommas als Platzhalter verwendet werden.

Die Parameternamen müssen 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. Angenommen, Sie verwenden folgendes Suchmuster:

```
<searchPattern paramName="p1,,p2">/(\w+)_(BIG|SMALL)_(\w+)/</searchPattern>
```

Zwei Parameter müssen extrahiert werden, zwischen denen Text steht. Beim Ausgangstext `<%= a_BIG_b %>` entspricht der erste Teilausdruck im Suchmuster `a`. Daraus ergibt sich `p1="a"`. Der zweite Teilausdruck wird ignoriert (beachten Sie die zwei Kommas `,,` 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* (Standardwert) durchsucht das gesamte im Attribut `whereToSearch` angegebene Tag.
- Der Wert *attribute+attribName* durchsucht nur den Wert des angegebenen Attributs, z. B.:

```
<searchPatterns whereToSearch="tag+FORM">
    <searchPattern limitSearch="attribute+ACTION">
        /MY_PATTERN/
    </searchPattern>
</searchPatterns>
```

In diesem Beispiel wird angegeben, dass nur der Wert des `ACTION`-Attributs 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 gültig, 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 gültig, wenn es sich bei `whereToSearch` um ein Tag handelt.

isOptional

Beschreibung
Dieses Attribut gibt an, dass das Suchmuster zum Suchen des Mitglieds nicht erforderlich ist. Es kann bei komplexen Mitgliedern verwendet werden, bei denen ggf. weniger wichtige Parameter zu extrahieren sind. Sie können zunächst Muster erstellen, mit denen ein Mitglied eindeutig identifiziert wird, und anschließend mit optionalen Mustern weniger wichtige 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* (Standardwert), wenn das Tag `searchPattern` erforderlich ist.

Betrachten Sie beispielsweise den 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 mehrere Parameter extrahieren. Auch wenn ein Parameter wie *cursorType* nicht gefunden wird, sollte bei diesem Muster dennoch eine Datensatzgruppe erkannt werden. Der Parameter *cursor* ist optional. In EDML sehen die Suchmuster beispielsweise so aus:

```
<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 sind erforderlich, 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 Mitgliedertext ersetzt wird. Wenn Sie ein `updatePatterns`-Tag angeben, muss es bestimmte Muster enthalten, um im Mitglied die einzelnen Parameter zu suchen und zu ersetzen.

Dieses Tag ist nützlich, wenn ein Benutzer den Mitgliedertext bearbeitet. Die präzisen Aktualisierungen erfolgen nur in den zu ändernden Textabschnitten.

Übergeordnet
`implementation`

Typ
Block-Tag.

Erforderlich
Nein.

<updatePattern>

Beschreibung
Dieses Tag ist ein bestimmter Typ regulärer Ausdrücke und ermöglicht 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 von Parametern, die sich zwischen zwei Teilausdrücken in Klammern befinden, im Format */(pre-pattern)parameter-pattern(post-pattern)/*. Für jeden im Einfügetext definierten eindeutigen `@@paramName@@`-Wert muss mindestens ein Aktualisierungsmuster festgelegt werden. Es folgt 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 gibt zwei Parameter: `rs` und `col`. 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 (wie bei den anderen, für reguläre Ausdrücke verwendeten Sonderzeichen) 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, z. B.:

```
<% ...
    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, z. B.:

```
<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 genau drei Positionen aktualisiert.

<updatePattern>-Attribute

Die folgenden Elemente sind gültige Attribute des `updatePattern`-Tags.

paramName

Beschreibung
Dieses Attribut gibt den Namen des Parameters an, dessen Wert zum Aktualisieren des Mitglieds verwendet wird. Dieser Parameter muss 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@@`, ist ein Parameter mit diesem Namen erforderlich:

```
<updatePattern paramName="rs">pattern</updatePattern>
```

<delete>

Beschreibung
Dieses Tag ist eine optionale erweiterte Funktion, mit der Sie den Löschvorgang eines Mitglieds steuern können. Ohne dieses Tag wird das Mitglied vollständig gelöscht (allerdings nur, wenn kein Serververhalten darauf verweist). Durch Angeben eines `delete`-Tags 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 des auszuführenden Löschvorgangs 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+**

- Der Wert *all* (Standardwert) bewirkt, dass die gesamte Direktive bzw. das gesamte Tag gelöscht wird. Bei Attributen wird die gesamte Definition gelöscht.
- Der Wert *none* bewirkt, dass das Mitglied nie automatisch gelöscht wird.
- Der Wert *tagOnly* bewirkt, dass nur das äußere Tag entfernt, der Inhalt des Tags `innerHTML` jedoch beibehalten wird. Bei Attributen wird auch das äußere Tag entfernt, wenn es sich um ein Block-Tag handelt. Bei Direktiven ohne Bedeutung.
- Der Wert *innerOnly* entfernt bei Anwendung auf Tags nur den Inhalt (das `innerHTML`-Tag). Bei Attributen wird nur der Wert entfernt. Bei Direktiven ohne Bedeutung.
- Der Wert *attribute+attribName* entfernt bei Anwendung auf Tags nur das angegebene Attribut. Bei Direktiven und Attributen ohne Bedeutung.
- Der Wert *attribute+** entfernt alle Attribute für Tags. Bei Direktiven und Attributen ohne Bedeutung.

Wenn das Serververhalten beispielsweise ausgewählten Text in einen Hyperlink konvertiert, können Sie den Hyperlink entfernen, indem Sie nur das äußere Tag entfernen, z. B.:

```
<delete deleteType="tagOnly"/>
```

In diesem Beispiel wird ein Hyperlink-Mitglied von `<A HREF="...">HELLO</A>` in `HELLO` geändert.

<translator>

Beschreibung
Dieses Tag enthält Informationen zum Übersetzen eines Mitglieds, sodass es möglicherweise anders dargestellt wird und ihm ein benutzerdefinierter Eigenschafteninspektor zugeordnet wird.

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 diese 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 mit Übersetzungsanweisungen, von denen jede angibt, an welcher Position nach dem Mitglied gesucht und wie mit dem Mitglied verfahren werden soll.

Übergeordnet
`translator`

Typ
Block-Tag.

Erforderlich
Nein.

<translation>

Beschreibung
Dieses Tag enthält eine einzelne Übersetzungsanweisung, die die Position für das Mitglied, die Art der auszuführenden 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, an welcher Position (relativ zur Einfügeposition) nach dem Text gesucht werden soll. Beide Werte sind daher sorgfältig zu wählen (siehe „location“ auf Seite 285).

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. Die Typen sind voreingestellt und stellen bestimmte Funktionen für die Übersetzung bereit. Wenn Sie beispielsweise `dynamic data` angeben, müssen zu übersetzende Daten sich verhalten wie dynamische Daten von Dreamweaver, d. h., sie müssen 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*

- Der Wert *dynamic data* gibt an, dass die übersetzten Direktiven im Aussehen und Verhalten den dynamischen Daten von Dreamweaver entsprechen, z. B.:

```
<translation whereToSearch="tag+IMAGE"
    limitSearch="attribute+SRC"
    translationType="dynamic data">
```

- Der Wert *dynamic image* gibt an, dass die übersetzten Attribute in Aussehen und Verhalten den dynamischen Bildern von Dreamweaver entsprechen, z. B.:

```
<translation whereToSearch="IMAGE+SRC"
    translationType="dynamic image">
```

- Der Wert *dynamic source* gibt an, dass die übersetzten Direktiven im Verhalten den dynamischen Quellen von Dreamweaver entsprechen, z. B.:

```
<translation whereToSearch="directive"
    translationType="dynamic source">
```

- Der Wert *`tabbed region start`* gibt an, dass die übersetzten `<CFLOOP>`-Tags den Anfang einer Registerkartenkontur definieren, z. B.:

```
<translation whereToSearch="CFLOOP"
    translationType="tabbed region start">
```

- Der Wert *`tabbed region end`* gibt an, dass die übersetzten `</CFLOOP>`-Tags das Ende einer Registerkartenkontur definieren, z. B.:

```
<translation whereToSearch="CFLOOP"
    translationType="tabbed region end">
```

- Der Wert *custom* ist der Standardfall, bei dem der Übersetzung keine interne Dreamweaver-Funktionalität hinzugefügt wird. Dieser Wert wird häufig beim Angeben eines Tags zum Einfügen eines benutzerdefinierten Eigenschafteninspektors verwendet, z. B.:

```
<translation whereToSearch="directive"
    translationType="custom">
```

<openTag>

Beschreibung
Ein optionales Tag, das am Anfang des Übersetzungsabschnitts eingefügt werden kann. Hierdurch können bestimmte andere Erweiterungen (z. B. benutzerdefinierte Eigenschafteninspektoren) die Übersetzung finden.

Übergeordnet
`translation`

Typ
Block-Tag.

Erforderlich
Nein.

Werte
Der Wert *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>` angeben, werden die dynamischen Daten in das Tag `MM_DYNAMIC_CONTENT` übersetzt.

<attributes>

Beschreibung
Dieses Tag enthält eine Liste der Attribute, die dem mit `openTag` angegebenen übersetzten Tag hinzuzufügen sind. Wenn das `openTag`-Tag nicht definiert ist und mit dem `searchPattern`-Tag `tag` angegeben wird, enthält dieses Tag eine Liste der übersetzten Attribute, die dem gefundenen Tag hinzuzufügen sind.

Übergeordnet
`translation`

Typ
Block-Tag.

Erforderlich
Nein.

<attribute>

Beschreibung
Dieses Tag gibt ein einzelnes Attribut (oder übersetztes Attribut) an, das dem übersetzten Tag hinzuzufügen ist.

Übergeordnet
`attributes`

Typ
Block-Tag.

Erforderlich
Ja (mindestens eines).

Werte
Die Angabe *attributeName*="*attributeValue*" legt einen Wert für ein Attribut fest. In der Regel ist der Attributname vorgegeben und der Wert enthält einige Parameterverweise, die durch die Parametermuster extrahiert werden, z. B.:

```
<attribute>SOURCE="@@rs@@"</attribute>
<attribute>BINDING="@@col@@"</attribute>
```

oder

```
<attribute>
    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
Der Wert von *displayString* ist ein beliebiger String, der Text und HTML enthalten kann. Er kann Parameterverweise enthalten, die durch Parametermuster extrahiert werden. Zum Beispiel bewirkt `<display>{@@rs@@.@@col@@}</display>` die Übersetzung als `{myRecordset.myCol}`.

<closeTag>

Beschreibung
Ein optionales Tag, das am Ende des übersetzten Abschnitts eingefügt werden soll. Hierdurch können bestimmte andere Erweiterungen (z. B. benutzerdefinierte Eigenschafteninspektoren) die Übersetzung finden.

Übergeordnet
`translation`

Typ
Block-Tag.

Erforderlich
Nein.

Werte
Der Wert von *tagName* ist ein gültiger Tag-Name, der mit einem *openTag*-Tag der Übersetzung übereinstimmen muss.

Beispiel
Wenn Sie den Wert `<closeTag>MM_DYNAMIC_CONTENT</closeTag>` angeben, endet die Übersetzung der dynamischen Daten mit dem Tag `</MM_DYNAMIC_CONTENT>`.

Serververhaltentechniken

Dieser Abschnitt behandelt die allgemeinen und erweiterten Techniken zum Erstellen und Bearbeiten von Serververhalten. Die meisten vorgeschlagenen Verfahren setzen bestimmte Einstellungen in den EDML-Dateien voraus.

Suchen von Serververhalten

Sie können Serververhalten wie folgt suchen:

- Verfassen von Suchmustern
- Verwenden optionaler Suchmuster

Verfassen von Suchmustern

Um Serververhalten aktualisieren oder löschen zu können, muss Dreamweaver jede Instanz in einem Dokument finden können. Hierzu sind ein `quickSearch`-Tag und mindestens ein `searchPattern`-Tag (innerhalb des `searchPatterns`-Tags) erforderlich.

Das `quickSearch`-Tag muss ein String (kein regulärer Ausdruck) sein, der angibt, dass das Serververhalten auf der Seite vorhanden ist. Die Groß- und Kleinschreibung spielt hierfür keine Rolle. Der String sollte kurz und eindeutig sein und keine Leerräume oder sonstigen Abschnitte enthalten, die der Benutzer ändern kann. Es folgt ein Beispiel für ein Mitglied, das aus einem einfachen ASP JavaScript-Tag besteht:

```
<% if (Recordset1.EOF) Response.Redirect("some_url_here") %>
```

Im folgenden Beispiel wird mit dem `quickSearch`-String nach diesem Tag gesucht:

```
<quickSearch>Response.Redirect</quickSearch>
```

Zur Leistungssteigerung wird das `quickSearch`-Muster zu Beginn des Prozesses zum Suchen von Serververhalteninstanzen eingesetzt. Wenn dieser String im Dokument gefunden wird und das Mitglied ein Serververhalten identifiziert (in der Gruppendatei, `partType="identifier"` für dieses Mitglied), werden die entsprechenden Serververhaltendateien geladen und die Funktion `findServerBehaviors()` wird aufgerufen. Wenn das Mitglied keine eindeutigen Strings enthält, nach denen gesucht werden kann (oder zur Fehlersuche), können Sie den `quickSearch`-String leer lassen, z. B.:

```
<quickSearch></quickSearch>
```

In diesem Beispiel wird das Serververhalten immer geladen und kann das Dokument durchsuchen.

Anschließend wird mit dem `searchPattern`-Tag das Dokument genauer als mit dem Tag `quickSearch` durchsucht und Parameterwerte werden aus dem Mitgliedercode extrahiert. In den Suchmustern (Attribut `whereToSearch`) ist angegeben, an welcher Position mit einer Reihe von `searchPattern`-Tags, die spezifische Muster enthalten, gesucht werden soll. In diesen Mustern können einfache Strings oder reguläre Ausdrücke verwendet werden. Der vorangegangene Beispielcode ist eine ASP-Direktive. Somit dienen die Spezifikation `whereToSearch="directive"` und ein regulärer Ausdruck zum Identifizieren der Direktive und zum Extrahieren der Parameter, z. B.:

```
<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. Dem umgekehrten Schrägstrich folgt ein `i`. Das bedeutet, es wird nicht zwischen Groß- und Kleinschreibung unterschieden. Innerhalb des regulären Ausdrucks wird die besondere Funktion von Sonderzeichen wie Klammern () und Punkten (.) durch einen vorangestellten umgekehrten Schrägstrich (\) aufgehoben. Die zwei Parameter `rs` und `new_url` werden durch Teilausdrücke in Klammern aus dem String extrahiert (die Parameter müssen in Klammern eingeschlossen sein). In diesem Beispiel werden die Parameter durch `(\w+)` und `([^\r\n]*)` angegeben. Diese Werte entsprechen den Werten regulärer Ausdrücke, die normalerweise von `$1` und `$2` zurückgegeben werden.

Optionale Suchmuster

In manchen Fällen möchten Sie möglicherweise ein Mitglied identifizieren, selbst wenn einige Parameter nicht gefunden werden. In dem Mitglied sind optionale Informationen gespeichert, z. B. eine Telefonnummer. Sie können dann z. B. den folgenden ASP-Code verwenden:

```
<% //address block
    LNAME = "joe";
    FNAME = "smith";
    PHONE = "123-4567";
%>
```

Sie können z. B. 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 muss die Telefonnummer angegeben sein. Durch Hinzufügen des Attributs `isOptional` können Sie die Telefonnummer als optional deklarieren, z. B.:

```
<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 erkannt, wenn die Telefonnummer nicht gefunden wird.

Zuordnen von Mitgliedern

Wenn ein Serververhalten mehrere Mitglieder hat, müssen diese Mitglieder im Dokument des Benutzers identifiziert und zugeordnet werden. Wenn der Benutzer mehrere Instanzen des Serververhaltens auf ein Dokument anwendet, muss somit jede Mitgliedergruppe entsprechend zugeordnet werden. Damit Mitglieder richtig zugeordnet werden, müssen Sie Parameter ändern oder hinzufügen und Mitglieder so konstruieren, dass sie eindeutig identifiziert werden können.

Beim Zuordnen müssen einige Regeln befolgt werden. Mitglieder werden dann zugeordnet, wenn alle gleichnamigen Parameter denselben Wert haben. Oberhalb 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 anhand ihrer relativen Position in Bezug auf die Auswahl oder auf gemeinsame, für Einfügungen verwendete Knoten zugeordnet.

Mitglieder ohne Parameter werden automatisch zugeordnet, wie im folgenden Beispiel eines Serververhaltens mit Gruppendatei dargestellt:

```
<group serverBehavior="test.htm">
    <title>Test</title>
    <groupParticipants>
        <groupParticipant name="test_p1" partType="identifier" />
        <groupParticipant name="test_p2" partType="identifier" />
    </groupParticipants>
</group>
```

Im folgenden Beispiel werden zwei einfache Mitglieder oberhalb des `html`-Tags eingefügt:

```
<% //test_p1 %>
<% //test_p2 %>
<html>
```

Diese Mitglieder werden gesucht und zugeordnet und im Bedienfeld „Serververhalten“ wird einmal *Test* angezeigt. Wenn Sie das Serververhalten erneut hinzufügen, werden keine Mitglieder hinzugefügt, da sie bereits vorhanden sind.

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 im Dialogfeld „Serververhalten - Test“ einen eindeutigen Namen angeben. Wenn der Benutzer den Namen **aaa** eingibt, werden folgende Mitglieder hinzugefügt:

Wenn Sie das Serververhalten erneut mit einem anderen Namen (z. B. **bbb**) hinzufügen, sieht das Dokument folgendermaßen aus:

```
<% //test_p1 name="aaa" %>
<% //test_p2 name="aaa" %>
<html>
```

Im Bedienfeld „Serververhalten“ sind zwei Instanzen von „Test“ aufgeführt. Wenn der Benutzer versucht, der Seite eine dritte Instanz mit dem Namen **aaa** hinzuzufügen, wird kein Inhalt hinzugefügt, da diese Instanz bereits vorhanden ist.

Innerhalb des `html`-Tags können zum Zuordnen gegebenenfalls auch Positionsinformationen verwendet werden. Im folgenden Beispiel wird je ein Mitglied vor und nach der Auswahl hinzugefügt:

```
<% if (expression) { //mySBName %>
```

Zufällige HTML-Auswahl:

```
<% } //end 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 weitere Instanz dieses Serververhaltens hinzufügen, z. B.:

```
<% if (expression) { //mySBName %>
```

Zufällige HTML-Auswahl:

```
<% } //end mySBName %>
```

Weiterer HTML-Code:

```
<% if (expression) { //mySBName %>
```

Weitere HTML-Auswahl:

```
<% } //end mySBName %>
```

Nun sind zwei identische Instanzen der jeweiligen Mitglieder vorhanden (dies ist innerhalb des HTML-Bereichs zulässig). Die Zuordnung der Instanzen erfolgt in Dreamweaver in der Reihenfolge, in der sie im Dokument vorkommen.

Im folgenden Beispiel ist ein Zuordnungsproblem und die entsprechende Behebung des Problems dargestellt. Sie können ein Mitglied erstellen, das Steuern anhand einiger dynamischer Daten berechnet und das Ergebnis in der aktuellen Auswahl anzeigt.

```
<% total = Recordset1.Fields.Item("itemPrice").Value * 1.0825 %>
<html>
<body>
    The total (with taxes) is $<%=total%>
</body>
</html>
```

Die beiden Mitglieder werden zugeordnet, weil sie keine gemeinsamen Parameter haben. Wenn Sie jedoch eine zweite Instanz dieses Serververhaltens hinzufügen, ergibt sich folgender Code:

```
<% total = Recordset1.Fields.Item("itemPrice").Value * 1.0825 %>
<% total = Recordset1.Fields.Item("salePrice").Value * 1.0825 %>
<html>
<body>
    The total0(with taxes) is $<%=total%>
    Sale price (with taxes) is $<%=total%>
</body>
</html>
```

Dieses Serververhalten funktioniert nicht mehr korrekt, da nur ein Parameter den Namen `total` hat. Sie können dieses Problem beheben, indem Sie einen Parameter mit einem eindeutigen Wert angeben, mit dem Mitglieder zugeordnet werden können. Im folgenden 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>
    The total0(with taxes) is $<%=itemPrice_total%>
    Sale price (with taxes) is $<%=salePrice_total%>
</body>
</html>
```

Die Suchmuster können die Mitglieder nun eindeutig erkennen und zuordnen.

Suchmuster-Auflösung

Dreamweaver unterstützt folgende Aktionen durch Verwenden der Mitgliederfunktionalität `searchPatterns`:

- Abhängigkeiten bei der Dateiübertragung
- Aktualisieren der Dateipfade für alle Dateiverweise (z. B. für Include-Dateien)

Wenn Dreamweaver Servermodelle anlegt, werden Listen mit Mustern erstellt, indem alle Mitglieder nach speziellen `paramNames`-Attributen durchsucht werden. Dreamweaver verwendet alle `searchPattern`-Tags, in denen eines der `paramNames`-Attribute auf `_url` endet, zum Suchen nach URLs für das Überprüfen auf Dateiabhängigkeiten und zum Korrigieren des Pfadnamens. In einem einzelnen `searchPattern`-Tag können mehrere URLs angegeben werden.

Für jedes `searchPattern`-Übersetzer-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-Verweise übersetzt werden. Zudem kann nur eine einzelne URL als Include-Datei übersetzt werden kann.

Beim Auflösen eines `searchPatterns`-Tags verwendet Dreamweaver folgenden Algorithmus:

1. Suche 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 ausgeführt werden, wird von einer Tag-Attribut-Kombination ausgegangen. Andernfalls sucht Dreamweaver nach `searchPattern`-Tags mit einem `paramName`-Attribut, das über das Suffix `_url` verfügt, sowie nach einem definierten regulären Ausdruck. (Weitere Informationen zu regulären Ausdrücken finden Sie unter „Reguläre Ausdrücke“ auf Seite 275.)

Das `searchPatterns`-Tag im folgenden Beispiel enthält kein Suchmuster, da es ein Tag (`cfinclude`) mit einem Attribut (`template`) kombiniert, um die URL zum Überprüfen der Dateiabhängigkeiten, zum Korrigieren des Pfads usw. zu extrahieren:

```
<searchPatterns whereToSearch="tag+cfinclude">
    <searchPattern paramNames="include_url" limitSearch="attribute+template" />
</searchPatterns>
```

Die im vorangegangenen Beispiel verwendete Tag-Attribut-Kombination kann nicht auf Übersetzungen angewendet werden, da Dreamweaver auf der JavaScript-Ebene stets zu Klartext übersetzt. Das Überprüfen der Dateiabhängigkeiten, das Korrigieren des Pfads usw. hingegen erfolgt immer auf der C-Ebene. Auf der C-Ebene teilt Dreamweaver das Dokument intern in Direktiven (reinen Text) und Tags (in einer effizienten Strukturansicht organisiert) auf.

Aktualisieren von Serververhalten

Sie können Serververhalten wie folgt aktualisieren:

- Ersetzungsaktualisierung
- Präzise Aktualisierung

Ersetzungsaktualisierung

Standardmäßig haben EDML-Mitgliederdateien kein `<updatePatterns>`-Tag und Instanzen der Mitglieder werden im Dokument beim Aktualisieren vollständig ersetzt. Wenn ein Benutzer ein vorhandenes Serververhalten bearbeitet und auf „OK“ klickt, werden alle Mitglieder gelöscht, deren Parameterwerte sich geändert haben. Die entsprechenden Mitglieder werden dann mit dem neuen Wert an derselben Position wieder eingefügt.

Wenn der Benutzer den Mitgliedercode im Dokument anpasst, wird das Mitglied möglicherweise nicht mehr erkannt, wenn mit den Suchmustern nach dem alten Code gesucht wird. Mit kürzeren Suchmustern kann der Benutzer den Mitgliedercode im Dokument anpassen. Durch Aktualisieren der Serververhalteninstanz wird das Mitglied jedoch möglicherweise ersetzt, wodurch die Anpassungen dann verloren gehen.

Präzise Aktualisierung

In einigen Fällen ist es erwünscht, dass Benutzer den Mitgliedercode anpassen, nachdem dieser in das Dokument eingefügt wurde. Zu diesem Zweck können in der EDML-Datei die Suchparameter eingeschränkt und Aktualisierungsmuster bereitgestellt werden. Nachdem Sie das Mitglied zur Seite hinzugefügt haben, aktualisiert das Serververhalten nur bestimmte Teile des Mitglieds. Es folgt ein Beispiel für ein einfaches Mitglied mit zwei Parametern:

```
<% if (Recordset1.EOF) Response.Redirect("some_url_here") %>
```

Dabei können folgende Suchmuster verwendet 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 kann dann einer bestimmten Instanz dieses Codes einen anderen Test hinzufügen, z. B.:

```
<% if (Recordset1.EOF || x > 2) Response.Redirect("some_url_here") %>
```

Bei den Suchmustern treten Fehler auf, denn sie suchen nach einer Klammer nach dem `EOF`-Parameter. Um die Toleranz der Suchmuster zu erhöhen, können Sie sie in mehrere kürzere Komponenten aufteilen, z. B.:

```
<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 das Serververhalten beim Klicken auf „OK“ jedoch die URL ändert, wird das Mitglied ersetzt und alle 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 angeordnet. Verwenden Sie für Suchmuster den Parameter `textBeforeParam(param)textAfterParam`. Verwenden Sie für Aktualisierungsmuster den Parameter `(textBeforeParam)param(textAfterParam)`. Der gesamte Text zwischen den beiden Teilausdrücken in Klammern wird durch den neuen Parameterwert ersetzt.

Löschen von Serververhalten

Sie können Serververhalten wie folgt löschen:

- Standardlöschung und Abhängigkeitszähler
- Einschränken der Mitgliederlöschung durch delete-Tags

Standardlöschung und Abhängigkeitszähler

Der Benutzer kann eine im Bedienfeld „Serververhalten“ ausgewählte Instanz löschen, indem er auf die Schaltfläche mit dem Minuszeichen (–) klickt oder die Entf-Taste drückt. Alle Mitglieder werden gelöscht. Ausgenommen sind Mitglieder, auf die auch andere Serververhalten zugreifen. Wenn mehrere Serververhalten einen Mitgliederverweis auf denselben Knoten enthalten, wird der Knoten nicht gelöscht.

Zum Löschen von Mitgliedern wird standardmäßig das gesamte Tag entfernt. Wenn es sich bei der Einfügeposition um `wrapSelection` handelt, wird nur das äußere Tag entfernt. Bei Attributen wird die gesamte Attributdeklaration entfernt. Im folgenden Beispiel ist ein Attributsmitglied des `ACTION`-Attributs eines `form`-Tags dargestellt:

```
<form action="<% my_participant %>">
```

Nach dem Löschen des Attributs verbleibt nur `form`.

Einschränken der Mitgliederlöschung durch delete-Tags

Um das Löschen von Mitgliedern einzuschränken, fügen Sie der EDML-Datei ein delete-Tag hinzu. Es folgt ein Beispiel für ein Mitglied, das ein `href`-Attribut eines Hyperlinks ist:

```
<a href="<%=MY_URL%>">Link Text</a>
```

Wenn dieses Attributmitglied gelöscht wird, lautet das Tag `<a>Link Text</a>` und wird in Dreamweaver nicht mehr als Hyperlink angezeigt. Es empfiehlt sich, nur den Attributwert zu löschen. Fügen Sie dazu der EDML-Mitgliederdatei das folgende Tag hinzu:

```
<delete deleteType="innerOnly"/>
```

Eine andere Möglichkeit besteht darin, das gesamte Tag zu entfernen. Durch Eingeben von `<delete deleteType="tagOnly"/>` wird das Attribut gelöscht. Der Ergebnistext lautet *Link Text*.

Gemeinsame Verwendung von im Speicher abgelegten JavaScript-Dateien

Wenn mehrere HTML-Dateien Verweise auf eine bestimmte JavaScript-Datei enthalten, lädt Dreamweaver die JavaScript-Datei an einen zentralen Speicherort, sodass die HTML-Dateien gemeinsam auf dieselbe JavaScript-Quelle zugreifen können. Diese Dateien enthalten folgende Zeile:

```
//SHARE-IN-MEMORY=true
```

Wenn eine JavaScript-Datei die Direktive `SHARE-IN-MEMORY` enthält und eine HTML-Datei auf die Direktive verweist (durch das `SCRIPT`-Tag mit dem Attribut `SRC`), lädt Dreamweaver die Datei an einen zentralen Ort im Speicher, von 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 keine 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 Programmieren neuer JavaScript-Dateien diese Dateien und ihre Benennungskonventionen.*

Kapitel 18: Datenquellen

Datenquellendateien werden im Ordner „Configuration/DataSources/*Servermodellname*“ gespeichert. Dreamweaver unterstützt derzeit die folgenden Servermodelle: ASP.NET/C#, ASP.NET/VisualBasic, ASP/JavaScript, ASP/VBScript, Adobe ColdFusion, JSP und PHP/MySQL. In jedem Unterordner für die Servermodelle befinden sich HTML- und EDML-Dateien, die den Datenquellen des jeweiligen Servermodells zugeordnet sind.

In der folgenden Tabelle sind die Dateien zum Erstellen von Datenquellen aufgeführt:

Pfad	**Datei**	**Beschreibung**
Configuration/DataSources/*Servermodellname*	*Datenquellenname*.htm	Gibt den Namen der Datenquelle und den Speicherort der unterstützenden JavaScript-Dateien an.
Configuration/DataSources/*Servermodellname*	*Datenquellenname*.edml	Definiert den Code, den Dreamweaver zum Darstellen der Datenquellenwerte in das Dokument einfügt.
Configuration/DataSources/*Servermodellname*	*Datenquellenname*.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 im Menü mit dem Pluszeichen (+) angezeigt werden, basieren auf dem für die jeweilige Seite angegebenen Servermodell. So können Benutzer beispielsweise Datensatzgruppen, Befehle, Anforderungsvariablen, Sitzungsvariablen oder Anwendungsvariablen für ASP-Anwendungen einfügen. Weitere Informationen finden Sie in den Erklärungen zur Funktion `dreamweaver.dbi.getDataSources()` im *Dreamweaver API-Referenzhandbuch*.

Im Folgenden werden die einzelnen Schritte beim Hinzufügen dynamischer Daten beschrieben:

1 Wenn der Benutzer im Bedienfeld „Bindungen“ auf das Menü mit dem Pluszeichen (+) klickt, wird ein Popupmenü angezeigt.

Um den Inhalt des Menüs zu ermitteln, sucht Dreamweaver zunächst nach der Datei „DataSources.xml“ in dem Ordner, in dem sich die Datenquellen befinden (z. B. „Configuration/DataSources/ASP_Js/DataSources.xml“). Die Datei „DataSources.xml“ beschreibt den Inhalt des Popupmenüs. Sie enthält Verweise auf die HTML-Dateien, die im Popupmenü angezeigt werden.

Dreamweaver prüft jede aufgeführte HTML-Datei auf title-Tags. Wenn die Datei ein title-Tag enthält, wird im Menü der Inhalt des title-Tags angezeigt. Wenn die Datei kein title-Tag enthält, wird im Menü der Dateiname angezeigt.

Nachdem Dreamweaver die Datei „DataSources.xml“ ausgewertet hat, wird der Rest des Ordners nach anderen Elementen durchsucht, die im Menü angezeigt werden sollen. Dieser Vorgang wird auch durchgeführt, wenn die Datei „DataSources.xml“ nicht vorhanden ist. Wenn Dreamweaver Dateien im Hauptordner findet, die sich nicht im Menü befinden, werden sie dem Menü hinzugefügt. Wenn Unterordner Dateien enthalten, die sich nicht im Menü befinden, erstellt Dreamweaver ein Untermenü und fügt dem Untermenü diese Dateien hinzu.

2 Wenn der Benutzer im Menü mit dem Pluszeichen (+) ein Menüelement auswählt, ruft Dreamweaver die Funktion `addDynamicSource()` auf. Auf diese Weise wird sichergestellt, dass dem Dokument des Benutzers Code für die Datenquelle hinzugefügt wird.

3 Dreamweaver prüft sämtliche Dateien im Ordner des entsprechenden Servermodells und ruft für jede Datei die Funktion `findDynamicSources()` auf. Für jeden Wert im zurückgegebenen Array ruft Dreamweaver für die entsprechende Datei die Funktion `generateDynamicSourceBindings()` auf. Dadurch wird eine aktuelle Liste sämtlicher Felder in allen Datenquellen für das Dokument des Benutzers abgerufen. Diese Felder werden dem Benutzer in Form von Struktursteuerelementen im Dialogfeld „Dynamische Daten" bzw. „Dynamischer Text" oder im Bedienfeld „Bindungen" angezeigt. Die Datenquellenstruktur für ein ASP-Dokument wird beispielsweise folgendermaßen dargestellt:

```
Recordset (Recordset1)
    ColumnOneInRecordset
    ColumnTwoInRecordset
Recordset (Recordset2)
    ColumnOfRecordset
Request
    NameOfRequestVariable
    NameOfAnotherRequestVariable
Session
    NameOfSessionVariable
```

4 Wenn der Benutzer im Bedienfeld „Bindungen" auf den Namen einer Datenquelle doppelklickt, ruft Dreamweaver die Funktion `editDynamicSource()` auf, um Änderungen innerhalb der Struktur zu verarbeiten.

5 Wenn der Benutzer auf die Schaltfläche mit dem Minuszeichen (–) klickt, ruft Dreamweaver die aktuelle Knotenauswahl aus der Datenquellenstruktur ab und übergibt sie an die Funktion `deleteDynamicSource()`. Die Funktion löscht den Code, der zuvor mit der Funktion `addDynamicSource()` hinzugefügt wurde. Wenn die aktuelle Auswahl nicht gelöscht werden kann, gibt die Funktion eine Fehlermeldung zurück. Nach Abschluss der Funktion `deleteDynamicSource()` aktualisiert Dreamweaver die Datenquellenstruktur, indem die Funktionen `findDynamicSources()` und `generateDynamicSourceBindings()` aufgerufen werden.

6 Wenn der Benutzer eine Datenquelle auswählt und im Dialogfeld „Dynamische Daten" oder „Dynamischer Text" auf „OK" klickt, ruft Dreamweaver die Funktion `generateDynamicDataRef()` auf. Dasselbe geschieht, wenn der Benutzer im Bedienfeld „Bindungen" auf „Einfügen" oder „Verbinden" klickt. Der zurückgegebene Wert wird im Dokument an der Stelle eingefügt, an der sich die Einfügemarke befindet.

7 Wenn der Benutzer im Dialogfeld „Dynamische Daten" ein dynamisches Datenobjekt bearbeitet, muss die Auswahl in der Datenquellenstruktur mit dem Objekt initialisiert werden. Dasselbe gilt, wenn der Benutzer ein dynamisches Datenobjekt im Dialogfeld „Dynamischer Text" bearbeitet. Zum Initialisieren des Struktursteuerelements durchläuft Dreamweaver sämtliche Dateien im Ordner des entsprechenden Servermodells (z. B. im Ordner „Configuration/DataSources/ASP_Js"). Dreamweaver durchläuft alle Dateien und ruft die Implementierung der Funktion `inspectDynamicDataRef()` auf.

   Dreamweaver ruft die Funktion `inspectDynamicDataRef()` auf, um das dynamische Datenobjekt aus dem Code im Dokument des Benutzers wieder in ein Element in der Datenstruktur zu konvertieren. (Dieser Vorgang ist die Umkehrung des Aufrufs der Funktion `generateDynamicDataRef()`.) Wenn die Funktion `inspectDynamicDataRef()` ein Array mit zwei Elementen zurückgibt, bietet Dreamweaver visuelle Hinweise, welches Strukturelement mit der aktuellen Auswahl verbunden ist.

8 Immer wenn der Benutzer die Auswahl ändert, ruft Dreamweaver die Funktion `inspectDynamicDataRef()` auf. Auf diese Weise wird ermittelt, 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 Es ist möglich, das Datenformat für ein dynamisches Textobjekt oder ein dynamisches Attribut zu ändern, das der Benutzer bereits in die Seite eingefügt hat. Verwenden Sie dazu das Dialogfeld „Dynamische Daten“ bzw. „Dynamischer Text“ oder das Bedienfeld „Bindungen“. 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 326). Der von der Funktion `formatDynamicDataRef()` zurückgegebene String wird in das Dokument des Benutzers eingefügt.

Einfaches Beispiel für Datenquellen

Diese Erweiterung fügt für Adobe ColdFusion-Dokumente dem Bedienfeld „Bindungen“ eine benutzerdefinierte Datenquelle hinzu. Benutzer können die von der neuen Datenquelle abzurufende Variable angeben.

In diesem Beispiel wird eine Datenquelle mit dem Namen „MyDatasource“ erstellt, die die JavaScript-Datei „MyDatasource.js“, die Datei „MyDatasource_DataRef.edml“ sowie MyDatasource Variable-Befehlsdateien umfasst, um ein Dialogfeld zu implementieren, 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“.

Zum Erstellen dieser Datenquelle erstellen Sie eine Definitionsdatei, eine EDML-Datei, eine JavaScript-Datei und unterstützende Befehlsdateien. Anschließend testen Sie die neue Datenquelle.

Erstellen der Definitionsdatei für die Datenquelle

Die Definitionsdatei für die Datenquelle legt den Namen der Datenquelle fest, 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 zu sammeln, die in den HTML-Dateien („.htm“) definiert sind. Um Benutzern eine neue Datenquelle zur Verfügung zu stellen, müssen Sie daher eine Definitionsdatei für die Datenquelle erstellen, die mithilfe des Tags `title` den Namen der Datenquelle und mithilfe des Tags `script` den Speicherort aller unterstützenden JavaScript-Dateien enthält.

Außerdem sind zur Implementierung dieser Datenquelle mehrere unterstützende Dateien erforderlich. Im Allgemeinen sind die Funktionen in diesen unterstützenden Dateien nicht unbedingt erforderlich, jedoch oft hilfreich (und im folgenden Beispiel notwendig). Beispielsweise enthält die Datei „dwscriptsServer.js“ die Funktion `dwscripts.stripCFOutputTags()`, die zum Implementieren 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.

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“.

Erstellen der EDML-Datei

Die EDML-Datei definiert den Code, den Dreamweaver zur Darstellung des Datenquellenwerts in das Dokument einfügt. (Weitere Informationen zu EDML-Dateien finden Sie unter „Serververhalten“ auf Seite 262). Wenn ein Benutzer einem Dokument einen bestimmten Wert aus einer Datenquelle hinzufügt, fügt Dreamweaver Code ein, der zur Laufzeit durch den eigentlichen Wert ersetzt wird. Die EDML-Mitgliederdatei definiert den Code für das Dokument (weitere Informationen finden Sie unter „EDML-Mitgliederdateien“ auf Seite 282).

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 abrufen möchte.

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[^#]*#/i]]></searchPattern>
    </searchPatterns>
</participant>
```

3 Speichern Sie die Datei unter dem Namen „MyDatasource_DataRef.edml“ im Ordner „Configuration/DataSources/ColdFusion“.

Erstellen der JavaScript-Datei zum Implementieren der API-Funktionen für Datenquellen

Nachdem Sie den Namen der Datenquelle, den Namen der unterstützenden Skriptdatei und den Code für das Dreamweaver-Arbeitsdokument definiert haben, müssen Sie die JavaScript-Funktionen angeben, damit der Benutzer den erforderlichen Code in einem Dokument hinzufügen, einfügen und löschen kann.

Ausgehend vom Aufbau der Datenquelle „Cookie Variable“ können Sie, wie im folgenden Beispiel dargestellt, die Datenquelle „MyXML“ implementieren. (Der Befehl `MyDatasource_Variable`, der in der Funktion `addDynamicSource()` verwendet wird, wird definiert unter „Erstellen der unterstützenden Befehlsdateien für Benutzereingaben“ auf Seite 314.)

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)
        {

    // Here you create an instance of the DataSource class as defined in the
    // DataSourceClass.js file to store the return values.

        retList.push(new DataSource("MyDatasource",
                                    MyDatasource_FILENAME,
                                    false,
                                    "MyDatasource.htm"))
        }
    }

    return retList;
```

```
}

function generateDynamicSourceBindings(sourceName)
{
    var retVal = new Array();

    var siteURL = dw.getSiteRoot();

    // For localized object name...
    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);

    // We need to strip the cfoutput tags if we are inserting into a CFOUTPUT tag
    // or binding to the attributes of a ColdFusion tag. So, we use the
    // dwscripts.canStripCfOutputTags() function from dwscriptsServer.js

    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.js“ im Ordner „Configuration/DataSources/ColdFusion“.

Erstellen der unterstützenden Befehlsdateien für Benutzereingaben

Die Funktion `addDynamicSource()` enthält den Befehl `dw.popupCommand("MyDatasource_Variable")`, mit dem ein Dialogfeld geöffnet wird, in dem ein Benutzer einen bestimmten Variablennamen eingeben kann. Dazu müssen Sie jedoch das Dialogfeld „MyDatasource Variable“ erstellen.

Um ein Dialogfeld für Benutzereingaben bereitzustellen, erstellen Sie einen Satz Befehlsdateien: eine Befehlsdefinitionsdatei in HTML und eine Befehlsimplementierungsdatei in JavaScript (weitere Informationen zu Befehlsdateien finden Sie unter „Funktionsweise von Befehlen“ auf Seite 141).

Die Befehlsdefinitionsdatei legt den Speicherort der unterstützenden JavaScript-Dateien für die Implementierung fest. Darüber hinaus wird die Art des Dialogfelds definiert, 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.

Erstellen der Befehlsdefinitionsdatei

1 Erstellen Sie eine neue, leere Datei.

2 Geben Sie Folgendes ein:

```
<!DOCTYPE HTML SYSTEM "-//Adobe//DWExtension layout-engine 10.0//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 valign="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.*

Erstellen der unterstützenden 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()");
}

//***************** LOCAL FUNCTIONS*****************

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);
    }
}
```

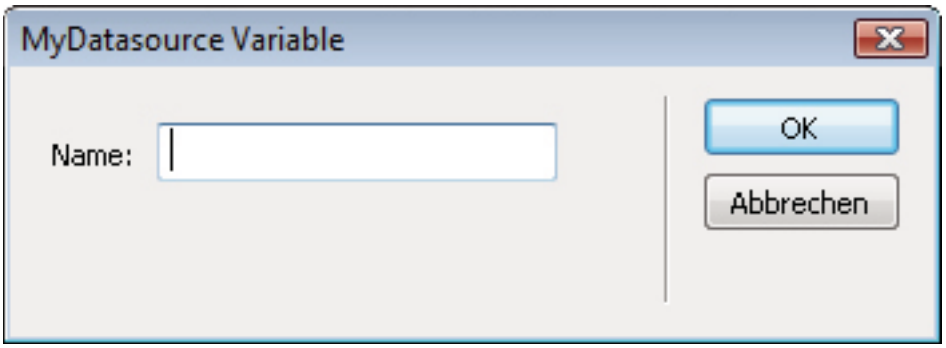
3 Speichern Sie die Datei unter dem Namen „MyDatasource_Variable.js“ im Ordner „Configuration/Commands“.

Testen der neuen Datenquelle

Sie können Dreamweaver nun öffnen (bzw. neu starten, wenn das Programm bereits geöffnet ist) und eine ColdFusion-Datei öffnen oder eine neue erstellen.

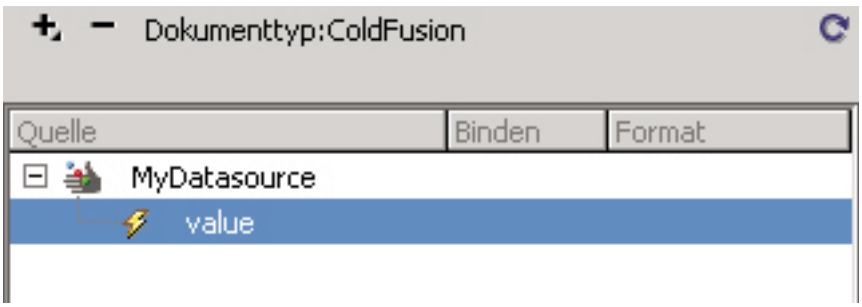
1 Klicken Sie auf das Menü mit dem Pluszeichen (+) des Bedienfelds „Bindungen“, um alle verfügbaren Datenquellen anzuzeigen. „MyDatasource“ sollte ganz unten in der Liste angezeigt werden.

2 Klicken Sie auf die Datenquelle „MyDatasource“. Daraufhin wird das von Ihnen erstellte Dialogfeld „MyDatasource Variable“ angezeigt:

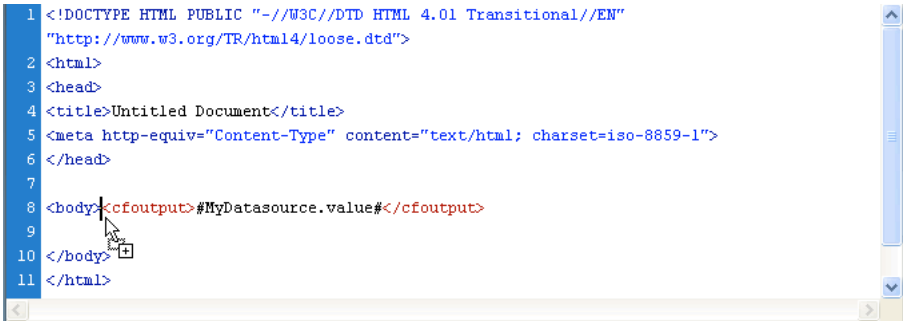


3 Geben Sie im Dialogfeld einen Wert ein und klicken Sie auf „OK“.

Die Datenquelle wird in einer Strukturansicht des Bedienfelds „Bindungen“ angezeigt. Dabei befindet sich die Variable aus dem Dialogfeld unterhalb des Datenquellennamens:



4 Ziehen Sie die Variable in Ihr Dokument. Dreamweaver fügt dann den entsprechenden Code aus der EDML-Datei ein:



API-Funktionen für Datenquellen

Über die Funktionen in der API für Datenquellen 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 jede Datenquellendatei eine Implementierung dieser Funktion enthält, ruft Dreamweaver die entsprechende Implementierung der Funktion `addDynamicSource()` auf, wenn im Menü mit dem Pluszeichen (+) eine Datenquelle ausgewählt wird.

Dreamweaver ruft für Datensatzgruppen oder Befehle beispielsweise die Funktion `dw.serverBehaviorInspector.popupServerBehavior()` auf, die in das Dokument ein neues Serververhalten einfügt. Für Anforderungsvariablen, Sitzungsvariablen und Anwendungsvariablen zeigt Dreamweaver ein HTML/JavaScript-Dialogfeld an, um die Eingabe des Variablennamens zu ermöglichen. Das Serververhalten speichert dann den Namen der Variable zur späteren Verwendung.

Nachdem die Funktion `addDynamicSource()` beendet ist, 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 in der Struktur auswählt und auf die Schaltfläche mit dem Minuszeichen (–) klickt.

Wenn in Dreamweaver z. B. eine Datensatzgruppe oder ein Befehl ausgewählt wurde, wird über die Funktion `deleteDynamicSource()` die Funktion `dw.serverBehaviorInspector.deleteServerBehavior()` aufgerufen. Wenn eine Anforderungsvariable, Sitzungsvariable oder Anwendungsvariable ausgewählt wurde, speichert die Funktion den Löschvorgang der Variable und zeigt diese nicht mehr an. Nachdem die Funktion `deleteDynamicSource()` beendet ist, löscht Dreamweaver den Inhalt der Datenquellenstruktur und ruft die Funktionen `findDynamicSources()` und `generateDynamicSourceBindings()` auf, um eine aktuelle Liste sämtlicher Datenquellen für das Dokument des Benutzers abzurufen.

Argumente
sourceName, *bindingName*

- Das Argument *sourceName* ist der Name des Knotens auf oberster Ebene, dem der untergeordnete Knoten zugeordnet ist.
- Das Argument *bindingName* ist der Name des untergeordneten Knotens.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

displayHelp()

Beschreibung
Wenn diese Funktion definiert ist, wird im Dialogfeld unter den Schaltflächen „OK“ und „Abbrechen“ die Schaltfläche „Hilfe“ angezeigt. Diese Funktion wird aufgerufen, wenn der Benutzer auf die Schaltfläche „Hilfe“ klickt.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
// The following instance of displayHelp() opens
// a file (in a browser) that explains how to use
// the extension.
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 vom Benutzer vorgenommene Änderungen in der Struktur zu verarbeiten. Andernfalls wird automatisch das Serververhalten aufgerufen, das der Datenquelle zugeordnet ist. Entwickler von Erweiterungen können mithilfe dieser Funktion die Standardimplementierung von Serververhalten überschreiben und eine benutzerdefinierte Verarbeitungsprozedur integrieren.

Argumente
sourceName, *bindingName*

- Das Argument *sourceName* ist der Name des Knotens auf oberster Ebene, 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 Bearbeitung erfolgreich abgeschlossen 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“ bzw. „Dynamischer Text“ oder im Bedienfeld „Bindungen“ angezeigt wird. Jede Datenquellendatei verfügt über eine Implementierung der Funktion `findDynamicSources()`. Wenn Dreamweaver die Struktur aktualisiert, werden alle Dateien im Ordner „DataSources“ gelesen und in jeder Datei wird die Funktion `findDynamicSources()` aufgerufen.

Rückgabewerte
Dreamweaver erwartet ein Array von JavaScript-Objekten, von denen jedes bis zu fünf Eigenschaften haben kann. Diese werden in der folgenden Aufstellung beschrieben.

- Die Eigenschaft `title` ist der Bezeichnungsstring, der rechts neben dem Symbol jedes übergeordneten Knotens angezeigt wird. Die Eigenschaft `title` muss immer angegeben werden.
- Die Eigenschaft `imageFile` ist der Pfad zu einer Datei, die ein Symbol (ein GIF-Bild) enthält, mit der der übergeordnete Knoten im Struktursteuerelement des Dialogfelds „Dynamische Daten“ bzw. „Dynamischer Text“ oder 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. Wenn diese Eigenschaft auf `true` gesetzt ist, 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 (sofern vorhanden). Einige Datenquellen (z. B. Datensatzgruppen) sind mit Serververhalten verknüpft. Wenn Sie eine Datensatzgruppe erstellen und ihr den Namen `rsAuthors` zuweisen, muss das Attribut `name` mit dem Wert von `rsAuthors` übereinstimmen. Die Eigenschaft `name` ist stets definiert, kann jedoch auch ein leerer String (`""`) sein, wenn mit der Datenquelle (z. B. einer Sitzungsvariable) 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 Knotens auf oberster Ebene, der mit dem untergeordneten Knoten verknüpft ist.
- 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()` übergeben werden kann, bevor er in das Dokument eines Benutzers eingefügt wird.

generateDynamicSourceBindings()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Diese Funktion gibt die untergeordneten Knoten eines Knotens auf oberster Ebene zurück.

Argumente
sourceName

- Das Argument *sourceName* ist der Name des Knotens auf oberster Ebene, dessen untergeordnete Knoten zurückgegeben werden sollen.

Rückgabewerte
Dreamweaver erwartet ein Array von JavaScript-Objekten, von denen jedes bis zu vier Eigenschaften haben kann. Diese werden in der folgenden Aufstellung 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` gesetzt ist und der Benutzer im Bedienfeld „Bindungen“ auf diesen Knoten klickt, wird die Schaltfläche mit dem Minuszeichen (–) deaktiviert. Wenn diese Eigenschaft auf den Wert `true` gesetzt ist, 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 (sofern vorhanden). Diese Eigenschaft ist erforderlich. Einige Datenquellen (z. B. Datensatzgruppen) sind mit Serververhalten verknüpft. Wenn Sie eine Datensatzgruppe erstellen und ihr den Namen `rsAuthors` zuweisen, muss die Eigenschaft `name` mit dem Wert von `rsAuthors` übereinstimmen. Andere Datenquellen (z. B. Sitzungsvariablen) verfügen nicht über entsprechende Serververhalten. Die name-Eigenschaft dieser Datenquellen 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 ermittelt für ein dynamisches Datenobjekt den entsprechenden Knoten in der Datenquellenstruktur. Die Funktion `inspectDynamicDataRef()` vergleicht den von Dreamweaver übergebenen String mit dem String, der von `generateDynamicDataRef()` für jeden Knoten in der Struktur zurückgegeben wird. Wenn eine Übereinstimmung gefunden wird, zeigt die Funktion `inspectDynamicDataRef()` an, welcher Knoten der Struktur mit dem übergebenen String übereinstimmt. Der Knoten wird mithilfe eines Arrays ermittelt, 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 mit dem eigenen Objekttyp. Die Datensatzgruppen-Implementierung der Funktion `inspectDynamicDataRef()` findet beispielsweise nur 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 von 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 308 wird erläutert, wie in Adobe Dreamweaver dynamische Daten in ein Dokument des Benutzers eingefügt werden, indem an der entsprechenden Stelle ein Serverausdruck hinzugefügt wird. Wenn das Dokument des Benutzers anschließend von einem Webserver abgerufen wird, wird dieser Serverausdruck in einen Wert aus einer Datenbank, den Inhalt einer Anforderungsvariable oder einen anderen dynamischen Wert konvertiert. Mit den Serverformaten von Dreamweaver können Sie festlegen, wie dieser dynamische Wert angezeigt wird.

Funktionsweise der Datenformatierung

Benutzer von Dreamweaver können Daten mithilfe integrierter Formate formatieren. Sie können darüber hinaus auf Grundlage der integrierten Formattypen oder mithilfe benutzerdefinierter Formattypen neue Formate erstellen.

Dynamische Daten können auf mehrere Arten formatiert werden. Verwenden Sie das Menü „Format“, um Daten zu formatieren, bevor sie in ein HTML-Dokument eingefügt werden. Das Menü „Format“ befindet sich im Dialogfeld „Dynamische Daten“ bzw. „Dynamischer Text“ oder im Bedienfeld „Bindungen“. Um ein Format zu erstellen, wählen Sie im Menü „Format“ die Option „Formatliste bearbeiten“ aus und klicken Sie dann im Menü mit dem Pluszeichen (+) auf den gewünschten Formattyp. Das Menü mit dem Pluszeichen (+) enthält eine Liste mit Formattypen. Bei Formattypen handelt es sich um grundlegende Formatkategorien, z. B. „Währung“, „Datum/Uhrzeit“ oder „Groß-/Kleinschreibung“. In Formattypen sind alle gemeinsamen Parameter einer Formatkategorie zusammengefasst. Dies vereinfacht das Erstellen eines Formats.

Ein Beispiel dafür ist das Erstellen eines Währungsformats. Die Währungsformatierung besteht aus folgenden Schritten: Konvertieren einer Zahl in einen String und Einfügen von Trennzeichen und Dezimalkommas 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.

Im Abschnitt zur API für Serverformate wird die API erläutert, mit der die dynamischen Daten formatiert werden, die von den unter „Datenquellen“ auf Seite 308 beschriebenen Funktionen zurückgegeben werden. Durch das Zusammenspiel der in beiden Abschnitten beschriebenen Funktionen werden dynamische Daten formatiert.

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 Auswahlmöglichkeiten im Menü „Format“ beschrieben. Dreamweaver fügt automatisch die Optionen „Formatliste bearbeiten“ und „Keine“ hinzu.

In diesem Ordner befindet sich außerdem eine HTML-Datei für jeden derzeit installierten Formattyp. Dazu gehören `Groß-/Kleinschreibung`, `Währung`, `Datum/Uhrzeit`, `Mathematik`, `Zahl`, `Prozent`, `Einfach` und `Feineinstellung`.

Hinweis: *Das Format „Währung“ ist nicht für PHP-Servermodelle verfügbar.*

Datei „Formats.xml“

Die Datei „Formats.xml“ enthält ein `format`-Tag für jedes Element im Menü „Format“. Jedes `format`-Tag enthält die folgenden obligatorischen Attribute:

- Das Attribut `file=`*`fileName`* ist die HTML-Datei für diesen Formattyp, z. B. `"Currency"`.
- Das Attribut `title=`*`string`* ist der String, der im Menü „Format“ angezeigt wird, z. B. `"Currency - default"`.

- Das Attribut `expression=`*`regexp`* ist ein regulärer Ausdruck, mit dem dynamische Datenobjekte gefunden werden, die dieses Format verwenden. Der Ausdruck legt fest, welches Format derzeit auf ein dynamisches Datenobjekt angewendet wird. Der Ausdruck für das Format `"Currency - default"` 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]`* gibt 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 `format`-Tag 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 `format`-Tag in der Datei „Formats.xml“ entspricht. Das Objekt enthält eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden `format`-Tags.

Im folgenden Beispiel ist das `format`-Tag für den String `"Currency - default"` dargestellt:

```
<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 `Currency`. Der String `"Currency - default"` 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 zwingend erforderliche Parameter für den Formattyp `Currency`. Diese Parameter werden im Dialogfeld „Parameter“ für den Formattyp `Currency` angezeigt. Das Dialogfeld „Parameter“ wird angezeigt, wenn ein Benutzer den Formattyp `Currency` im Menü mit dem Pluszeichen (+) des Dialogfelds „Formatliste bearbeiten“ auswählt. Mit den für diese Parameter angegebenen Werten wird das neue Format definiert.

Menü mit dem Pluszeichen (+) des Dialogfelds „Formatliste bearbeiten“

Wenn eine Datei im Ordner „ServerFormats“ nicht im Menü mit dem Pluszeichen (+) des Dialogfelds „Formatliste bearbeiten“ angezeigt werden soll, fügen Sie den folgenden String als erste Zeile in die HTML-Datei ein:

```
<!-- MENU-LOCATION=NONE -->
```

Um den Inhalt des Menüs zu ermitteln, sucht Dreamweaver zunächst nach der Datei „ServerFormats.xml“ in dem Ordner, in dem sich die Datenformate befinden (z. B. im Ordner „Configuration/ServerFormats/ASP/“). Die Datei „ServerFormats.xml“ beschreibt den Inhalt des Menüs mit dem Pluszeichen (+) für das Dialogfeld „Formatliste bearbeiten“. Sie enthält Verweise auf die HTML-Dateien, die im Menü aufgeführt werden.

Dreamweaver prüft jede aufgeführte HTML-Datei auf `title`-Tags. Wenn die Datei ein title-Tag enthält, wird im Menü der Inhalt des title-Tags angezeigt. Wenn die Datei kein title-Tag enthält, wird im Menü der Dateiname angezeigt.

Nachdem Dreamweaver die Suche nach der Datei abgeschlossen hat oder wenn die Datei nicht vorhanden ist, wird der Rest des Ordners nach anderen Elementen durchsucht, die im Menü angezeigt werden sollen. Wenn Dreamweaver Dateien im Hauptordner findet, die sich nicht bereits im Menü befinden, werden sie dem Menü hinzugefügt. Wenn Unterordner Dateien enthalten, die sich nicht bereits im Menü befinden, erstellt Dreamweaver ein Untermenü und fügt diese Dateien in das Untermenü ein.

Szenarios mit 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 in der Datenquellenstruktur und im Menü „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 in der Einstellung „Code“ angezeigt. Wenn der Benutzer auf „OK“ klickt, wird der Codestring in das Dokument des Benutzers eingefügt. Dann ruft Dreamweaver die Funktion `applyFormat()` auf, um eine Funktionsdeklaration einzufügen. Weitere Informationen finden Sie unter „generateDynamicDataRef()“ auf Seite 320. Ein ähnliches Verfahren wird durchgeführt, wenn der Benutzer das Bedienfeld „Bindungen“ verwendet.
- Wenn der Benutzer das Format ändert oder das dynamische Datenelement 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*“.

  Wenn der Benutzer im Menü mit dem Pluszeichen (+) ein Format auswählt, 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 das Dialogfeld „Parameter“ nicht angezeigt werden muss, ruft Dreamweaver die Funktion `applyFormatDefinition()` auf, wenn der Benutzer den Formattyp im Menü mit dem Pluszeichen (+) auswählt.
- Wenn der Benutzer später das Format bearbeitet (durch Auswählen des Formats im Dialogfeld „Formatliste bearbeiten“ und Klicken auf die Schaltfläche „Bearbeiten“), ruft Dreamweaver vor der Anzeige des Dialogfelds „Parameter“ die Funktion `inspectFormatDefinition()` auf, damit die Formularsteuerelemente auf die korrekten Werte initialisiert werden können.

API-Funktionen für Serverformate

Die API für Serverformate umfasst die folgenden Funktionen zur Datenformatierung.

applyFormat()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Mit dieser Funktion wird das 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: Es wird die entsprechende Formatfunktion vor dem HTML-Tag hinzugefügt (sofern diese nicht bereits vorhanden ist) und das dynamische Datenobjekt geändert, damit die entsprechende Formatfunktion aufgerufen wird.

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 am Anfang des Benutzerdokuments hinzuzufügen. Wenn der Benutzer beispielsweise „Currency - Default“ auswählt, fügt die Funktion die Funktionsdeklaration `Currency` hinzu.

Argumente
format

- Das Argument *format* ist ein JavaScript-Objekt, das das anzuwendende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem `format`-Tag in der Datei „Formats.xml“ entspricht. Das Objekt enthält eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden `format`-Tags.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

applyFormatDefinition()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Übernimmt die Änderungen an einem Format, das über das Dialogfeld „Formatliste bearbeiten“ erstellt wurde.

Im 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. Zudem können mit dieser Funktion zusätzliche, willkürlich benannte Eigenschaften für das Objekt festgelegt werden. Jede Eigenschaft wird als Attribut des `format`-Tags in der Datei „Formats.xml“ gespeichert.

Argumente
format

- Das Argument *format* entspricht dem JavaScript-Objekt `format`. Die Funktion muss die Eigenschaft `expression` des JavaScript-Objekts auf den regulären Ausdruck für das Format setzen. Zudem können mit dieser Funktion zusätzliche, willkürlich benannte Eigenschaften für das Objekt festgelegt werden. Die einzelnen Eigenschaften werden als Attribute des `format`-Tags gespeichert.

Rückgabewerte
Dreamweaver erwartet das Formatobjekt, wenn die Funktion erfolgreich ausgeführt wurde. Wenn ein Fehler auftritt, gibt die Funktion einen Fehlerstring zurück. Wenn ein leerer String zurückgegeben wird, wird das Formular geschlossen, ohne dass das neue Format erstellt wird. Dies entspricht dem Vorgang „Abbrechen“.

deleteFormat()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Entfernt die Format-Funktionsdeklaration aus dem Anfangsbereich des Benutzerdokuments.

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 Anfangsbereich des Dokuments zu entfernen und den Funktionsaufruf aus dem dynamischen Datenobjekt zu löschen.

Verwenden Sie das DOM mit der Funktion `deleteFormat()`, um die Funktionsdeklaration aus dem Anfangsbereich des aktuellen Dokuments 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 `format`-Tag in der Datei „Formats.xml“ entspricht.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

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: Es wird die entsprechende Formatfunktion vor dem HTML-Tag hinzugefügt (sofern diese nicht bereits vorhanden ist) und das dynamische Datenobjekt geändert, damit die entsprechende Formatfunktion aufgerufen wird.

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 Textfeld „Format“ des Dialogfelds „Dynamische Daten“ bzw. „Dynamischer Text“ oder des Bedienfelds „Bindungen“ auswählt. Dabei wird das Dokument des Benutzers nicht geändert.

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 `format`-Tag in der Datei „Formats.xml“ entspricht. Das Objekt enthält eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden `format`-Tags.

Rückgabewerte
Dreamweaver erwartet den neuen Wert für das dynamische Datenobjekt.

Wenn ein Fehler auftritt, zeigt die Funktion unter bestimmten Bedingungen eine Warnmeldung an. Wenn die Funktion einen leeren String zurückgibt, wurde im Textfeld „Format“ die Option „Kein“ ausgewählt.

inspectFormatDefinition()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Initialisiert Formularsteuerelemente, wenn der Benutzer ein Format im Dialogfeld „Formatliste bearbeiten“ bearbeitet.

Argumente
format

Das Argument *format* ist ein JavaScript-Objekt, das das anzuwendende Format beschreibt. Das JavaScript-Objekt ist der Knoten, der dem `format`-Tag in der Datei „Formats.xml“ entspricht. Das Objekt enthält eine JavaScript-Eigenschaft für jedes Attribut des entsprechenden `format`-Tags.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Kapitel 20: Komponenten

Mit Adobe Dreamweaver 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. Sie können sich die *Kapselung* wie das Erzeugen eines in einer virtuellen Blackbox verborgenen Objekts vorstellen. Zur Verwendung des Objekts müssen Sie dessen Funktionsweise nicht kennen. Vielmehr müssen Sie wissen, welche Informationen für das Funktionieren erforderlich sind und welche Informationen nach Abschluss des Vorgangs ausgegeben werden. Beispiel: Ein Programmierer schreibt ein Programm, mit dem Daten aus einer Mitarbeiterdatenbank abgerufen werden. Alle Personen sowie andere Programme können dann mithilfe dieses Programms die Datenbank abfragen. Dieses Programm ist somit wiederverwendbar.

Die Erfahrung hat gezeigt, dass gut strukturierte Programme mit Kapselung einfacher gewartet, verbessert und wiederverwendet werden können. Verschiedene Technologien bieten Programmierern unterschiedliche Möglichkeiten zur Kapselung. Diese Strategien werden unterschiedlich bezeichnet, u. a. als *Funktionen* oder *Module*. In Dreamweaver wird der Begriff *Komponente* für einige der verbreiteteren und moderneren Kapselungsstrategien wie z. B. Adobe ColdFusion-Komponenten (CFCs) verwendet. Benutzer werden daher beim Erstellen von Webanwendungen in Dreamweaver durch CFCs im Bedienfeld „Komponenten“ unterstützt.

Komponenten aus neuen Technologien (z. B. Webdienste, JavaBeans oder CFCs) sind selbsterklärend. In der Regel sind Informationen zur jeweiligen Komponente in die zugehörigen Dateien integriert. Die Fähigkeit einer Komponente, diese Informationen zu veröffentlichen oder freizugeben, wird als *Introspektion* bezeichnet. Somit kann mit einem Programm wie Dreamweaver aus einer Komponente die Liste der bereitgestellten Funktionen abgerufen werden. (Bereitgestellte Funktionen sind Funktionen, die von einem anderen Programm geladen werden können). Je nach verwendeter Technologie können Komponenten auch andere Informationen über sich selbst bereitstellen.

Erweitern des Bedienfelds „Komponenten“

Wenn Sie ein Komponentenverfahren verwenden möchten, das im Bedienfeld „Komponenten“ noch nicht vorhanden ist, können Sie den Code des Bedienfelds erweitern. Auf diese Weise können in diesem Bedienfeld neue Komponententypen bearbeitet werden.

Wenn Sie eine neue Komponente zum Bedienfeld „Komponenten“ in Dreamweaver hinzufügen möchten, suchen Sie die verfügbaren Komponenten (in der Benutzerumgebung). Rufen Sie dann Beschreibungen der einzelnen Komponenten ab (oder analysieren Sie sie, wenn sie mithilfe von ASCII-Dateien programmiert 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 zum Erweitern des Bedienfelds „Komponenten“ programmierte JavaScript-Code hängt somit von der jeweils hinzuzufügenden Komponententechnologie ab. Die hier beschriebenen Funktionen sollen Sie beim Abrufen von Informationen zur Anzeige im Bedienfeld „Komponenten“ unterstützen. Den Code zum Suchen und Überprüfen von Komponenten müssen Sie jedoch größtenteils selbst programmieren. Dazu gehören die Abfrage der internen Komponentenstruktur und das Bereitstellen der zugehörigen Felder, Methoden und Eigenschaften in Dreamweaver.

Schließlich unterstützen Servermodelle wie ASP.NET, JSP/J2EE und ColdFusion einige, jedoch nicht alle Komponententypen. ASP.NET unterstützt beispielsweise Webdienste, jedoch nicht JavaBeans. Adobe ColdFusion unterstützt Webdienste und CFCs. Wenn Sie einen neuen Komponententyp zum Bedienfeld „Komponenten“ hinzufügen, muss dieser für das entsprechende Servermodell geeignet sein. Wenn ein Dreamweaver-Benutzer beispielsweise eine ColdFusion-Website bearbeitet, werden im Popupmenü des Bedienfelds „Komponenten“ die CF-Komponenten angezeigt.

In einigen Fällen müssen Sie zum Ändern von Dateien JavaScript-Code programmieren, mit dem bestimmte komponentenspezifische Funktionen aufgerufen werden.

Anpassen des Bedienfelds „Komponenten“

Im Bedienfeld „Komponenten“ von Dreamweaver können Benutzer Komponenten laden und bearbeiten. In diesem Bedienfeld sind alle verfügbaren Komponententypen aufgeführt, die mit dem jeweils aktivierten Servermodell kompatibel sind. CFCs sind beispielsweise nur für Adobe ColdFusion-Seiten geeignet und werden daher nur für das Adobe ColdFusion-Servermodell im Bedienfeld „Komponenten“ angezeigt.

Im Rahmen der Erweiterbarkeit können Sie neue Komponententypen in das Bedienfeld aufnehmen. Zum Hinzufügen eines neuen Komponententyps zum Bedienfeld „Komponenten“ müssen mehrere allgemeine Schritte ausgeführt werden:

1. Fügen Sie die Komponente zur Liste der verfügbaren Komponententypen für die entsprechenden Servermodelle hinzu.
2. Fügen Sie Anweisungen zum Einrichten der Komponente im Bedienfeld „Komponenten“ oder in einem Dialogfeld hinzu (abhängig von der Erweiterung, für die diese Schritte implementiert werden). Die Anweisungen werden auch als Setup-Schritte bezeichnet und als interaktive, nummerierte Schritte angezeigt. Stellen Sie sicher, dass neben allen durch den Benutzer ausgeführten Schritten Häkchen angezeigt werden.
3. Listen Sie die Komponenten des entsprechenden Komponententyps auf, die entweder nur auf dem Computer des Benutzers oder nur in der aktuellen Site vorhanden sind.
4. Erstellen Sie eine Komponente, wenn der Benutzer im Bedienfeld „Komponenten“ auf die Schaltfläche mit dem Pluszeichen (+) klickt.

   Darüber hinaus können Sie dem Benutzer ermöglichen, Komponenten zu bearbeiten und zu löschen.

Anpassen von Dateien für das Bedienfeld „Komponenten“

Der Ordner „Configuration“ enthält einen Unterordner für jedes implementierte Servermodell. Komponentendateien werden im Ordner „Configuration/Components/*Servermodell*/*Komponententyp*“ gespeichert. Sie können andere Servermodelle und unterstützende Servererweiterungen hinzufügen. (Weitere Informationen finden Sie unter „Servermodelle“ auf Seite 342 und „Serververhalten“ auf Seite 262.)

Erstellen einer benutzerdefinierten Komponente für das Bedienfeld „Komponenten“

1. Erstellen Sie eine HTML-Datei, die die Speicherorte der unterstützenden JavaScript- und Bilddateien angibt.
2. Programmieren Sie den JavaScript-Code zur Aktivierung der Komponente.
3. Geben Sie bereits vorhandene GIF-Bilddateien für die Darstellung der Komponente im Bedienfeld „Komponenten“ an oder erstellen Sie sie.

Wenn der Komponententyp in einer Strukturansicht angezeigt werden soll, müssen Sie die verknüpften optionalen Dateien erstellen und die Daten in die Strukturansicht übernehmen.

Sie können einen Komponententyp auf drei Ebenen einrichten: für eine einzelne Webseite, für eine Gruppe von Webseiten oder für die gesamte Website. Der 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.

Hinzufügen einer neuen LDAP-Dienstkomponente (Lightweight Directory Access Protocol)

1. Erstellen Sie anhand vorhandener Komponentendateien (z. B. der Dateien im Anwendungsordner „Configuration/Components/Common/WebServices“) alle erforderlichen Dateien und die gewünschten optionalen Dateien, um den neuen Komponententyp im Bedienfeld „Komponenten“ von Dreamweaver anzuzeigen. Siehe dazu folgende Tabelle:

Dateiname	Beschreibung	Erforderlich/Optional
*.htm	Die Erweiterungsdatei, die andere unterstützende JavaScript- und GIF-Dateien angibt.	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	Das Repository 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	Symbolleistenbilder, die ein- oder ausgeblendet werden können, wie im folgenden Beispiel dargestellt: `ToolBarImageUp.gif ToolBarImageDown.gif ToolBarImageDisabled.gif` Oder Strukturknotenbilder.	Optional

Hinweis: *Sie sollten dasselbe Präfix für alle Dateien einer Komponente verwenden, sodass diese Dateien problemlos ihren jeweiligen Komponenten zugeordnet werden können.*

2. Programmieren Sie den JavaScript-Code zum Implementieren der neuen Serverkomponente.

Die Erweiterungsdatei (HTM) legt den Speicherort der Dateien mit JavaScript-Code im `SCRIPT`-Tag fest. 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 API-Funktionen für das Bedienfeld „Komponenten“ finden Sie unter „API-Funktionen für das Bedienfeld „Komponenten““ auf Seite 332.

Beim Hinzufügen eines neuen Diensts sollten Sie im Bedienfeld „Komponenten“ die Metadaten durchsuchen, sodass die Daten beim Erstellen der Erweiterung sofort verfügbar sind. Dreamweaver kann hinzugefügte Komponenten durchsuchen und Knoten in der Komponentenstruktur anzeigen. Die Funktion „Ziehen und Ablegen“ sowie die Tastatursteuerung werden in der Codeansicht des Bedienfelds „Komponenten“ unterstützt.

Eigenschaften von Struktursteuerelementen

Mithilfe der Eigenschaft `ComponentRec` können Sie ein Struktursteuerelement des Bedienfelds „Komponenten“ füllen, sodass es in diesem Bedienfeld an der richtigen Position angezeigt wird. Jeder Knoten in einem Struktursteuerelement muss folgende Eigenschaften aufweisen:

Name der Eigenschaft	Beschreibung	Erforderlich/Optional
`name`	Name des Strukturknotenelements.	Erforderlich
`image`	Symbol für das Strukturknotenelement. Wenn es nicht angegeben ist, wird ein Standardsymbol verwendet.	Optional
`hasChildren`	Beim Klicken auf die Schaltfläche mit dem Pluszeichen (+) oder Minuszeichen (-) im Struktursteuerelement werden untergeordnete Elemente geladen. Sie können eine Struktur verwenden, die nicht bereits gefüllt ist.	Erforderlich
`toolTipText`	QuickInfo-Text für das Strukturknotenelement.	Optional
`isCodeViewDraggable`	Legt fest, ob das Element durch Ziehen in die Codeansicht verschoben werden kann.	Optional
`isDesignViewDraggable`	Legt fest, 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;
// the following allows of enabling/disabling of the button that appears
// above the Component Tree
this.allowDelete = true;
this.isDesignViewDraggable = false;
```

API-Funktionen für das Bedienfeld „Komponenten“

In diesem Abschnitt werden die API-Funktionen zum 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 Stammebene geladen werden können, muss diese Funktion Zugriff auf die zugehörigen Metadaten im permanenten Speicher haben.

Argumente
`{parentComponentRec}`

Das Argument *parentComponentRec* ist das Objekt `componentRec` des übergeordneten Elements. Wenn es nicht angegeben ist, 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
Gibt die ID des Kontextmenüs für den Komponententyp zurück. Jedem Komponententyp kann ein Kontextmenü zugeordnet sein. Die Popupmenüs für das Kontextmenü sind in der Datei „*Komponentenname*Menus.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 angibt.

Beispiel

Im folgenden Beispiel wird das Menü „Optionen“ des Bedienfelds „Komponenten“ für CFCs festgelegt, die mit dem Adobe ColdFusion-Servermodell verknüpft sind. Im Beispielcode werden auch die Tastaturbefehle für die Menüeintrage definiert:

```
function getContextMenuId()
{
    return "DWCFCsContext";
}
```

„DWWebServicesContext“ für das Menü wird in der Datei „Configuration/Components/ColdFusion/CFCs/CFCsMenus.xml“ wie folgt definiert:

```
<menubar xmlns:MMString="http://www.macromedia.com/schemes/dat/string/" name=""
id="DWCFCsContext">
    <menu MMString:name="Components/ColdFusion/CFCs/CFCsMenus_xml/DWContext_CFCs/menu/name"
id="DWContext_CFCs">
    <menuitem
MMString:name="Components/ColdFusion/CFCs/CFCsMenus_xml/DWContext_CFCs_createNewCFC/menuitem
/name" domRequired="false" enabled="true" command="createCFC()"
id="DWContext_CFCs_createNewCFC" />
    <menuitem
MMString:name="Components/ColdFusion/CFCs/CFCsMenus_xml/DWContext_CFCs_editCode/menuitem/nam
e" domRequired="false" enabled="canGetSelectedCFC()" command="editCFC();"
id="DWContext_CFCs_editCode" />
<separator/>
    <menuitem
MMString:name="Components/ColdFusion/CFCs/CFCsMenus_xml/DWContext_CFCs_getDetails/menuitem/n
ame" domRequired="false" enabled="canGetDetails()" command="getDetails()"
id="DWContext_CFCs_getDetails" />
    <menuitem
MMString:name="Components/ColdFusion/CFCs/CFCsMenus_xml/DWContext_CFCs_getDescription/menuit
em/name" domRequired="false" enabled="canGetSelectedCFC()" command="getDescription()"
id="DWContext_CFCs_getDescription" />
    <separator/>
    <menuitem
MMString:name="Components/ASP_NET_Csharp/Connections/ConnectionsMenus_xml/DWShortcuts_Server
Component_Insert/menuitem/name" domRequired="false" enabled="insertCFCEnabled();"
command="clickedInsertCFC();" id="DWShortcuts_ServerComponent_Insert" />
    </menu>
</menubar>
```

getCodeViewDropCode()

Verfügbarkeit

Dreamweaver MX.

Beschreibung

Diese Funktion ruft den Code ab, der aus dem Bedienfeld „Komponenten“ in die Codeansicht gezogen bzw. über den Befehl „Ausschneiden“ in die Codeansicht kopiert wird.

Argumente

componentRec

- Das Argument *componentRec* ist ein Objekt.

Rückgabewerte

Der String, der den Code für die Komponente enthält.

Beispiel

Im folgenden Beispiel ist der Code für eine Adobe ColdFusion-Komponente (CFC) angegeben:

```
  function getCodeVi e wDropCode(compone nt  Rec )
{
    var codeToDrop="";
    if (componentRec)
     {
        if  (componentRec.objectType == "Connection")
        {
            v ar  connPart = new Participant("d at asource_tag");
            var paramObj = new Obje ct( );
            paramObj.datasource = componentRec.name;
            codeToDrop =  con nPart.getInsertString(paramObj, "aboveHTML");}
        el se if ((componentRec.objectType == "Column")| |
                 (componentRec.objectType == "Paramete r "))
        {
            c  odeToDrop = componentRec.dropcode;
        }
        e ls e{
            codeToDrop =  co mponentRec.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.

Die Funktion gibt ein Array der Strings zurück, die je nach Erweiterungstyp in Dreamweaver im Dialogfeld mit den Setup-Anweisungen oder im Bedienfeld „Komponenten“ angezeigt werden.

Argumente

Keine.

Rückgabewerte

Ein Array von *n*+1 Strings, wobei *n* die Anzahl der auszuführenden Schritte ist (siehe Beschreibung in der folgenden Aufstellung).

- Der Titel, der oberhalb der Setup-Schritte angezeigt wird.
- Die Textanweisungen 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 einfügen:

  ```
  <a href="#" onMouseDown="handler">Blue Underlined Text</a>
  ```

  Der Wert *"handler"* kann durch einen der folgenden Strings 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.
- Eine `"Event:SetRDSPassword"`-Ereignisprozedur öffnet ein Dialogfeld, in dem der Benutzername und das Kennwort für den Remote Development Service (RDS) festgelegt werden können (nur Adobe ColdFusion).
- Eine `"Event:CreateCFDataSource"`-Ereignisprozedur öffnet den Adobe ColdFusion-Administrator in einem Browser.

Beispiel

Im folgenden Beispiel werden vier Schritte für Adobe ColdFusion-Komponenten festgelegt. Der vierte Schritt enthält einen Hyperlink, damit Benutzer den RDS-Benutzernamen und das entsprechende Kennwort eingeben können:

```
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 das Bedienfeld „Komponenten“ angezeigt wird. Anschließend wird die Funktion `getSetupSteps()` aufgerufen, wenn die Funktion `setupStepsCompleted()` den Wert Null oder eine positive Ganzzahl zurückgibt.

Argumente
Keine.

Rückgabewerte
Eine Ganzzahl, die die Anzahl der Setup-Schritte angibt, die der Benutzer bereits ausgeführt hat, wie in der folgenden Aufstellung beschrieben:

- Der Wert Null oder eine positive Ganzzahl gibt die Anzahl der bereits ausgeführten Schritte an.
- Der Wert `-1` gibt an, dass alle erforderlichen Schritte ausgeführt wurden. In diesem Fall wird die Anweisungsliste nicht 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. Wenn es nicht vorhanden ist, verwendet Dreamweaver MX ein Standardsymbol.
- Die Eigenschaft `hasChildren` ist ein boolescher Wert, der angibt, ob das Strukturknotenelement erweiterbar ist. Wenn die Eigenschaft den Wert `true` aufweist, zeigt Dreamweaver MX für das Strukturknotenelement die Schaltflächen mit dem Pluszeichen (+) und Minuszeichen (-) an. Bei `false` ist das Element nicht erweiterbar.
- Die Eigenschaft `toolTipText` ist eine optionale QuickInfo für das Strukturknotenelement.
- Die Eigenschaft `isCodeViewDraggable` ist ein boolescher Wert, der angibt, ob ein Knotenelement durch Ziehen und Ablegen in die Codeansicht verschoben werden kann.
- Die Eigenschaft `isDesignViewDraggable` ist ein boolescher Wert, der angibt, ob ein Knotenelement durch Ziehen und Ablegen in die Entwurfsansicht verschoben werden kann.

Rückgabewerte
Ein boolescher Wert, der angibt, ob der Ablegevorgang erfolgreich war: `true`, wenn der Vorgang erfolgreich durchgeführt wurde, andernfalls `false`.

Beispiel
Im folgenden Beispiel wird ermittelt, 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. Wenn das Symbol nicht angegeben ist, verwendet Dreamweaver ein Standardsymbol.
- Die Eigenschaft `hasChildren` ist ein boolescher Wert, der angibt, ob das Strukturknotenelement erweiterbar ist. Wenn die Eigenschaft den Wert `true` aufweist, zeigt Dreamweaver für das Strukturknotenelement die Schaltflächen mit dem Pluszeichen (+) und Minuszeichen (-) an. Bei `false` ist das Element nicht erweiterbar.
- Die Eigenschaft `toolTipText` ist ein optionaler QuickInfo-Text für das Strukturknotenelement.
- Die Eigenschaft `isCodeViewDraggable` ist ein boolescher Wert, der angibt, ob das Strukturknotenelement durch Ziehen und Ablegen in die Codeansicht verschoben werden kann.
- Die Eigenschaft `isDesignViewDraggable` ist ein boolescher Wert, der angibt, ob ein Strukturknotenelement durch Ziehen und Ablegen in die Entwurfsansicht verschoben werden kann.

Rückgabewerte
Keine.

Beispiel
Im folgenden Beispiel reagiert die Erweiterung auf einen Doppelklick auf Strukturknotenelemente. Wird der Wert `false` zurückgegeben, werden die Knoten ein- oder ausgeblendet (Standardverhalten).

```
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 geben von links nach rechts die Symbole der Symbolleiste an. Jedes `toolBarButtonRec`-Objekt enthält die folgenden Eigenschaften:

Name der Eigenschaft	Beschreibung
`image`	Pfad zur Bilddatei
`disabledImage`	(Optional) Pfad zum Bild für die deaktivierte Schaltfläche – sucht nach der Symbolleistenschaltfläche
`pressedImage`	(Optional) Pfad zum Bild für die gedrückte Schaltfläche – sucht nach der Symbolleistenschaltfläche
`toolTipText`	QuickInfo für die Symbolleistenschaltfläche
`toolStyle`	Links/rechts

Name der Eigenschaft	Beschreibung
`enabled`	JavaScript-Code, der einen booleschen Wert zurückgibt (`true` oder `false`). Die Enabler werden unter folgenden Umständen aufgerufen: • Wenn die Funktion `dreamweaver.serverComponents.refresh()` aufgerufen wird • Wenn sich die Auswahl in der Struktur ändert • Wenn sich das Servermodell ändert
`command`	Der auszuführende JavaScript-Code. Die Befehlsprozedur kann eine Aktualisierung mit der Funktion `dreamweaver. serverComponents.refresh()` erzwingen.
`menuId`	Die eindeutige Menü-ID für die Schaltfläche des Popupmenüs, wenn auf die Schaltfläche geklickt wird. Wenn diese ID vorhanden ist, setzt sie die Befehlsprozedur außer Kraft. Mit anderen Worten, die Schaltfläche kann entweder mit einem Befehl oder mit einem Popupmenü verknüpft sein, nicht jedoch mit beiden gleichzeitig.

Argumente

Keine.

Rückgabewerte

Ein Array von Symbolleistenschaltflächen in der Reihenfolge von links nach rechts.

Beispiel

Im folgenden Beispiel werden den Symbolleistenschaltflächen 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
    {
        //look in the site for potential server model
        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        = "dwscripts.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        = "dwscripts.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);
        }
        //add the rebuild proxy button for windows only.
        //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 sind die Technologien, mit denen Skripts auf einem Server ausgeführt werden. 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 dem Dokument hinzufügt.

Konfigurationsdateien für Servermodelle sind im Ordner „Configuration/ServerModels“ gespeichert. In diesem Ordner gibt es für jedes Servermodell jeweils eine HTML-Datei, die die erforderlichen Funktionen für das entsprechende Servermodell implementiert.

Anpassen von Servermodellen

Mithilfe der Funktionen der API für Servermodelle können Sie einige Einstellungen eines Servermodells anpassen.

Beim ersten Starten von Adobe Dreamweaver müssen Benutzer die zu verwendenden Servermodelle angeben. Für den Fall, dass ein Benutzer kein Servermodell festlegt, können Sie ein dynamisches Dialogfeld erstellen, in dem der Benutzer aufgefordert wird, die erforderlichen Schritte auszuführen. Dieses Dialogfeld wird angezeigt, wenn der Benutzer versucht, ein Serverobjekt einzufügen. Informationen zum Erstellen dieses Dialogfelds finden Sie in den Beschreibungen der Funktionen „getSetupSteps()“ auf Seite 334 und „setupStepsCompleted()“ auf Seite 336.

Sie können auch ein spezielles Servermodell erstellen. Adobe rät davon ab, die im Lieferumfang von Dreamweaver enthaltenen Servermodelle zu bearbeiten, und empfiehlt stattdessen, neue Servermodelle zu erstellen. (Informationen zum Erstellen neuer Dokumenttypen, die das von Ihnen verwendete Servermodell unterstützt, finden Sie unter „Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 14.)

Wenn Sie ein neues Servermodell erstellen, müssen Sie in die Servermodelldatei eine Implementierung der Funktion `canRecognizeDocument()` aufnehmen. Diese Funktion legt den Präferenzgrad Ihres Servermodells zum Verarbeiten eines Dateityps in Dreamweaver fest, wenn eine bestimmte Dateinamenerweiterung mehreren Servermodellen zugewiesen ist.

API-Funktionen für Servermodelle

In diesem Abschnitt werden die Funktionen beschrieben, mit denen Servermodelle für Dreamweaver konfiguriert werden.

canRecognizeDocument()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Beim Öffnen eines Dokuments, dessen 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. Wenn eine Dateinamenerweiterung mehreren Servermodellen zugewiesen ist, erhält das Servermodell Priorität, für das die Funktion die höchste Ganzzahl zurückgibt.

Hinweis: *Da alle für Dreamweaver definierten Servermodelle den Wert* `1` *zurückgeben, können Servermodelle von Drittanbietern die Zuweisung der Dateinamenerweiterungen außer Kraft setzen.*

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 folgenden 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 Standardservermodell von Dreamweaver.

```
var retVal = -1;
var langRE = /@\s*language\s*=\s*(\"|\')?javascript(\"|\')?/i;
// Search for the string language="javascript"
var oHTML = dom.documentElement.outerHTML;
if (oHTML.search(langRE) > -1)
    retVal = 2;
return retVal;
```

getFileExtensions()

Verfügbarkeit
Dreamweaver UltraDev 1, veraltet in Dreamweaver MX.

Beschreibung
Gibt die Dateinamenerweiterungen der Dokumentdateien zurück, die mit einem Servermodell verwendet werden können. So unterstützt das Servermodell ASP beispielsweise die Erweiterungen „.asp“ und „.htm“. Von dieser Funktion wird ein Array von Strings zurückgegeben. Dreamweaver füllt mit diesen Strings die Liste „Standard-Seitenerweiterung“ in der Kategorie „Anwendungsserver-Infos“ des Dialogfelds „Site-Definition“.

Hinweis: *Die Liste „Standard-Seitenerweiterung“ ist nur in Dreamweaver bis Version 4 verfügbar. In Dreamweaver MX und späteren Versionen werden im Dialogfeld „Site-Definition“ keine Einstellungen für die Dateinamenerweiterung angezeigt. Dreamweaver liest stattdessen die Datei „Extensions.txt“ und analysiert das Element in der Datei „mmDocumentTypes.xml“. (Weitere Informationen zu diesen beiden Dateien und dem Element* `documenttype` *finden Sie unter „**Erweiterbare Dokumenttypen in Dreamweaver“ auf Seite 14.**)*

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet ein Array von Strings mit den zulässigen Dateinamenerweiterungen.

getLanguageSignatures()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Diese Funktion gibt ein Objekt zurück, das die von der Skriptsprache verwendeten Methoden- und Array-Signaturen beschreibt. Die Funktion `getLanguageSignatures()` erleichtert die Zuordnung von generischen und sprachspezifischen Signaturen für die folgenden Elemente:

- Funktion
- Konstruktoren
- Ablagecode (Rückgabewerte)
- Arrays
- Ausnahmen
- Datentypzuordnungen für elementare 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 (z. B. eine Methode für Webdienste) durch Ziehen und Ablegen verschiebt.

Beispiele zum Programmieren dieser Funktion finden Sie in den HTML-Implementierungsdateien für die Servermodelle JSP und ASP.NET. Implementierungsdateien für Servermodelle befinden sich im Ordner „Configuration/ServerModels“.

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, veraltet in Dreamweaver MX.

Beschreibung
Diese Funktion gibt die Standarddateierweiterung von Dateien zurück, die das aktuelle Servermodell verwenden. Das Objekt `serverModel` wird auf das Servermodell der ausgewählten Site gesetzt, wenn derzeit kein Benutzerdokument ausgewählt ist.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String, der die unterstützten Dateinamenerweiterungen enthält.

getServerInfo()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Diese Funktion gibt ein JavaScript-Objekt zurück, auf das aus dem JavaScript-Code zugegriffen werden kann. Sie können dieses Objekt mit der JavaScript-Funktion `dom.serverModel.getServerInfo()` abrufen. `serverName`, `serverLanguage` und `serverVersion` sind besondere Eigenschaften, auf die Sie über folgende JavaScript-Funktionen zugreifen 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, veraltet in Dreamweaver MX.

Beschreibung
Diese Funktion gibt die unterstützten Skriptsprachen eines Servermodells in einem String-Array zurück. Dreamweaver füllt mit diesen Strings die Liste „Standard-Skriptsprache“ in der Kategorie „Anwendungsserver-Infos“ des Dialogfelds „Site-Definition“.

Hinweis: *Die Liste „Standard-Skriptsprache“ steht nur in Dreamweaver bis Version 4 zur Verfügung. In Dreamweaver MX und späteren Versionen werden im Dialogfeld „Site-Definition“ keine unterstützten Skriptsprachen angezeigt. Zudem wird die Funktion `getServerLanguages()` nicht mehr verwendet. Dies ist darauf zurückzuführen, 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 angewendet 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 im Menü mit dem Pluszeichen (+) des Bedienfelds „Serververhalten“ angezeigt wird, wenn die ausgewählte Skriptsprache `XXX` ist.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet ein Array von Strings, die die unterstützten Skriptsprachen enthalten.

getServerModelExtDataNameUD4()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Diese Funktion gibt den Namen der Servermodellimplementierung zurück, den Dreamweaver beim Zugreifen auf die UltraDev 4-Erweiterungsdateien im Ordner „Configuration/ExtensionData“ verwenden muss.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String, z. B. `"ASP/JavaScript"`.

getServerModelDelimiters()

Verfügbarkeit
Dreamweaver MX.

Beschreibung
Diese Funktion gibt die Skripttrennzeichen zurück, die vom Anwendungsserver verwendet werden. Zudem gibt sie an, ob die einzelnen Trennzeichen beim Zusammenführen von Codeblöcken 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 die Anfangstrennzeichenfolge des Skripts erkennt (z. B. `<%`).

- Die Eigenschaft *endPattern* ist ein regulärer Ausdruck, der die Abschlusstrennzeichenfolge des Skripts erkennt (z. B. `%>`).
- Die Eigenschaft *participateInMerge* ist ein boolescher Wert, der angibt, ob sich der zwischen die aufgeführten Trennzeichenfolgen platzierte Inhalt für das Zusammenführen von Codeblöcken eignet (`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 mit JavaScript über die Funktion `dom.serverModel.getDisplayName()` abrufen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String, 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()` abrufen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet einen String, 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 mit 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, veraltet in Dreamweaver MX.

Beschreibung
Diese Funktion ermittelt die Zuordnung von Servertechnologien zu bestimmten Versionsnummern. Diese Funktion wird von der Funktion `dom.serverModel.getServerVersion()` aufgerufen.

Argumente
Keine.

Rückgabewerte
Dreamweaver erwartet ein Array von Versionsobjekten, jedes davon mit einem Versionsnamen und einer Versionsnummer, wie in den folgenden Beispielen aufgeführt:

- `ASP Version 2.0`
- `ADODB Version 2.1`

Kapitel 22: Datenübersetzer

Datenübersetzer konvertieren spezielle Markups in Code, der in Adobe Dreamweaver gelesen und angezeigt werden kann. Beispiele für spezielle Markups sind serverbasierte Include-Dateien, konditionelle JavaScript-Anweisungen oder anderer Code wie PHP3, JSP, CFML oder ASP. In Dreamweaver können Sie Tag-Attribute sowie ganze Tags und Codeblöcke übersetzen lassen. Alle Datenübersetzer, ob Block/Tag- oder Attribut-Übersetzer, sind HTML-Dateien.

Das Übersetzen von Daten kann komplexe Operationen erfordern, die sich entweder mit JavaScript nicht durchführen lassen oder effizienter in C durchgeführt werden können. Dies gilt besonders bei ganzen Tags oder Codeblöcken. Wenn Sie mit C oder C++ vertraut sind, sollten Sie auch den Abschnitt „C-Level-Erweiterbarkeit“ auf Seite 369 lesen.

In der folgenden Tabelle sind die Dateien zum Erstellen von Datenübersetzern aufgeführt:

Pfad	Datei	Beschreibung
Configuration/ThirdPartyTags/	*Sprache*.xml	Enthält Informationen über Tags in der Markup-Sprache.
Configuration/ThirdPartyTags/	*Sprache*.gif	Symbol für Tags in der Sprache.
Configuration/Translators/	*Sprache*.htm	Enthält JavaScript-Funktionen für den Datenübersetzer.

Funktionsweise von Datenübersetzern

Dreamweaver verarbeitet alle Übersetzerdateien auf die gleiche Weise. Dabei spielt es keine Rolle, ob ganze Tags oder lediglich Attribute übersetzt werden. 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 beeinträchtigt wird, werden Fehler in Übersetzerdateien erst gemeldet, nachdem alle Übersetzer geladen wurden. Weitere Informationen zum Debuggen von Übersetzern finden Sie unter „Fehlersuche im Übersetzer“ auf Seite 356.*

Darüber hinaus ruft Dreamweaver in allen relevanten Übersetzerdateien (entsprechend der Angabe in den Übersetzungsvoreinstellungen) die Funktion `translateMarkup()` auf, wenn der Benutzer neuen Inhalt hinzufügt oder vorhandenen Inhalt ändert, der übersetzt werden muss. Dreamweaver ruft die Funktion `translateMarkup()` auf, wenn der Benutzer eine der folgenden Aktionen ausführt:

- Öffnen einer Datei in Dreamweaver
- Zurückwechseln in die Entwurfsansicht, nachdem im Bedienfeld „HTML“ oder in der Codeansicht Änderungen vorgenommen wurden
- Ändern der Eigenschaften eines Objekts im aktuellen Dokument
- Einfügen eines Objekts (über die Einfügeleiste oder das Menü „Einfügen“)
- Aktualisieren des aktuellen Dokuments, nachdem in einer anderen Anwendung daran Änderungen vorgenommen wurden
- Zuweisen einer Vorlage zum Dokument
- Einfügen eines Objekts in das Dokumentfenster oder Verschieben eines Objekts im Dokumentfenster
- Speichern von Änderungen in einer abhängigen Datei

- Aufrufen einer Befehls-, Verhaltens-, Serververhaltens-, Eigenschafteninspektor- oder sonstigen Erweiterung, mit der die Eigenschaft `innerHTML` oder `outerHTML` eines Tag-Objekts bzw. die `data`-Eigenschaft eines Kommentarobjekts festgelegt wird
- Auswählen von „Datei“ > „Konvertieren“ > „3.0 Browser-kompatibel“
- Auswählen von „Modifizieren“ > „Konvertieren“ > „Tabellen in AP-Divs konvertieren“
- Auswählen von „Modifizieren“ > „Konvertieren“ > „AP-Divs in Tabelle konvertieren“
- Ändern eines Tags oder Attributs im Quick Tag Editor und Drücken der Tabulator- bzw. Eingabetaste

Ermitteln des zu verwendenden Übersetzers

Alle Übersetzer müssen die Funktionen `getTranslatorInfo()` und `translateMarkup()` umfassen und 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 überprüft werden muss. Diese Unterschiede sind im Folgenden beschrieben.

- Für kleinere Server-Markup-Fragmente, die Attributwerte festlegen oder einem HTML-Standard-Tag bedingungsabhängig Attribute hinzufügen, programmieren Sie einen Attributübersetzer. HTML-Standard-Tags mit übersetzten Attributen können mit den Eigenschafteninspektoren überprüft werden, die Bestandteil von Dreamweaver sind. Es ist nicht nötig, einen benutzerdefinierten Eigenschafteninspektor zu programmieren (siehe „Aufnehmen übersetzter Attribute in ein Tag“ auf Seite 350).
- Zum Übersetzen ganzer Tags (z. B. eines Server-Side Includes (SSI)) oder eines Codeblocks (z. B. JavaScript, ColdFusion, PHP oder andere Skripts) programmieren Sie einen Block/Tag-Übersetzer. Der von einem Block/Tag-Übersetzer generierte Code kann nicht mit den integrierten Eigenschafteninspektoren von Dreamweaver überprüft werden. Sie müssen für den übersetzten Inhalt einen benutzerdefinierten Eigenschafteninspektor programmieren, wenn es Benutzern möglich sein soll, die Eigenschaften des Originalcodes zu ändern (siehe „Sperren übersetzter Tags und Codeblöcke“ auf Seite 352).

Aufnehmen übersetzter Attribute in ein Tag

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 Server-Markup mit anderen öffnenden und schließenden Markierungen verwenden, müssen Sie die Datenbank für Tags von Drittanbietern so anpassen, dass der Übersetzer korrekt funktioniert. Weitere Informationen zum Ändern der Datenbank für Tags von Drittanbietern 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 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 der zugehörige Wert sind im Bedienfeld „HTML“ oder in der Codeansicht nicht sichtbar und werden auch nicht mit dem Dokument gespeichert.

Das Attribut `mmTranslatedValue` darf nur einmal innerhalb eines Tags verwendet werden. Wenn die Wahrscheinlichkeit besteht, dass der Übersetzer für Tags mehr als je ein Attribut übersetzen muss, müssen Sie eine Routine in den Übersetzer aufnehmen, die Nummern an das `mmTranslatedValue`-Attribut anfügt (z. B. `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 für `<? if (dayType == weekday) then src="open.jpg" else src="closed.jpg" ?>` ist. Da `mmTranslatedValue="%22open.jpg%22"` nur den Wert, jedoch nicht das Attribut enthält, ist der Ausdruck in beiden Beispielen ungültig.

Gleichzeitiges Übersetzen mehrerer Attribute

Das Attribut `mmTranslatedValue` kann mehr als ein gültiges Attribut-Wert-Paar enthalten. Das lässt sich am folgenden nicht übersetzten Code veranschaulichen:

```
<img <? if (dayType==weekday) then src="open.jpg" width="320" height="100" else
src="closed.jpg" width="100" height="320" ?> alt="We're open 24 hours a day from
12:01am Monday until 11:59pm Friday">
```

Es folgt ein Beispiel für übersetzten Markup:

```
<img <? if (dayType==weekday) then src="open.jpg" width="320" height="100" else
src="closed.jpg" width="100" height="320" ?>
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 im `mmTranslatedValue`-Attribut sind nicht kodiert. Da Dreamweaver nach diesen Leerzeichen sucht, wenn der übersetzte Wert dargestellt werden soll, muss jedes Attribut-Wert-Paar im Attribut `mmTranslatedValue` separat kodiert und anschließend zurückgegeben werden, um die Vollständigkeit von `mmTranslatedValue` zu gewährleisten. Ein Beispiel für diesen Vorgang finden Sie unter „Einfaches Beispiel für einen Attributübersetzer“ auf Seite 357.

Überprüfen übersetzter Attribute

Für übersetzte Attribute sind keine Sperren erforderlich. Dadurch wird das Überprüfen von Tags mit diesen Attributen zu einer einfachen Angelegenheit.

Wenn ein Server-Markup ein einzelnes Attribut festlegt, das in einem Eigenschafteninspektor dargestellt wird, zeigt Dreamweaver das Attribut im Eigenschafteninspektor an.

Das Markup wird in jedem Fall angezeigt, auch wenn ihm kein Übersetzer zugeordnet ist. Der Übersetzer wird ausgeführt, wenn der Benutzer das im Eigenschafteninspektor angezeigte Markup bearbeitet.

Wenn ein Server-Markup mehrere Attribute in einem Tag steuert, wird es nicht im Eigenschafteninspektor angezeigt. Durch das Blitzsymbol wird jedoch angegeben, dass für das ausgewählte Element ein übersetztes Markup vorhanden ist.

Hinweis: *Das Blitzsymbol wird nicht angezeigt, wenn Text, Tabellenzellen, Zeilen oder Spalten ausgewählt sind. Die Übersetzung wird fortgesetzt, wenn der Benutzer das Server-Markup im Bedienfeld bearbeitet und ein für den entsprechenden Markup-Typ geeigneter Übersetzer vorhanden ist.*

Die Textfelder im Eigenschafteninspektor sind bearbeitbar. Benutzer können Werte für Attribute eingeben, die vom Server-Markup gesteuert werden. Dadurch kann es zu Duplikaten von Attributen kommen. Wenn für ein bestimmtes Attribut sowohl ein übersetzter als auch ein regulärer Wert festgelegt ist, zeigt Dreamweaver im Dokumentfenster den übersetzten Wert an, stellt fest, ob der Übersetzer nach Duplikaten von Attributen sucht und entfernt diese.

Sperren übersetzter Tags und Codeblöcke

In den meisten Fällen soll ein Übersetzer das Markup ändern, sodass Dreamweaver es anzeigen kann. Dennoch soll das Original-Markup – und nicht die vorgenommenen Änderungen – gespeichert werden. Zu diesem Zweck stellt Dreamweaver spezielle XML-Tags zur Verfügung, in die der übersetzte Inhalt eingeschlossen wird 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, nicht übersetzte Markup in die Datei geschrieben. Dreamweaver zeigt in der Codeansicht den nicht übersetzten Inhalt an.

Es folgt ein Beispiel für die Syntax der XML-Tags:

```
<MM:BeginLock translatorClass="translatorClass" ¬
type="tagNameOrType" depFiles="dependentFilesList" ¬
orig="encodedOriginalMarkup">
Translated content
<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 der Funktion `getTranslatorInfo()` zurückgegeben wird.
- Der Wert *tagNameOrType* ist ein String, der den Markup-Typ innerhalb des gesperrten Bereichs angibt (oder den Tag-Namen, der dem Markup zugeordnet ist). Der String darf nur alphanumerische Zeichen, Bindestriche (-) oder Unterstriche (_) enthalten. Sie können diesen Wert in der Funktion `canInspectSelection()` eines benutzerdefinierten Eigenschafteninspektors überprüfen, um festzustellen, ob sich der Eigenschafteninspektor für den Inhalt eignet. Weitere Informationen finden Sie unter „Erstellen von Eigenschafteninspektoren für gesperrte Inhalte“ auf Seite 353. Gesperrter Inhalt kann nicht von den in Dreamweaver integrierten Eigenschafteninspektoren überprüft werden. So ist es beispielsweise nicht möglich, durch Festlegen von `type="IMG"` das Bedienfeld „Bild“ aufzurufen.
- Der Wert *dependentFilesList* ist ein String mit einer durch Kommas getrennten Liste von Dateien, von denen das gesperrte Markup abhängt. Dateien werden als URLs und relativ zum Dokument des Benutzers referenziert. Wenn der Benutzer eine der Dateien im String *dependentFilesList* aktualisiert, übersetzt Dreamweaver automatisch erneut den Inhalt in dem Dokument, das die Liste enthält.
- Der Wert von *encodedOriginalMarkup* ist ein String mit dem nicht übersetzten Original-Markup, das unter Verwendung einer kleinen·Teilmenge der URL-Kodierung kodiert·ist (`%22` steht für `"`, `%3C` für `<`, `%3E` für `>` und `%25` für `%`). Die schnellste Möglichkeit zur URL-Kodierung eines Strings bietet die Methode `escape()`. Wenn beispielsweise `myString` den Wert `'<img src="foo.gif">'` hat, gibt `escape(myString)` den Wert `%3Cimg%20src=%22foo.gif%22%3E` zurück.

Im folgenden Beispiel ist der gesperrte Codebereich dargestellt, der bei der Übersetzung des Server-Side Include `<!--#include virtual="/footer.html" -->` ggf. generiert wird:

```
<MM:BeginLock translatorClass="MM_SSI" type="ssi" ¬
depFiles="C:\sites\webdev\footer.html" orig="%3C!--#include ¬
virtual=%22/footer.html%22%20--%3E">
<!-- begin footer -->
<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>
<!-- end footer -->
<MM:EndLock>
```

Das Erstellen eines Übersetzers, der Code innerhalb eines `script`-Tags einschließt, kann zum Fehlschlagen des Übersetzers führen. Angenommen, es liegt folgender Code vor:

```
<script language="javascript">
<!--
function foo() {
alert('<bean:message key="show.message"/>');
}
// -->
</script>
```

Anschließend erstellen Sie einen Übersetzer für das `bean:message`-Tag. Das Ausführen des Übersetzers führt zu einem Fehler, da Sie einen `MM:BeginLock`-Abschnitt innerhalb eines `MM:BeginLock`-Abschnitts erstellen. Sie können dies umgehen, indem Sie das `bean:message`-Tag in einen JSP-Wrapper einschließen, der reguläre JSP-Tags wie `<%= My_lookup.lookup("show.message") %>` verwendet. Dies bewirkt, dass der Übersetzer diesen Code überspringt, und die Übersetzung kann erfolgreich abgeschlossen werden.

Erstellen von Eigenschafteninspektoren für gesperrte Inhalte

Nachdem Sie einen Übersetzer erstellt haben, müssen Sie einen Eigenschafteninspektor für den Inhalt erstellen, damit der Benutzer dessen Eigenschaften ändern kann (z. B. die einzuschließende Datei oder eine der Bedingungen in einer bedingten Anweisung). Das Überprüfen von übersetztem Inhalt stellt aus mehreren Gründen ein sehr spezielles Problem dar:

- Der Benutzer möchte möglicherweise die Eigenschaften des übersetzten Inhalts ändern. Diese Änderungen müssen im nicht übersetzten Inhalt umgesetzt werden.
- Das Dokumentobjektmodell (DOM) enthält den übersetzten Inhalt (d. h., die `lock`-Tags und die von ihnen eingeschlossenen·Tags sind Knoten im DOM), die Eigenschaft `outerHTML` des Objekts `documentElement`und die Funktionen `dreamweaver.getSelection()` und `dreamweaver.nodeToOffsets()` greifen jedoch auf die nicht übersetzten Quelldaten zu.
- Die überprüften Tags sind vor und nach dem Übersetzen verschieden.

Ein Eigenschafteninspektor für das Tag `HAPPY` kann einen Kommentar ähnlich dem folgenden Beispielcode beinhalten:

```
<!-- tag:HAPPY,priority:5,selection:exact,hline,vline, attrName:xxx,¬ attrValue:yyy -->
```

Der Eigenschafteninspektor für das übersetzte `HAPPY`-Tag hätte jedoch einen Kommentar ähnlich dem folgenden Beispiel:

```
<!-- tag:*LOCKED*,priority:5,selection:within,hline,vline -->
```

Die Funktion `canInspectSelection()` für den Eigenschafteninspektor des nicht übersetzten `HAPPY`-Tags 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 jedoch 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 Überprüfen von übersetzten Inhalten. Wenn Sie die Funktion `dom.getSelection()` aufrufen, sind die standardmäßig zurückgegebenen Werte Offsets in die nicht übersetzte Quelle. Um die Auswahl korrekt zu erweitern, sodass der gesperrte Bereich (und nur dieser) ausgewählt wird, verwenden Sie folgende Technik:

```
var currentDOM = dw.getDocumentDOM();
var offsets = currentDOM.getSelection();
var theSelection = currentDOM.offsetsToNode(offsets[0],offsets[0]+1);
```

Wenn Sie als zweites Argument `offsets[0]+1` verwenden, ist sichergestellt, dass Sie innerhalb des öffnenden Sperr-Tags 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 wie im folgenden Beispiel das `type`-Attribut überprüfen, um zu ermitteln, 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 zu füllen, müssen Sie den Wert des Attributs `orig` analysieren. Wenn der nicht übersetzte 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(curSelection[0],curSelection[0]+1);

    if (theObj.nodeType != Node.ELEMENT_NODE) {
        return;
    }

    // To convert the encoded characters back to their
    // original values, use the unescape() method.
    var origAtt = unescape(theObj.getAttribute("ORIG"));

    // Convert the string to lowercase for processing
    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;
}
```

Nachdem Sie das Attribut `orig` analysiert haben, um die Felder des Eigenschafteninspektors für das übersetzte Tag zu füllen, empfiehlt es sich, den Wert für das Attribut `orig` für den Fall festzulegen, dass der Benutzer den Wert in einem der Textfelder ändert. Möglicherweise gelten jedoch Einschränkungen für Änderungen in einem gesperrten Bereich. Dieses Problem lässt sich umgehen, indem Sie das Original-Markup ändern und neu übersetzen.

Für den Eigenschafteninspektor für übersetzte Server-Side Includes (die Datei „ssi_translated.js“ im Ordner „Configuration/Inspectors“) wird diese Vorgehensweise in der Funktion `setComment()` demonstriert. Statt das Attribut `orig` umzuschreiben, stellt der Eigenschafteninspektor einen neuen SSI-Kommentar zusammen. Anschließend wird dieser 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 Vorgehensweise:

```
// Assemble the new include comment. radioStr and URL are
// variables defined earlier in the code.
newInc = "<!--#include " + radioStr + "=" + '"' + URL + '"' + " -->";
// Get the contents of the document.
var entireDocObj = dreamweaver.getDocumentDOM();
var docSrc = entireDocObj.documentElement.outerHTML;
// Store everything up to the SSI comment and everything after
// the SSI comment in the beforeSelStr and afterSelStr variables.
var beforeSelStr = docSrc.substring(0, curSelection[0] );
var afterSelStr= docSrc.substring(curSelection[1]);
// Assemble the new contents of the document.
docSrc = beforeSelStr + newInc + afterSelStr;
// Set the outerHTML of the HTML tag (represented by
// the documentElement object) to the new contents,
// and then set the selection back to the locked region
// surrounding the SSI comment.
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, schlägt beim Aufrufen jedoch ohne eine Fehlermeldung fehl. Durch den Verzicht auf Fehlermeldungen wird zwar die Stabilität von Dreamweaver erhöht, jedoch die Codeentwicklung erschwert, besonders wenn ein kleiner Syntaxfehler in mehreren Codezeilen gesucht werden muss.

Wenn Ihr Übersetzer nicht ausgeführt werden kann, besteht ein effektives Fehlersuchverfahren darin, ihn in einen Befehl umzuwandeln. 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. Kommentieren Sie am Ende der Funktion `translateMarkup()` die Zeile `return`*`jeweiliger_Rückgabewert`* aus und ersetzen Sie sie durch `dreamweaver.getDocumentDOM().documentElement.outerHTML =` *`jeweiliger_Rückgabewert`*, wie im folgenden Beispiel dargestellt:

```
    // return theCode;
    dreamweaver.getDocumentDOM().documentElement.outerHTML = theCode;
}
/* end of translateMarkup() */
```

4. Fügen Sie im body-Bereich des Dokuments das folgende Formular ohne Textfelder 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.

   Wenn keine Fehlermeldung angezeigt wird und die Übersetzung weiterhin fehlschlägt, liegt vermutlich ein logischer Fehler im Code vor.

6. Fügen 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);
    /* rest of loop */
}
```

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 Beispiel für einen Attributübersetzer

Ein Beispiel erleichtert das Verständnis der Attributübersetzung. Der folgende Übersetzer ist für das Markup „Pound Conditional“ (Poco) bestimmt. Dabei handelt es sich um eine mit ASP und PHP verwandte Syntax.

Sie erstellen den Attributübersetzer, indem Sie ein `tagspec`-Tag, ein Symbol und einen Attributübersetzer erstellen.

Erstellen des tagspec-Tags

Beim Erstellen des Übersetzers legen Sie zunächst ein `tagspec`-Tag für Poco-Markup an. Dadurch wird die Analyse nicht übersetzter Poco-Anweisungen durch Dreamweaver verhindert.

1. Erstellen Sie eine neue, leere Datei.
2. Geben Sie Folgendes ein:

```
<tagspec tag_name="poco" start_string="<#" end_string="#>" ¬
detect_in_attribute="true" icon="poco.gif" icon_width="17" ¬
icon_height="15"></tagspec>
```

3. Speichern Sie die Datei unter dem Namen „poco.xml“ im Ordner „Configuration/ThirdPartyTags“.

Ein Beispiel für das Erstellen des `tagspec`-Tags finden Sie in der Datei „Tags.xml“ im Ordner „Configuration/ThirdPartyTags“.

Erstellen des Symbols

Als Nächstes erstellen Sie das Symbol für Poco-Tags.

1. Erstellen Sie eine Bilddatei 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“.

Erstellen des Attributübersetzers

Sie erstellen eine HTML-Datei mit den für den Attributübersetzer erforderlichen Funktionen.

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">

/*************************************************************
 * This translator handles the following statement syntaxes: *
 * <# 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" #>           *
 *                                                           *
 * It does not handle statements with no else clause.        *
 *************************************************************/

var count = 1;

function translateMarkup(docNameStr, siteRootStr, inStr){
var count = 1;
// Counter to ensure unique mmTranslatedValues
var outStr = inStr;
// String that will be manipulated
var spacer = "";
// String to manage space between encoded attributes
var start = inStr.indexOf('<# if');   // 1st instance of Pound Conditional code
// Declared but not initalized //
var attAndValue;
// Boolean indicating whether the attribute is part of
// the conditional statement
var trueStart;
// The beginning of the true case
var falseStart;
// The beginning of the false case
var trueValue;
// The HTML that would render in the true case
var attName;
// The name of the attribute that is being
// set conditionally.
var equalSign;
// The position of the equal sign just to the
// left of the <#, if there is one
var transAtt;
// The entire translated attribute
var transValue;
// The value that must be URL-encoded
var back3FromStart;
// Three characters back from the start position
// (used to find equal sign to the left of <#
var tokens;
// An array of all the attributes set in the true case
var end;
// The end of the current conditional statement.
// As long as there's still a <# conditional that hasn't been
// translated.
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,' ');

// If attAndValue is false, find out what attribute you're
// translating by backing up from the equal sign to the
// first space. The substring between the space and the
// equal sign is the attribute.
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);
// If attAndValue is true, and tokens is greater than
// 1, then trueValue is a series of attribute/value
// pairs, not just one. In that case, each attribute/value
// pair must be encoded separately and then added back
// together to make the translated value.
}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)

// If attAndValue is true and tokens is not greater
// than 1, then trueValue is a single attribute/value pair.
// This is the simplest case, where all that is necessary is
// to encode trueValue.
}else{
    transValue = trueValue;
    transAtt = ' mmTranslatedValue' + count + '="' + ¬
    escape(transValue) + '"';
    outStr = outStr.substring(0,end+3) + transAtt + ¬
    outStr.substring(end+3);
}
// Increment the counter so that the next instance
// of mmTranslatedValue will have a unique name, and
// then find the next <# conditional in the code.
count++;
```

```
    start = outStr.indexOf('<# if',end);
}
// Return the translated string.
return outStr
}
function getTranslatorInfo(){
    returnArray = new Array(7);

    returnArray[0] = "Pound_Conditional";               // The translatorClass
    returnArray[1] = "Pound Conditional Translator";    // The title
    returnArray[2] = "2";                               // The number of extensions
    returnArray[3] = "html";                            // The first extension
    returnArray[4] = "htm";                             // The second extension
    returnArray[5] = "1";                               // The number of expressions
    returnArray[6] = "<#";                              // The first expression
    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 verdeutlicht werden, der ausschließlich in JavaScript programmiert ist und bei dem keine C-Bibliothek für Funktionen herangezogen wird. Der folgende Übersetzer wäre in C effizienter, die JavaScript-Version ist jedoch einfacher und eignet sich daher besser für Demonstrationszwecke.

Wie die meisten Übersetzer dient auch dieser dazu, Serververhalten zu imitieren. Im Beispiel wird davon ausgegangen, dass der Webserver so konfiguriert ist, dass das `KENT`-Tag durch das Bild eines jeweils anderen Ingenieurs ersetzt wird – je nach Wochentag, Uhrzeit und Plattform des Benutzers. Der Übersetzer führt die gleiche Aktion aus, allerdings lokal.

Erstellen des Block/Tag-Übersetzers

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">
/*************************************************************
 * The getTranslatorInfo() function provides information  *
 * about the translator, including its class and name,    *
 * the types of documents that are likely to contain the  *
 * markup to be translated, the regular expressions that  *
 * a document containing the markup to be translated      *
 * would match (whether the translator should run on all  *
 * files, no files, in files with the specified           *
 * extensions, or in files matching the specified         *
 * expressions).                                          *
 *************************************************************/
function getTranslatorInfo(){
    //Create a new array with 6 slots in it
    returnArray = new Array(6);

    returnArray[0] = "DREAMWEAVER_TEAM";           // The translatorClass
    returnArray[1] = "Kent Tags";                  // The title
    returnArray[2] = "0";                          // The number of extensions
    returnArray[3] = "1";                          // The number of expressions
    returnArray[4] = "<kent";                      // Expression
    returnArray[5] = "byExpression";               // run if the file contains "<kent"
    return returnArray;
}

/*************************************************************************
 * The translateMarkup() function performs the actual translation.        *
 * In this translator, the translateMarkup() function is written          *
 * entirely in JavaScript (that is, it does not rely on a C library) --   *
 * and it's also extremely inefficient. It's a simple example, however,   *
 * which is good for learning.                                            *
*************************************************************************/
function translateMarkup(docNameStr, siteRootStr, inStr){
    var outStr = "";                       // The string to be returned after translation
    var start = inStr.indexOf('<kent>');  // The first position of the KENT tag
                                           // in the document.
    var replCode = replaceKentTag();       // Calls the replaceKentTag() function
                                           // to get the code that will replace KENT.
        var outStr = "";                   // The string to be returned after translation
     //If the document does not contain any content, terminate the translation.
    if ( inStr.length <= 0 ){
         return "";
    }

    // As long as start, which is equal to the location in inStr of the
    // KENT tag, is not equal to -1 (that is, as long as there is another
    // KENT tag in the document)
    while (start != -1){
         // Copy everything up to the start of the KENT tag.
         // This is very important, as translators should never change
         // anything other than the markup that is to be translated.
         outStr = inStr.substring(0, start);
         // Replace the KENT tag with the translated HTML, wrapped in special
```

```
        // locking tags. For more information on the repacement operation, see
        // the comments in the replaceKentTag() function.
        outStr = outStr + replCode;

        // Copy everything after the KENT tag.
        outStr = outStr + inStr.substring(start+6);

        // Use the string you just created for the next trip through
        // the document. This is the most inefficient part of all.
        inStr = outStr;
        start = inStr.indexOf('<kent>');

    }
    // When there are no more KENT tags in the document, return outStr.
    return outStr;
}

/************************************************************
* The replaceKentTag() function assembles the HTML that will  *
* replace the KENT tag and the special locking tags that will *
* surround the HTML. It calls the getImage() function to      *
* determine the SRC of the IMG tag.                           *
************************************************************/
function replaceKentTag(){
    // The image to display.
    var image = getImage();
    // The location of the image on the local disk.
    var depFiles = dreamweaver.getSiteRoot() + image;
    // The IMG tag that will be inserted between the lock tags.
    var imgTag = '<IMG SRC="/' + image + '" WIDTH="320" HEIGHT="240" ALT="Kent">\n';
    // 1st part of the opening lock tag. The remainder of the tag is assembled
    below.
    var start = '<MM:BeginLock translatorClass="DREAMWEAVER_TEAM" type="kent"';
    // The closing lock tag.
    var end = '<MM:EndLock>';

    //Assemble the lock tags and the replacement HTML.
    var replCode = start + ' depFiles="' + depFiles + '"';
    replCode = replCode + ' orig="%3Ckent%3E">\n';
    replCode = replCode + imgTag;
    replCode = replCode + end;

    return replCode;
}

/*************************************************************
 * The getImage() function determines which image to display      *
 * based on the day of the week, the time of day and the          *
 * user's platform. The day and time are figured based on UTC     *
 * time (Greenwich Mean Time) minus 8 hours, which gives          *
 * Pacific Standard Time (PST). No allowance is made for Daylight *
 * Savings Time in this routine.                                  *
*************************************************************/
function getImage(){
    var today = new Date();                 // Today's date & time.
    var day = today.getUTCDay();            // Day of the week in the GMT time zone.
                                            // 0=Sunday, 1=Monday, and so on.
```

```
    var hour = today.getUTCHours();     // The current hour in GMT, based on the
                                        // 24-hour clock.
var SFhour = hour - 8;                  // The time in San Francisco, based on the
                                        // 24-hour clock.
var platform = navigator.platform;      // User's platform. All Windows computers
                                        // are identified by Dreamweaver as "Win32",
                                        // all Macs as "MacPPC".
    var imageRef;                       // The image reference to be returned.
        // If SFhour is negative, you have two adjustments to make.
        // First, subtract one from the day count because it is already the wee
        // hours of the next day in GMT. Second, add SFhour to 24 to
        // give a valid hour in the 24-hour clock.
 if (SFhour < 0){
     day = day - 1;
         // The day count back one would make it negative, and it's Saturday,
         // so set the count to 6.
         if (day < 0){
                 day = 6;
         }
     SFhour = SFhour + 24;
 }

// Now determine which photo to show based on whether it's a work day or a
// weekend; what time it is; and, if it's a time and day when Kent is
// working, what platform the user is on.

//If it's not Sunday
if (day != 0){
     //And it's between 10am and noon, inclusive
     if (SFhour >= 10 && SFhour <= 12){
            imageRef = "images/kent_tiredAndIrritated.jpg";
        //Or else it's between 1pm and 3pm, inclusive
     }else if (SFhour >= 13 && SFhour <= 15){
            imageRef ="images/kent_hungry.jpg";
     //Or else it's between 4pm and 5pm, inclusive
     }else if (SFhour >= 16 && SFhour <= 17){
            //If user is on Mac, show Kent working on Mac
            if (platform == "MacPPC"){
                 imageRef = "images/kent_gettingStartedOnMac.jpg";
            //If user is on Win, show Kent working on Win
            }else{
                 imageRef = "images/kent_gettingStartedOnWin.jpg";
            }
     //Or else it's after 6pm but before the stroke of midnight
     }else if (SFhour >= 18){
             //If it's Saturday
             if (day == 6){
                 imageRef = "images/kent_dancing.jpg";
            //If it's not Saturday, check the user's platform
            }else if (platform == "MacPPC"){
                 imageRef = "images/kent_hardAtWorkOnMac.jpg";
```

```
                }else{
                     imageRef = "images/kent_hardAtWorkOnWin.jpg";
                }
           }else{
                imageRef = "images/kent_sleeping.jpg";
           }
     //If it's after midnight and before 10am, or anytime on Sunday
     }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“.

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 eindeutig den Übersetzer. Dieser String muss mit einem Buchstaben beginnen und darf nur alphanumerische Zeichen, Bindestriche (-) und Unterstriche (_) enthalten.

2 Der String *title* beschreibt den Übersetzer mit maximal 40 Zeichen.

3 Der String *nExtensions* gibt die Anzahl der nachfolgenden Dateinamenerweiterungen an. Wenn *nExtensions* gleich 0 ist, kann der Übersetzer auf jede Datei angewendet werden. Wenn *nExtensions* gleich 0 ist, stellt *nRegExps* das nächste Element des Arrays dar.

4 Der String *extension* gibt eine Erweiterung für Dateien an (z. B. `"htm"` oder `"SHTML"`), auf die der 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, die in *nExtensions* angegeben sind.

5 Der String *nRegExps* gibt die Anzahl der nachfolgenden regulären Ausdrücke an. Wenn *nRegExps* gleich 0 ist, ist *runDefault* das nächste Element im Array.

6 Der String *regExps* gibt einen regulären Ausdruck an, den Sie überprüfen können. Das Array muss die gleiche Anzahl von *regExps*-Elementen enthalten, die in *nRegExps* angegeben sind. Mindestens ein *regExps*-Element muss einem Bereich im Quellcode des Dokuments entsprechen, damit der Übersetzer eine Datei bearbeiten kann.

7 Der String *runDefault* legt den Ausführungszeitpunkt des Übersetzers fest. Im Folgenden 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 Dateinamenerweiterungen 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 angegebenen Strings enthält.

Hinweis: *Wenn Sie `runDefault` auf `"byExtension"` setzen, jedoch keine Erweiterungen angeben (siehe Schritt 4), ist das Ergebnis dasselbe wie bei der Einstellung `"allFiles"`. Wenn Sie `runDefault` auf `"byExpression"` setzen, jedoch keine regulären Ausdrücke angeben (siehe Schritt 6), ist das Ergebnis dasselbe wie bei der Einstellung `"noFiles"`.*

8 Der String *priority* gibt die Standardpriorität zum Ausführen des Übersetzers an. Die Priorität ist ein Wert zwischen `0` und `100`. Wenn Sie keine Priorität angeben, gilt der Standardwert `100`. Die höchste Priorität ist `0` und die niedrigste `100`. Wenn auf ein Dokument mehrere Übersetzer angewendet werden, legt diese Einstellung die Reihenfolge der Übersetzer fest. Der Übersetzer mit der höchsten Priorität wird zuerst angewendet. Haben mehrere Übersetzer dieselbe Priorität, werden sie von `translatorClass` in alphabetischer Reihenfolge angewendet.

Beispiel

Die folgende Instanz der Funktion `getTranslatorInfo()` liefert Informationen über einen Übersetzer für SSI:

```
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 Übersetzungsdurchgänge aus. Beim ersten Durchgang werden alle Übersetzer durchlaufen und die jeweilige Funktion `translateMarkup()` aufgerufen. Nachdem diese Funktionen aufgerufen wurden, werden im zweiten Übersetzungsdurchgang die `translateDOM()`-Funktionen aufgerufen. Das übergebene `dom` ist das zu übersetzende `dom`. Die einzigen Bearbeitungen, die während des zweiten Durchgangs zulässig sind, betreffen übersetzte Attribute.

Argumente
dom, sourceStr

- Das *dom*-Argument.
- Das *sourceStr*-Argument ist derselbe String, der an `translateMarkup` übergeben wird. Er wird zu Referenzzwecken bereitgestellt, alle Übersetzungen sollten jedoch mit dem *dom*-Argument durchgeführt werden und nicht mit dem *sourceStr*-Argument.

Rückgabewerte
Dreamweaver erwartet keine Rückgabewerte.

Beispiel

```
translateDOM( dom, sourceStr ); //kein Rückgabewert
```

Die folgende Instanz der Funktion `translateDOM()` blendet das Tag mit der ID `div1` im Dokument aus.

```
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*

- Das Argument *docName* ist ein String mit der URL für das zu übersetzende Dokument im Format „file://“.
- Das Argument *siteRoot* ist ein String mit der „file://“-URL für den Stamm der Site, die das zu übersetzende Dokument enthält. Wenn sich das Dokument außerhalb einer Site befindet, 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
Die folgende Instanz für die Funktion `translateMarkup()` ruft die C-Funktion `translateASP()` auf, die sich unter Windows in einer dynamischen Bibliothek (DLL) bzw. auf dem Macintosh in einer Codebibliothek mit dem Namen „ASPTrans“ befindet.

```
function translateMarkup(docName, siteRoot, docContent){
    var translatedString = "";
    if (docContent.length > 0){
    translatedString = ASPTrans.translateASP(docName, siteRoot, ¬
    docContent);
    }
    return translatedString;
}
```

Ein vollständig in JavaScript implementiertes Beispiel finden Sie unter „Einfaches Beispiel für einen Attributübersetzer“ auf Seite 357 oder „Einfaches Beispiel für einen Block/Tag-Übersetzer“ auf Seite 360.

liveDataTranslateMarkup()

Verfügbarkeit
Dreamweaver UltraDev 1.

Beschreibung
Diese Funktion übersetzt Dokumente, wenn Benutzer das Fenster „Live Data“ verwenden. Wenn der Benutzer den Befehl „Ansicht“ > „Live Data“ auswählt oder auf die Schaltfläche „Aktualisieren“ klickt, ruft Dreamweaver die Funktion `liveDataTranslateMarkup()` (anstelle von `translateMarkup()`) auf.

Argumente
docName, *siteRoot*, *docContent*

- Das Argument *docName* ist ein String mit der URL für das zu übersetzende Dokument im Format „file://“.
- Das Argument *siteRoot* ist ein String mit der „file://“-URL für den Stamm der Site, die das zu übersetzende Dokument enthält. Wenn sich das Dokument außerhalb einer Site befindet, 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

Die folgende Instanz der Funktion `liveDataTranslateMarkup()` ruft die C-Funktion `translateASP()` auf, die sich unter Windows in einer dynamischen Bibliothek (DLL) bzw. auf dem Macintosh in einer Codebibliothek mit dem Namen „ASPTrans“ befindet.

```
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 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 Ordner „Configuration/JSExtensions“ des Dreamweaver-Anwendungsordners und rufen dann die JavaScript-Funktionen über den integrierten JavaScript-Interpreter von Dreamweaver auf.

So können 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 programmieren.

Integration von C-Funktionen

Sie können 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>
Enter the name of the file to be inserted:
<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 der Funktion `readContentsOfFile()` finden Sie unter „C-Level-Erweiterbarkeit und JavaScript-Interpreter“ auf Seite 371.

```
JSBool
readContentsOfFile(JSContext *cx, JSObject *obj, unsigned int ¬
argc, jsval *argv, jsval *rval)
{
    char *fileName, *fileContents;
    JSBool success;
    unsigned int length;

    /* Make sure caller passed in exactly one argument. If not,
     * then tell the interpreter to abort script execution.*/
    if (argc != 1){
        JS_ReportError(cx, "Wrong number of arguments", 0);
    return JS_FALSE;
    }

    /* Convert the argument to a string */
    fileName = JS_ValueToString(cx, argv[0], &length);
    if (fileName == NULL){
        JS_ReportError(cx, "The argument must be a string", 0);
        return JS_FALSE;
    }

    /* Use the string (the file name) to open and read a file */
    fileContents = exerciseLeftToTheReader(fileName);

    /* Store file contents in rval, which is the return value passed
    * back to the caller */
    success = JS_StringToValue(cx, fileContents, 0, *rval);
    free(fileContents);

    /* Return true to continue or false to abort the script */
    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, um drei Datenelemente abzurufen:

- JavaScript-Name der Funktion
- Zeiger auf die Funktion
- Anzahl der Argumente, die die Funktion erwartet

Es folgt ein Beispiel für die Funktion `MM_Init()` für die Bibliothek `myLibrary`.

```
void
MM_Init()
{
    JS_DefineFunction("readContentsOfFile", readContentsOfFile, 1);
}
```

Ihre Bibliothek muss genau eine Instanz des folgenden Makros enthalten:

```
/* MM_STATE is a macro that expands to some definitions that are
 * needed to interact with Dreamweaver. This macro must
 * be defined exactly once in your library. */
MM_STATE
```

Hinweis: *Die Bibliothek kann entweder in C oder C++ implementiert sein. Die Datei, die die Funktion* `MM_Init()` *und das Makro* `MM_STATE` *enthält, muss jedoch in C implementiert sein. Der C++-Compiler verändert Funktionsnamen, sodass Dreamweaver die Funktion* `MM_Init()` *nicht finden kann.*

C-Level-Erweiterbarkeit und JavaScript-Interpreter

Es gibt drei Fälle, in 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 übergebenen 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 am Anfang jeder Datei in der Bibliothek die Datei „mm_jsapi.h“ mit der folgenden Zeile einfügen:

```
#include "mm_jsapi.h"
```

Zusammen mit der Datei „mm_jsapi.h“ wird die Datei „mm_jsapi_environment.h“ eingefügt, die die Struktur `MM_Environment` definiert.

Datentypen

Der JavaScript-Interpreter definiert die folgenden Datentypen.

typedef struct JSContext JSContext
An die C-Level-Funktion wird ein Zeiger auf diesen unspezifischen Datentyp übergeben. Einige API-Funktionen akzeptieren diesen Zeiger als Argument.

typedef struct JSObject JSObject
An die C-Level-Funktion wird ein Zeiger auf diesen unspezifischen Datentyp übergeben. Dieser Datentyp steht für ein Objekt, z. B ein Array-Objekt oder ein anderer Objekttyp.

typedef struct JSVal JSVal
Eine unspezifische Datenstruktur, die eine Ganzzahl, einen Zeiger auf eine Gleitkommazahl, einen String oder ein Objekt enthalten kann. Einige API-Funktionen können die Werte von Funktionsargumenten einlesen, indem der Inhalt der JSVal-Struktur gelesen wird. Andere API-Funktionen dienen zum Schreiben des Funktionsrückgabewerts, indem eine JSVal-Struktur geschrieben wird.

typedef enum { JS_FALSE = 0, JS_TRUE = 1 } JSBool
Ein einfacher Datentyp, der einen booleschen Wert enthält.

C-Level-API

Die API für die C-Level-Erweiterbarkeit 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:

- Der Zeiger *cx* ist ein Zeiger auf eine unspezifische `JSContext`-Struktur, der an einige Funktionen der JavaScript-API übergeben werden muss. Diese Variable enthält den Ausführungskontext des Interpreters.
- Der Zeiger *obj* ist ein Zeiger auf das Objekt, in dessen Kontext das Skript ausgeführt wird. Während der Skriptausführung entspricht das Schlüsselwort `this` diesem Objekt.
- Die Ganzzahl *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.
- Der Zeiger *rval* ist ein Zeiger auf eine einzelne `JSVal`-Struktur. Der Rückgabewert der Funktion muss in `*rval` geschrieben werden.

Bei einem erfolgreichen Abschluss 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-Level-Funktion registriert hat, die Sie im *call*-Argument angeben, können Sie sie in einem JavaScript-Skript aufrufen, indem Sie mit dem im *name*-Argument angegebenen Namen darauf verweisen. Bei dem Argument *name* ist die Groß- und Kleinschreibung zu beachten.

In der Regel wird diese Funktion über die Funktion `MM_Init()` aufgerufen, die Dreamweaver beim Starten aktiviert.

Argumente

`char *`*name*, `JSNative`*call*, `unsigned int nargs`

- Das Argument *name* ist der Name der Funktion, der in JavaScript verwendet wird.
- Das Argument *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.
- Das Argument *nargs* ist die von der Funktion erwartete Zahl von Argumenten.

Rückgabewerte

Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

char *JS_ValueToString()

Beschreibung
Diese Funktion extrahiert ein Funktionsargument aus einer JSVal-Struktur, konvertiert es nach Möglichkeit in einen String und gibt den konvertierten Wert zurück.

Hinweis: *Ändern Sie nicht den zurückgegebenen Pufferzeiger. Dadurch werden möglicherweise die Datenstrukturen des JavaScript-Interpreters beschädigt. Wenn Sie den String ändern möchten, müssen Sie die Zeichen in einen anderen Puffer kopieren und einen JavaScript-String erstellen.*

Argumente
JSContext `*cx`, JSVal `v`, unsigned integer `*pLength`

- Das Argument `*cx` ist ein Zeiger auf die unspezifische JSContext-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument `v` ist die JSVal-Struktur aus der der String in Dreamweaver extrahiert wird.
- Das Argument `*pLength` ist ein Zeiger auf eine vorzeichenlose Ganzzahl. Die Funktion weist `*pLength` die Länge des Strings in Byte zu.

Rückgabewerte
Bei einem erfolgreichen Vorgang ein Zeiger auf einen mit Null abgeschlossenen String bzw. bei Fehlern den Wert `null`. Die aufrufende Routine darf diesen String bei Beendigung nicht freigeben.

JSBool JS_ValueToInteger()

Beschreibung
Diese Funktion extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in eine Ganzzahl (sofern möglich) und gibt den konvertierten Wert zurück.

Argumente
`JSContext *cx`, `JSVal v`, `long *lp`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *v* ist die `JSVal`-Struktur, aus der die Ganzzahl extrahiert werden soll.
- Das Argument *lp* ist ein Zeiger auf eine 4-Byte-Ganzzahl. Diese Funktion speichert den konvertierten Wert in **lp*.

Rückgabewerte
Ein boolescher Wert. `JS_TRUE` gibt an, dass der Vorgang erfolgreich war. `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToDouble()

Beschreibung
Diese Funktion extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in eine Gleitkommazahl doppelter Genauigkeit (sofern möglich) und gibt den konvertierten Wert zurück.

Argumente

`JSContext *cx, JSVal v, double *dp`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *v* ist die `JSVal`-Struktur, aus der die Gleitkommazahl doppelter Genauigkeit extrahiert werden soll.
- Das Argument *dp* ist ein Zeiger auf eine 8-Byte-Gleitkommazahl. Diese Funktion speichert den konvertierten Wert in **dp*.

Rückgabewerte

Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToBoolean()

Beschreibung

Diese Funktion 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`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *v* ist die `JSVal`-Struktur, aus der der boolesche Wert extrahiert werden soll.
- Das Argument *bp* ist ein Zeiger auf einen booleschen Wert des Typs `JSBool`. Diese Funktion speichert den konvertierten Wert in **bp*.

Rückgabewerte

Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ValueToObject()

Beschreibung

Diese Funktion extrahiert ein Funktionsargument aus einer `JSVal`-Struktur, konvertiert es in ein Objekt (sofern möglich) und gibt den konvertierten Wert zurück. Wenn es sich bei dem Objekt um ein Array handelt, verwenden Sie `JS_GetArrayLength()` und `JS_GetElement()`, um den Inhalt zu lesen.

Argumente

`JSContext *cx, JSVal v, JSObject **op`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *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` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JS_ValueToUCString()

Beschreibung
Diese Funktion extrahiert ein Funktionsargument aus einer JSVal-Struktur, konvertiert es nach Möglichkeit in einen String und gibt den konvertierten Wert zurück.

Hinweis: *Ändern Sie nicht den zurückgegebenen Pufferzeiger. Dadurch werden möglicherweise die Datenstrukturen des JavaScript-Interpreters beschädigt. Wenn Sie den String ändern möchten, müssen Sie die Zeichen in einen anderen Puffer kopieren und einen JavaScript-String erstellen.*

Argumente
JSContext *`*cx`*, JSVal *`v`*, unsigned int *`*pLength`*

- Das Argument *`*cx`* ist der Zeiger auf eine unspezifische JSContext-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *`v`* ist die JSVal-Struktur aus der der String in Dreamweaver extrahiert wird.
- Das Argument *`*pLength`* ist ein Zeiger auf eine vorzeichenlose Ganzzahl. Die Funktion weist *`*pLength`* die Länge des Strings in Byte zu.

Rückgabewerte
Bei einem erfolgreichen Vorgang ein Zeiger auf einen mit Null abgeschlossenen UTF-8-String bzw. bei Fehlern den Wert `null`. Die aufrufende Routine darf diesen String bei Beendigung nicht freigeben.

JSBool JS_StringToValue()

Beschreibung
Diese Funktion speichert einen Rückgabewert-String in einer JSVal-Struktur. Sie erstellt ein neues JavaScript-Stringobjekt.

Argumente
JSContext *`*cx`*, JSVal *`*bytes`*, *size_t sz*, JSVal*`*vp`*

- Das Argument *`*cx`* ist der Zeiger auf eine unspezifische JSContext-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *`bytes`* ist der String, der von Dreamweaver in der JSVal-Struktur gespeichert wird. Die Stringdaten werden kopiert, sodass der aufrufende Code den String freigeben muss, wenn er nicht mehr benötigt wird. Wenn die Größe des Strings nicht angegeben ist (siehe Argument *`sz`*), muss der Wert mit Null abgeschlossen sein.
- Das Argument *`sz`* ist die Größe des Strings in Byte. Wenn *`sz`* gleich `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 wird.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JSBool JS_UCStringToValue()

Beschreibung
Diese Funktion speichert einen Rückgabewert-String in einer `JSVal`-Struktur. Sie erstellt ein neues JavaScript-Stringobjekt.

Argumente
JSContext *`*cx`*, JSVal *`*bytes`*, `size_t` *`sz`*, JSVal *`*vp`*

- Das Argument *`*cx`* ist der Zeiger auf eine unspezifische JSContext-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *`*bytes`* ist der String, der von Dreamweaver in der JSVal-Struktur gespeichert wird. Die Stringdaten werden kopiert, sodass der aufrufende Code den String freigeben muss, wenn er nicht mehr benötigt wird. Wenn die Größe des Strings nicht angegeben ist (siehe Argument *`sz`*), muss der Wert mit Null abgeschlossen sein.
- Das Argument *`sz`* ist die Größe des Strings in Byte. Wenn *`sz`* gleich `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 wird.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

Hinweis: *Die Methode `JS_UCStringToValue()` gleicht `JSBool JS_StringToValue()` in jeder Beziehung, gibt jedoch einen String im Format UTF-8 zurück.*

JSBool JS_DoubleToValue()

Beschreibung
Diese Funktion speichert einen Rückgabewert als Gleitkommazahl in einer JSVal-Struktur.

Argumente
`JSContext *cx`, `double dv`, `JSVal *vp`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *dv* ist eine 8-Byte-Gleitkommazahl.
- Das Argument *vp* ist ein Zeiger auf die `JSVal`-Struktur, in die der Inhalt der Gleitkommazahl kopiert werden soll.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

Rückgabewerte
Eine `JSVal`-Struktur mit dem booleschen Wert, der als Argument an die Funktion übergeben wird.

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 als Argument an die Funktion übergeben wurde.

JSVal JS_ObjectToValue()

Beschreibung
Diese Funktion speichert ein Rückgabewert-Objekt in `JSVal`. Verwenden Sie `JS_NewArrayObject()`, um ein Array-Objekt zu erstellen, und `JS_SetElement()`, um den entsprechenden 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 als Argument an die Funktion übergeben wurde.

char *JS_ObjectType()

Beschreibung
Wenn ein Objektverweis vorhanden ist, gibt `JS_ObjectType()` den Klassennamen des Objekts zurück. Für ein DOM-Objekt gibt die Funktion beispielsweise den Wert "`Document`" zurück. Ist das Objekt ein Knoten im Dokument, gibt die Funktion den Wert "`Element`" zurück. Für ein Array-Objekt gibt die Funktion den Wert "`Array`" zurück.

Hinweis: *Ändern Sie den zurückgegebenen Pufferzeiger nicht, da die Datenstrukturen des JavaScript-Interpreters sonst möglicherweise beschädigt werden.*

Argumente

`JSObject *obj`

In der Regel wird dieses Argument an die Funktion `JS_ValueToObject()` übergeben und konvertiert.

Rückgabewerte

Ein Zeiger auf einen mit Null abgeschlossenen String. Dieser String darf vom aufrufenden Code nach Abschluss nicht freigegeben werden.

JSObject *JS_NewArrayObject()

Beschreibung

Diese Funktion erstellt ein neues Objekt mit einem Array von `JSVal`-Werten.

Argumente

`JSContext *`*`cx`*, `unsigned int` *`length`*, `JSVal *v`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *length* ist die Anzahl der Elemente, die das Array enthalten kann.
- Das Argument *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. bei einem Fehler der Wert `null`.

long JS_GetArrayLength()

Beschreibung

Wenn ein Zeiger auf ein Array-Objekt verweist, ruft diese Funktion die Anzahl der Elemente im Array ab.

Argumente

`JSContext *`*`cx`*, `JSObject*`*`obj`*

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *obj* ist ein Zeiger auf ein Array-Objekt.

Rückgabewerte

Die Anzahl der Elemente im Array. 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`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *obj* ist ein Zeiger auf ein Array-Objekt.
- Das Argument *index* ist ein Ganzzahlindex für das Array. Das erste Element ist `index` 0 und das letzte Element ist `index` (`length - 1`).
- Das Argument *v* ist ein Zeiger auf einen `jsval`-Wert, in den der Inhalt der `JSVal`-Struktur des Arrays kopiert werden soll.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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*

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *obj* ist ein Zeiger auf ein Array-Objekt.
- Das Argument *index* ist ein Ganzzahlindex für das Array. Das erste Element ist `index` 0 und das letzte Element ist `index` (`length - 1`).
- Das Argument *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` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

JSBool JS_ExecuteScript()

Beschreibung
Diese Funktion kompiliert einen JavaScript-String und führt ihn aus. Wenn das Skript einen Rückgabewert generiert, wird dieser in **rval* zurückgegeben.

Argumente

`JSContext *cx`, `JSObject *obj`, *`char *script`*, *`unsigned int sz`*, `JSVal *rval`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *obj* ist ein Zeiger auf das Objekt, in dessen Kontext das Skript ausgeführt wird. Während der Skriptausführung entspricht das Schlüsselwort `this` diesem Objekt. In der Regel handelt es sich dabei um den Zeiger auf `JSObject`, der an die JavaScript-Funktion übergeben wird.
- Das Argument *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 sein.
- Das Argument *sz* ist die Größe des Strings in Byte. Wenn *sz* 0 ist, wird die Länge des mit Null abgeschlossenen Strings automatisch berechnet.
- Das Argument *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 zur Ursache des Skriptfehlers anzuzeigen (z. B. „Falsche Anzahl von Argumenten“).

Argumente

`JSContext *cx`, `char *error`, `size_t sz`

- Das Argument *cx* ist der Zeiger auf eine unspezifische `JSContext`-Struktur, der an die JavaScript-Funktion übergeben wird.
- Das Argument *error* ist ein String, der die Fehlermeldung enthält. Der String wird kopiert und muss vom aufrufenden Code freigegeben werden, wenn er nicht mehr benötigt wird. Wenn die Größe des Strings nicht angegeben wird (siehe Argument *sz*), muss der Wert mit Null abgeschlossen sein.
- Das Argument *sz* ist die Größe des Strings in Byte. Wenn *sz* gleich `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, wird mit den Funktionen auf die angegebene Datei oder den angegebenen Ordner zugegriffen.

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“. Dies hat zur Folge, dass nur Benutzer mit Administratorrechten Änderungen am Dreamweaver-Ordner „Configuration“ vornehmen können. Damit auch Benutzer in Mehrbenutzersystemen eigene Konfigurationen erstellen und verwalten können, wird in Dreamweaver ein separater Ordner „Configuration“ für jeden Benutzer erstellt. Jedes Mal, wenn Dreamweaver oder eine JavaScript-Erweiterung in den Ordner „Configuration“ von Dreamweaver schreibt, verwendet Dreamweaver automatisch den Ordner „Configuration“ des Benutzers. Jeder Benutzer kann so eigene Dreamweaver-Konfigurationseinstellungen festlegen, ohne dabei die Einstellungen anderer Benutzer zu beeinträchtigen.

Dreamweaver erstellt den Ordner „Configuration“ des Benutzers an einem Speicherort, auf den der jeweilige Benutzer vollen Schreib- und Lesezugriff hat. Der Speicherort des Benutzerordners „Configuration“ auf dem Computer ist vom Betriebssystem abhängig.

Unter Windows 2000 und Windows XP:

```
<drive>:\Documents and Settings\<username>\Application Data\Adobe\\
Dreamweaver CS5\Configuration
```

Hinweis: *Dieser Ordner befindet sich unter Windows XP möglicherweise in einem versteckten Ordner.*

Unter Mac OS X:

```
<drive>:Users:<username>:Library:Application Support:Adobe:Dreamweaver CS5:Configuration
```

In vielen Fällen öffnen JavaScript-Erweiterungen Dateien und schreiben in den Ordner „Configuration“. JavaScript-Erweiterungen können auf das Dateisystem über DWFile oder MMNotes sowie durch Übergeben einer URL an die Funktion `dreamweaver.getDocumentDOM()` zugreifen. Wenn eine Erweiterung auf das Dateisystem in einem Ordner „Configuration“ zugreift, wird in der Regel die Funktion `dw.getConfigurationPath()` verwendet und der Dateiname hinzugefügt. Alternativ wird der Pfad über die Eigenschaft `dom.URL` eines geöffneten Dokuments abgerufen und dann der Dateiname hinzugefügt. 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 sich das Dokument im Ordner „Configuration“ eines Benutzers befindet.

Immer wenn eine JavaScript-Erweiterung eine Datei im Ordner „Configuration“ von Dreamweaver öffnet, unterbricht Dreamweaver den Zugriff und überprüft zunächst 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. Anschließend wird der Aufruf an den Ordner „Configuration“ des Benutzers weitergeleitet.

Wenn ein Benutzer unter Windows 2000 oder Windows XP die Datei „file:///C|/Programme/Adobe/Adobe Dreamweaver CS5/Configuration/Objects/Common/Table.htm“ aufruft, sucht Dreamweaver im Ordner „C:\Dokumente und Einstellungen*Benutzername*\adobe\Dreamweaver CS5\Configuration\Objects\Common“ nach der Datei „Table.htm“ und verwendet diese Datei, sofern sie vorhanden ist.

C-Level-Erweiterungen bzw. gemeinsam genutzte Bibliotheken müssen für Schreib- und Lesezugriffe auf den Dreamweaver-Ordner „Configuration“ die API für Dateizugriff- und Mehrbenutzerkonfiguration verwenden. Mithilfe der API für Dateizugriff- und Mehrbenutzerkonfiguration kann in Dreamweaver der Lese- und Schreibzugriff auf den Ordner „Configuration“ des Benutzers hergestellt werden. Darüber hinaus wird sichergestellt, dass Dateibearbeitungen nicht aufgrund fehlender Zugriffsberechtigungen fehlschlagen. Verwenden Sie die API für Dateizugriff- und Mehrbenutzerkonfiguration, wenn die Dateien im Dreamweaver-Ordner „Configuration“, auf die Ihre C-Level-Erweiterung zugreift, über JavaScript mit DWFile-, MMNotes- oder DOM-Bearbeitungen erstellt wurden. Es ist möglich, dass sich diese Dateien im Ordner „Configuration“ des Benutzers befinden.

Hinweis: *Die meisten JavaScript-Erweiterungen müssen nicht geändert werden, um Schreibvorgänge im Ordner „Configuration“ des Benutzers durchzuführen. Nur gemeinsam genutzte C-Bibliotheken, die in diesen Ordner schreiben, müssen aktualisiert werden, damit die Funktionen der API für Dateizugriff- und Mehrbenutzerkonfiguration verwendet werden können.*

Wenn Sie eine Datei aus dem Dreamweaver-Ordner „Configuration“ löschen, wird ein Eintrag in einer Maskierungsdatei eingefügt. Dieser Eintrag wird hinzugefügt, um anzugeben, welche Dateien im Ordner „Configuration“ nicht in der Benutzeroberfläche angezeigt werden sollen. Maskierte Dateien oder Ordner 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 „onepixelborder.csn“ löschen, wird in Dreamweaver eine Datei mit der Bezeichnung „mm_deleted_files.xml“ in den Ordner „Configuration“ des Benutzers geschrieben. Die Datei sieht wie folgt aus:

```
<?xml version = "1.0" encoding="utf-8" ?>
    <deleteditems>
    <item name="snippets/javascript/" />
    <item name="snippets/html/onepixelborder.csn" />
    </deleteditems>
```

Beim Übernehmen von Daten in das Bedienfeld „Codefragmente“ werden in Dreamweaver alle Dateien im Ordner „Configuration/Snippets“ des Benutzers gelesen. Darüber hinaus werden alle Dateien im Dreamweaver-Ordner „Configuration/Snippets“ gelesen, mit Ausnahme des Ordners „Configuration/Snippets/javascript“ und der Datei „Configuration/Snippets/html/onepixelborder.csn“. Die dabei erstellte Liste der Dateien wird der Liste des Bedienfelds „Codefragmente“ hinzugefügt.

Wenn eine C-Level-Erweiterung die Funktion `MM_ConfigFileExists()` für die URL `file:///c|Programme/Adobe/Adobe Dreamweaver CS5/Configuration/Snippets/JavaScript/onepixelborder.csn` aufruft, wird der Wert `false` zurückgegeben. Wenn eine JavaScript-Erweiterung die Funktion `dw.getDocumentDom(file:///c|Programme/Adobe/Adobe Dreamweaver CS5/Configuration/Snippets/JavaScript/onepixelborder.csn)` aufruft, wird der Wert `null` zurückgegeben.

Durch Bearbeiten der Datei „mm_deleted_files.xml“ können Sie verhindern, dass in Dreamweaver bestimmte Dateien in der Benutzeroberfläche angezeigt werden, z. B. Objekte, gelöschte Inhalte eines neuen Dialogfelds usw. Sie können die Funktion `MM_DeleteConfigfile()` aufrufen, um der Datei „mm_deleted_files.xml“ Dateipfade 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 Ordner ab, die im Ordner „Configuration“ des Benutzers und im Dreamweaver-Ordner „Configuration“ vorhanden sind und durch die Datei „mm_deleted_files.xml“ gefiltert werden.

Argumente
*char *fileURL, char *constraints*

- Das Argument *char *fileUrl* verweist auf den String mit dem Namen des Ordners, dessen Inhalt Sie abrufen möchten. Der String muss dem URL-Format „file:///“ entsprechen. Die Funktion akzeptiert im URL-String „file:///“ Sternchen (*) und Fragezeichen (?) als gültige Platzhalterzeichen. Mithilfe von Sternchen (*) können Sie ein oder mehrere unbekannte Zeichen und mit Fragezeichen (?) ein einzelnes unbekanntes Zeichen angeben.
- Als Wert für das Argument *char *constraints* kann `files`, `directories` oder der Wert `null` angegeben werden. 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 im Dreamweaver-Ordner „Configuration“, die durch die 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, wird sie im Ordner „Configuration“ des Benutzers oder im Dreamweaver-Ordner „Configuration“ gesucht. Die Funktion überprüft auch, ob der Dateiname in der Datei „mm_deleted_files.xml“ aufgeführt ist. Ist dies der Fall, gibt die Funktion den Wert `false` zurück.

Argumente
*char *fileUrl*

Das Argument *char *fileUrl* verweist auf den String für die gewünschte Datei im URL-Format „file:///“.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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 bei Aufrufen für Systemdateifunktionen verwenden. Sie müssen das Datei-Handle mit einem Aufruf der Systemfunktion `_close` schließen.

Wenn die Datei eine Konfigurationsdatei ist, wird sie entweder im Ordner „Configuration“ des Benutzers oder im Dreamweaver-Ordner „Configuration“ gefunden. Wenn Sie die Konfigurationsdatei zum Schreiben öffnen, wird mit dieser Funktion die Datei im Ordner „Configuration“ des Benutzers angelegt, auch wenn sie bereits im Dreamweaver-Ordner „Configuration“ vorhanden ist.

Hinweis: *Wenn Sie die Datei vor dem Schreiben lesen möchten, können Sie sie im Modus `"read"` öffnen. Zum Schreiben schließen Sie dann das read-Handle und öffnen die Datei erneut im Modus `"write"` oder `"append"`.*

Argumente

*char *fileURL, char *mode*

- Das Argument *char *fileURL* verweist auf einen String mit dem Namen der zu öffnenden Datei im URL-Format „file://“. Wird dabei ein Pfad im Dreamweaver-Ordner „Configuration“ festgelegt, ermittelt die Funktion `MM_OpenConfigFile()` den Pfad vor dem Öffnen der Datei.
- Das Argument *char *mode* verweist auf einen String, der festlegt, wie Sie die Datei öffnen möchten. Sie können `null`, `"read"`, `"write"` oder `"append"` angeben. Wenn Sie `"write"` für eine Datei angeben, die noch nicht vorhanden ist, wird diese mit der Funktion `MM_OpenconfigFile()` erstellt. Wenn Sie `"write"` angeben, öffnet die Funktion `MM_OpenConfigFile()` die Datei mit Exklusivzugriff. Wenn Sie `"read"` angeben, öffnet die Funktion `MM_OpenConfigFile()` die Datei mit nicht exklusivem Zugriff.

  Wenn Sie die Datei im Modus `"write"` öffnen, werden bereits vorhandene Daten in der Datei vor dem Schreiben neuer Daten gekürzt. Wenn Sie die Datei im Modus `"append"` öffnen, werden alle neu geschriebenen Daten am Ende der Datei angefügt.

Rückgabewerte

Eine Ganzzahl, die das Datei-Handle des Betriebssystems für diese Datei angibt. Gibt `-1` zurück, wenn die Datei nicht gefunden wurde oder nicht vorhanden ist.

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.
- Das Argument *unsigned long *attrs* ist die Adresse einer Ganzzahl mit den zurückgegebenen Attributen (Informationen zu verfügbaren Attributen siehe „JSBool MM_SetConfigFileAttributes()“ auf Seite 386).
- 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 zur 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 zur Systemzeit finden Sie unter `DWfile.getCreationDate()` im *Dreamweaver API-Referenzhandbuch*.

Rückgabewerte

Ein boolescher Wert. `JS_TRUE` gibt an, dass der Vorgang erfolgreich war. `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist. Gibt `JS_FALSE` zurück, wenn die Datei nicht vorhanden ist oder beim Abrufen der Attribute ein Fehler 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 angegebenen Attribute für die Datei fest, wenn sie sich von den aktuellen Attributen unterscheiden.

Wenn sich die angegebene Datei-URL im Ordner „Configuration“ von Dreamweaver befindet, kopiert diese Funktion die Datei zunächst in den Ordner „Configuration“ des Benutzers, bevor die Attribute festgelegt werden. Wenn die Attribute identisch mit den aktuellen Dateiattributen sind, wird die Datei nicht kopiert.

Argumente

*char *fileURL, unsigned long attrs*

- Das Argument *char *fileURL* verweist auf einen String mit dem Namen der Datei im URL-Format „file://“, deren Attribute Sie festlegen möchten.
- Das Argument *unsigned long attrs* legt die Attribute für die Datei fest. Sie können die folgenden Attributkonstanten mit einem logischen `ODER` verknüpfen:

  ```
  MM_FILEATTR_NORMAL
  MM_FILEATTR_RDONLY
  MM_FILEATTR_HIDDEN
  MM_FILEATTR_SYSTEM
  MM_FILEATTR_SUBDIR
  ```

Rückgabewerte

Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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 erstellt am angegebenen Speicherort einen Ordner.

Wenn das Argument *fileURL* einen Unterordner im Dreamweaver-Ordner „Configuration“ angibt, erstellt die Funktion diesen Ordner im Ordner „Configuration“ des Benutzers. Wenn mit *fileURL* kein Ordner im Dreamweaver-Ordner „Configuration“ angegeben wird, werden mit der Funktion der angegebene Ordner sowie alle nicht bereits vorhandenen übergeordneten Ordner erstellt.

Argumente
*char *fileURL*

- Das Argument *char *fileURL* verweist auf einen String im URL-Format „file://“ für den Konfigurationsordner, den Sie erstellen möchten.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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 die zugehörigen Dateien und Unterordner. Wenn sich der Ordner im Ordner „Configuration“ von Dreamweaver befindet, wird er in der Datei „mm_deleted_files.xml“ als gelöscht markiert.

Argumente
*char *fileURL*

- Das Argument *char *fileURL* verweist auf einen String für den zu entfernenden Ordner im URL-Format „file://“.

Rückgabewerte
Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, 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, wenn sie vorhanden ist. Wenn die Datei in einem Unterordner des Ordners „Configuration“ von Dreamweaver vorhanden ist, markiert die Funktion die Datei in der Datei „mm_deleted_files.xml“ als gelöscht.

Wenn das Argument *fileURL* keinen Ordner im Ordner „Configuration“ von Dreamweaver angibt, löscht die Funktion die angegebene Datei.

Argumente

*char *fileURL*

- Das Argument *char *fileURL* verweist auf einen String, der den Namen des zu entfernenden Konfigurationsordners im URL-Format „file://“ enthält.

Rückgabewerte

Ein boolescher Wert: `JS_TRUE` gibt an, dass der Vorgang erfolgreich war, `JS_FALSE` gibt an, dass der Vorgang fehlgeschlagen ist.

Beispiel

```
char dwConfig = "file:///c:|Program Files/Adobe/Adobe Dreamweaver CS3
    /Configuration/Objects/insertbar.xml";
MM_DeleteConfigFile(dwConfig);
```

Aufrufen von C-Funktionen über JavaScript

Nachdem Sie sich mit der C-Level-Erweiterbarkeit in Dreamweaver und deren Abhängigkeit von bestimmten Datentypen und Funktionen vertraut gemacht haben, erhalten Sie im Folgenden Informationen zum Erstellen von Bibliotheken und zum Aufrufen von Funktionen.

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:

- Die Header-Datei „mm_jsapi.h“ enthält die Definitionen für alle Datentypen und Funktionen, die unter „C-Level-Erweiterbarkeit und JavaScript-Interpreter“ auf Seite 371 erläutert sind.
- Die Datei „mm_jsapi_environment.h“ definiert die MM_Environment.h-Struktur.
- Die Datei „MMInfo.h“ ermöglicht den Zugriff auf die API für Design Notes.
- 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 kompilieren können. Die entsprechende Datei zum Erzeugen eines Mach-O-Bundles mit Metrowerks CodeWarrior ist „Sample.mcp“. „Sample.xcode“ ist die entsprechende Datei für Apple Xcode. Bei Verwendung eines anderen Tools können Sie das Makefile selbst erstellen.

Erstellen der DLL mit VS.Net 2003 unter Windows

1. Wählen Sie „Datei“ > „Öffnen“ und dann die Datei „Sample.mak“ aus. Geben Sie dazu als Dateityp „Alle Dateien“ (*.*) an. (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“ aus.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei „Sample.dll“ im gleichen Ordner wie „Sample.mak“ (bzw. in einem Unterordner).

Erstellen der DLL mit Microsoft Visual C++ unter Windows

1. Wählen Sie in Microsoft Visual C++ den Befehl „Datei“ > „Arbeitsbereich öffnen“ und dann die Datei „Sample.mak“ aus.
2. Wählen Sie „Erstellen“ > „Alles neu erstellen“ aus.

Nach Abschluss des Erstellungsvorgangs befindet sich die Datei „Sample.dll“ im gleichen Ordner wie „Sample.mak“ (bzw. in einem Unterordner).

Erstellen der gemeinsam genutzten Bibliothek mit Metrowerks CodeWarrior ab Version 9 auf Macintosh-Systemen

1. Öffnen Sie die Datei „Sample.mcp“.
2. Erstellen Sie das Projekt („Project“ > „Make“), um ein Mach-O-Bundle zu generieren.

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.*

Erstellen der gemeinsam genutzten Bibliothek mit Apple Xcode ab Version 1.5 auf Macintosh-Systemen

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.*

Aufrufen der computeSum()-Funktion über das Objekt „Horizontale Linie einfügen“

1. Erstellen Sie den Unterordner „JSExtensions“ im Dreamweaver-Ordner „Configuration“.
2. Kopieren Sie die Datei „Sample.dll“ (Windows) bzw. „Sample.bundle“ (Macintosh) in den Ordner „JSExtensions“.
3. Öffnen Sie die Datei „HR.htm“ im Ordner „Configuration/Objects/Common“ mit einem Texteditor.
4. Fügen Sie die Zeile `alert(Sample.computeSum(2,2));` wie folgt in die Funktion `objectTag()` ein:

```
function objectTag() {
    // Return the html tag that should be inserted
    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“ aus.

Ein Dialogfeld mit der Zahl 4, d. h. dem Ergebnis der Addition der Zahlen 2 und 2, wird angezeigt.

Kapitel 24: Ordner „Shared“

Der Ordner „Shared“ ist das zentrale Repository für Dienstprogrammfunktionen, Klassen und Bilder, die in allen Erweiterungen verwendet werden. Jede Erweiterung kann auf die Dateien in den Unterordnern des Ordners „Shared“ verweisen. Sie können zudem gemeinsame benutzerdefinierte Dienstprogramme zu denen hinzufügen, die bereits in Adobe Dreamweaver bereitgestellt werden. Die Konfigurationsordner für mehrere Benutzer, die unter Windows XP, Windows 2000 und Mac OS X installiert sind, enthalten ebenfalls einen Ordner „Shared“ für benutzerdefinierte Anpassungen. Wenn Sie beispielsweise eine Erweiterung von Adobe Exchange installieren, stellen Sie möglicherweise fest, dass mit der neuen Erweiterung Inhalte zu Ihrem Benutzerordner „Configuration/Shared“ statt zum Ordner „Configuration/Shared“ von Dreamweaver hinzugefügt werden. Weitere Informationen zu den Dreamweaver-Konfigurationsordnern auf einem Mehrbenutzercomputer finden Sie unter „Konfigurationsordner bei mehreren Benutzern“ auf Seite 83.

Inhalt des Ordners „Shared“

Der Ordner „Shared“ verfügt über Unterordner mit Dateien, die von mehreren Erweiterungen verwendet werden, einschließlich Funktionen zum Durchsuchen der Ordnerstruktur eines Benutzers, zum Einfügen eines Struktursteuerelements, zum Erstellen bearbeitbarer Raster und weiterer Funktionen.

Hinweis: *Die JavaScript-Dateien im Ordner „Shared“ verfügen über Kommentare im Code, die Informationen zu den entsprechenden Funktionen enthalten.*

Sie sollten sich nicht nur die JavaScript-Dateien im Ordner „Shared“ ansehen, um zu überprüfen, wie diese verwendet werden, sondern auch im Ordner „Configuration“ nach HTML-Dateien suchen, die diese JavaScript-Dateien enthalten.

In der Regel verwenden Sie die Funktionen und Ressourcen in den Ordnern „Common“ und „MM“ oder Sie fügen im Ordner „Common“ Ressourcen zur Verwendung mit neuen Erweiterungen ein. 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 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.

Ordner „Common“

Der Ordner „Common“ enthält gemeinsam verwendete Skripts und Klassen für Erweiterungen von Drittanbietern.

Datei	Beschreibung
CodeBehindMgr.js	Enthält Funktionen zum Erstellen von CodeBehind-Dokumenten. Mit CodeBehind-Dokumenten können Sie spezielle Seiten erstellen, die den Code für die Programmlogik einer Benutzeroberfläche vom Code für den Entwurf der Benutzeroberfläche trennen. Mit den in dieser Datei definierten Methoden von `JSCodeBehindMgr` können neue CodeBehind-Dokumente erstellt und die Verknüpfungen zu Entwurfsdokumenten verwaltet werden.
ColumnValueNodeClass.js	Enthält Funktionen zum Zuordnen von Datenbankspalten zu Werten. Mithilfe der in dieser Datei definierten Methoden von `ColumnValueNode` können Sie verschiedene Werte und Eigenschaften einer Datenbankspalte abrufen und festlegen. Dreamweaver verwendet diese Speicherklasse beim Anwenden und Untersuchen von Objekten für Bearbeitungsfunktionen (Objekte zum Einfügen und Aktualisieren von Datensätzen) sowie beim Verwenden der SQLStatement-Klasse.
CompilerClass.js	Enthält Funktionen für eine Basisklasse, die von `CompilerASPNetCSharp` und `CompilerASPNVBNet` verwendet wird, jedoch nicht zur Unterstützung weiterer Compiler erweitert werden konnte.
DataSourceClass.js	Enthält Funktionen, die die Rückgabestruktur für „findDynamicSources()“ auf Seite 319 definieren.
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 z. B. in den Serververhalten erweiterter Datensatzgruppen zu erstellen, erstellen Sie in Ihrer HTML-Datei eine spezielle `<select>`-Liste mit `type="mmdatabasetree"`. Fügen Sie eine `CBTreeControl`-Klasse an das HTML-Steuerelement an, indem Sie den `<select>`-Listennamen an den Klassenkonstruktor übergeben. Verwenden Sie anschließend die `DBTreeControl`-Funktionen zum Ändern des Steuerelements.
dotNetUtils.js	Enthält Funktionen zur Verwendung von Objekteigenschafteninspektoren und Serververhalten für ASP.NET-Formularsteuerelemente, die übersetzt werden.
dwscripts.js	In dieser Hauptdatei finden Sie zahlreiche nützliche Funktionen für alle Dreamweaver-Erweiterungen. Sie enthält Funktionen zum Verwenden von Strings, Dateien, Design Notes usw.
dwscriptsExtData.js	Diese Datei ist eine Erweiterung der Datei „dwscripts.js“. Diese Datei vereinfacht die Verwendung von Serververhalten, insbesondere der EDML-Dateien für Serververhalten. Sie wird in Dreamweaver sehr häufig bei der Implementierung von Serververhalten verwendet.
dwscriptsServer.js	Diese Datei ist eine Erweiterung der Datei „dwscripts.js“. Sie enthält spezifische Funktionen für Servermodelle. Viele dieser Funktionen werden für Serververhalten verwendet.
GridControlClass.js	Verwenden Sie diese Klasse zum Erstellen und Ändern bearbeitbarer Raster. Erstellen Sie eine spezielle Auswahlliste in Ihrem HTML-Code und fügen Sie in JavaScript diese Klasse an, um das Raster zu bearbeiten.
ImageButtonClass.js	Diese Klasse vereinfacht die Steuerung der Darstellung einer Schaltfläche in folgenden Fällen: gedrückte Schaltfläche, Mauszeiger über der gedrückten Schaltfläche, Mauszeiger über der Schaltfläche, deaktivierte gedrückte Schaltfläche.
ListControlClass.js	Enthält Funktionen zum Verwalten eines `<select>`-Tags, das auch als Listensteuerelement bezeichnet wird. Die Methoden des Objekts `ListControl` in dieser Datei rufen den Wert des Steuerelements `SELECT` 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 `RadioGroup` 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.
SBDatabaseCallClass.js	Eine Unterklasse der Klasse `ServerBehavior`. Diese Klasse enthält spezifische Funktionen für Datenbankaufrufe, z. B. zum Aufrufen einer gespeicherten Prozedur, zum Verwenden von SQL für die Rückgabe einer Datensatzgruppe usw. Hierbei handelt es sich um eine abstrakte Basisklasse, die nicht allein erstellt und verwendet werden kann. Um sie verwenden zu können, müssen Sie Unterklassen für `SBDatabaseCall()` erstellen und die Platzhalterfunktionen implementieren. Dreamweaver verwendet diese Klasse zum Implementieren der zugehörigen Datensatzgruppe und der zugehörigen gespeicherten Prozeduren für Serververhalten.

Datei	Beschreibung
ServerBehaviorClass.js	Enthält Funktionen, die Informationen über Serververhalten an Dreamweaver übermitteln. Sie können diese Klasse bei der Implementierung benutzerdefinierter Serververhalten als Unterklasse verwenden.
ServerSettingsASPNet.js	Enthält Funktionen, die Eigenschaften eines ASP.NET-Servers speichern.
SQLStatementClass.js	Enthält Funktionen, mit denen Sie SQL-Anweisungen (z. B. `SELECT`, `INSERT`, `UPDATE` und `DELETE`) sowie Anweisungen für gespeicherte Prozeduren erstellen und bearbeiten können.
tagDialogsCmn.js	Enthält Funktionen, mit denen Sie benutzerdefinierte Tag-Dialogfelder erstellen können. Die in dieser Datei definierten Methoden des Objekts `tagDialog` ändern Attribute und Werte eines bestimmten Tags.
TagEditClass.js	Enthält Funktionen, die Tags bearbeiten, ohne das DOM der aktuellen Seite zu ändern. Die in dieser Datei definierten Methoden des Objekts `TagEdit` rufen den Wert, die Attribute und die untergeordneten Elemente des Tags ab und legen sie fest.
TreeControlClass.js	Enthält Funktionen zum Verwalten eines Struktursteuerelements in Dreamweaver. Die in dieser Datei definierten Methoden des Objekts `TreeControl` rufen die Werte in der Struktur ab, legen sie fest und ordnen sie neu an. Sie fügen diese Klasse an ein spezielles `MM:TREECONTROL`-Tag in Ihrem HTML-Code an, um die Funktionen des Struktursteuerelements zu verwalten.
XMLPropSheetClass.js	Enthält Funktionen zum Verwalten des Speicherorts und der Werte einer XML-Eigenschaftenseite.

Ordner „MM“

Der Ordner „MM“ enthält die gemeinsam verwendeten Skripts, Bilder und Klassen, die von den im Lieferumfang von Dreamweaver enthaltenen Erweiterungen verwendet werden. Dazu gehören u. a. die Skripts zum Erstellen einer Navigationsleiste, zum Angeben von Aufrufen für das Vorausladen sowie zum Festlegen der Tastaturbefehle.

Unterordner „Scripts“

Der Unterordner „Scripts“ enthält die folgenden Dienstprogrammfunktionen.

Datei	Beschreibung
CFCutilities.js	Enthält Dienstprogrammfunktionen für Adobe ColdFusion-Komponenten. Die 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 „menus.xml“ und Hinzufügen von Ereignisbenachrichtigungen zur Datei „menus.xml“.
FlashObjects.js	Enthält Funktionen zum Aktualisieren einer Farbauswahl, zum Überprüfen auf hexadezimale Farbwerte oder absolute Hyperlinks, zum Hinzufügen einer Dateierweiterung zu einem Dateinamen, zum Erzeugen von Fehlermeldungen, zum Festlegen von Flash-Attributen, zum Prüfen eines Hyperlinks auf Flash-Objekte usw.
insertFireworksHTML.js	Enthält Funktionen zum Einfügen von HTML-Code von Adobe Fireworks CS3 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. Enthält darüber hinaus zugehörige Dienstprogrammfunktionen.
jumpMenuUI.js	Enthält Funktionen für das Sprungmenü-Objekt und das Sprungmenü-Verhalten. Diese Funktionen füllen Menüoptionen, 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 zur Verwendung mit einer Navigationsleiste und den Navigationsleistenelementen. Dazu gehören Funktionen zum Hinzufügen, Löschen und Bearbeiten von Navigationsleistenelementen.

Datei	Beschreibung
NBInit.js	Enthält Funktionen für Navigationsleisten-Bildverhalten.
pageEncodings.js	Definiert verschiedene Sprachcodes.
preload.js	Enthält Funktionen zum Hinzufügen und Löschen von Vorlade-Bildaufrufen der Prozedur `BODY/onLoad MM_preloadImages`.
RecordsetDialogClass.js	Enthält die Klasse „static“ und Funktionen zum Anzeigen der Benutzeroberfläche für Serververhalten von Datensatzgruppen. Diese Funktionen legen fest, ob die einfache oder die erweiterte Benutzeroberfläche angezeigt wird. Enthält außerdem Funktionen, die von den verschiedenen Implementierungen der Benutzeroberflächen gemeinsam verwendet werden, und verwaltet den Wechsel zwischen den Benutzeroberflächen.
sbUtils.js	Enthält gemeinsam verwendete Funktionen zur Verwendung in Adobe-Serververhalten. Die Klasse `dwscripts` im Ordner „Configuration/Shared/Common/Scripts“ enthält Dienstprogramme für allgemeinere Zwecke.
setText.js	Enthält Funktionen zum Maskieren aller Zeichen mit spezieller Bedeutung eines Ausdruck-Strings bzw. zum Rückgängigmachen dieses Vorgangs und zum 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 zum Prüfen des Typs und der Version des Browsers.

Der Ordner „Scripts“ enthält außerdem die beiden Unterordner „Class“ und „CMN“.

Ordner „Class“

Der Ordner „Class“ enthält die folgenden Dienstprogrammfunktionen.

Datei	Beschreibung
classCheckbox.js	Unterstützt die Bearbeitung eines Kontrollkästchen-Steuerelements in der HTML-Erweiterung.
FileClass.js	Enthält eine Klasse, die eine Datei im Dateisystem darstellt. Die Pfade werden zur plattformübergreifenden Kompatibilität als URLs angegeben. Zu den Methoden zählen `toString()`, `getName()`, `getSimpleName()`, `getExtension()`, `getPath()`, `setPath()`, `isAbsolute()`, `getAbsolutePath()`, `getParent()`, `getAbsoluteParent()`, `exists()`, `getAttributes()`, `canRead()`, `canWrite()`, `isFile()`, `isFolder()`, `listFolder()`, `createFolder()`, `getContents()`, `setContents()`, `copyTo()` und `remove()`.
GridClass.js	Enthält eine Klasse, die `MM:TREECONTROL` verwaltet.
GridControlClass.js	Ältere Version von `GridControlClass` im Ordner „Common“. Siehe Datei „GridControlClass.js“ im Ordner „Shared/Common/Scripts“.
ImageButtonClass.js	Ältere Version von `ImageButtonClass` im Ordner „Common“. Siehe Datei „ImageButtonClass.js“ im Ordner „Shared/Common/Scripts“.
ListControlClass.js	Ältere Version von `ListControlClass` im Ordner „Common“. Siehe Datei „ListControlClass.js“ im Ordner „Shared/Common/Scripts“.
NameValuePairClass.js	Erstellt und verwaltet eine Liste mit Name-Wert-Paaren. Namen können beliebige Zeichen enthalten. Werte können leer sein, jedoch nicht auf `null` gesetzt werden, da dies einem Löschen entspricht.
PageControlClass.js	Beispiel einer Seitenklasse, die mit der Klasse `TabControl` verwendet wird. Siehe Beschreibung für die Datei „TabControlClass.js“.

Datei	Beschreibung
PreferencesClass.js	Enthält ein Objekt und Methoden, die alle Voreinstellungsinformationen für einen Befehl enthalten.
RadioGroupClass.js	Ältere Version von `RadioGroupClass` im Ordner „Common“. Siehe Datei „RadioGroupClass.js“ im Ordner „Shared/Common/Scripts“.
TabControlClass.js	Unterstützt die Erstellung einer Erweiterung mit mehreren Registerkartenansichten, `page.lastUnload()`.

Ordner „CMN“

Der Ordner „CMN“ enthält die folgenden Dienstprogrammfunktionen.

Datei	Beschreibung
dateID.js	Enthält zwei Funktionen: `createDateID()` und `decipherDateID()`. Liegen drei Strings vor (`"dayFormat"`, `"dateFormat"` und `"timeFormat"`), erstellt `createDateID()` eine ID für die drei Strings. Liegt ein Datums-Array vor, gibt `decipherDateID()` ein Array mit drei Elementen zurück: `dayFormat`, `dateFormat` und `timeFormat`.
displayHelp.js	Enthält eine Funktion, die das angegebene Hilfedokument anzeigt.
docInfo.js	Enthält Funktionen, die Informationen zum Benutzerdokument anzeigen. Zu den mit diesen Funktionen durchgeführten Vorgängen gehören die Rückgabe eines Arrays von Objektverweisen 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 für das Dreamweaver-DOM. Umfasst Funktionen, die den Stammknoten des aktiven Dokuments abrufen, ein bestimmtes benanntes Tag suchen, eine Liste von Knoten von dem angegebenen Startknoten erstellen, überprüfen, ob ein bestimmtes Tag in einem anderen Tag enthalten ist, verschiedene Vorgänge für Verhaltensfunktionen durchführen usw.
enableControl.js	Enthält die Funktion `SetEnabled()`, die je nach empfangenen Argument ein Steuerelement aktiviert oder deaktiviert. Es ist kein Problem, ein bereits aktiviertes Steuerelement zu aktivieren oder ein bereits deaktiviertes Steuerelement zu deaktivieren.
errmsg.js	Enthält Protokollierungsfunktionen zum Sammeln von Tracing-Ausgaben in einem Array von Protokollseiten, die in einem Dialogfeld angezeigt werden.
file.js	Enthält Funktionen, die sich auf Dateivorgänge beziehen. Mit diesen Funktionen kann der Benutzer nach lokalen Dateinamen suchen, den relativen Pfad in den URL-Pfad der Datei konvertieren, den Dateinamen des aktuellen Dokuments zurückgeben sowie feststellen, ob ein bestimmtes Dokument in der aktuellen Site gespeichert ist (und den dokumentrelativen Pfad zurückgeben) oder ob eine bestimmte Datei derzeit geöffnet ist.
form.js	Enthält Funktionen, die einem Textstring ein Formular hinzufügen, wenn im aktuellen Dokument oder AP-Element noch kein Formular vorhanden ist. Dazu gehören u. a. Funktionen zum Ermitteln, 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 einige nützliche Funktionen, die Kodierung ersetzen, das Maskieren von Anführungszeichen (") in Strings rückgängig machen, nach doppelten Objektnamen suchen sowie überprüfen, ob sich ein Knoten in einem Auswahlbereich befindet.
insertion.js	Enthält die Funktion `insertIntoDocument()`, die einen Textstring an der Einfügemarke in ein Dokument einfügt. Enthält außerdem die unterstützenden Funktionen `getHigherBlockTag()` und `arrContains()`. Die Funktion `getHigherBlockTag()` ruft das nächsthöhere `blockTag` ab, wie im `blockTags`-Array definiert, und die Funktion `arrContains()` sucht in einem Array nach einem bestimmten Element.
localText.js	Reservierte Variablen, die nicht zur allgemeinen Verwendung vorgesehen sind. Verwenden Sie stattdessen „Startup/mminit.htm“ oder die Strings in den XML-Dateien im Dreamweaver-Ordner „Configuration/Strings/*.xml“.

Datei	Beschreibung
menuItem.js	Enthält Funktionen, die Werte zu einem aufgeführten Menüelement hinzufügen oder aus einem aufgeführten Menüelement löschen.
niceName.js	Enthält Funktionen, die ein Array von Objektverweisen 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 Bearbeitung und Analyse von Textstrings. Zu diesen Funktionen gehören `extractArgs()`, `escQuotes()`, `unescQuotes()`, `quoteMeta()`, `errMsg()`, `badChars()`, `getParam()`, `quote()`, `stripSpaces()`, `StripChars()`, `AllInRange()`, `reformat()`, `trim()`, `createDisplayString()`, `entityNameEncode()`, `entityNameDecode()`, `stripAccelerator()` und `SprintF()`.
TemplateUtils.js	Enthält Dienstprogrammfunktionen für Dreamweaver-Vorlagen. Die Funktionen fügen einen bearbeitbaren Bereich oder einen 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 Warnmeldungen um.

Weitere Ordner

Im Folgenden werden weitere wichtige Unterordner des Ordners „Shared“ beschrieben:

Controls Der Ordner „Controls“ enthält die Steuerelemente, mit denen ein Serververhalten erstellt wird. Zu diesen Steuerelementen zählen Benutzeroberflächen für Text- und Datensatzgruppenmenüs.

Hinweis: *Diese Steuerelemente werden von der Serververhalten-Erstellung von Dreamweaver und von vielen Dreamweaver-Serververhalten verwendet. Einige davon sind jedoch auch zum Verwalten von Steuerelementen in Erweiterungen hilfreich.*

Fireworks Der Ordner „Fireworks“ enthält die Unterstützungsdateien für die Fireworks-Integration.

UltraDev Dieser Ordner wird in Dreamweaver hauptsächlich zur Gewährleistung der Abwärtskompatibilität verwendet. Er sollte nicht für neue Erweiterungen verwendet werden. Verwenden Sie den Dreamweaver-Ordner „Configuration/Shared/Common“, in dem sich die meisten dieser Funktionen ebenfalls befinden. Siehe „Ordner „Common““ auf Seite 391.

Verwenden des Ordners „Shared“

Suchen Sie zunächst 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-Bereich der Datei oder in einer anderen enthaltenen JavaScript-Datei verwenden. Objekte und Befehle können sogar mehrere JavaScript-Dateien miteinander verknüpfen. Diese JavaScript-Dateien können Ressourcen im Ordner „Shared“ verwenden.

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 Deklarationen `new listControl` 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 an:

```
<td  align="left"> <input name="linkText" type="text" class="basicTextField" value="">
```

und

```
<td align="left" nowrap><select name="linkPath" class="basicTextField" editable="true">
```

Nun 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.