

# HTMLBusiness- Sprachbeschreibung



**Release 4.6C**



## Copyright

© Copyright 2001 SAP AG. Alle Rechte vorbehalten.

Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die ausdrückliche schriftliche Genehmigung durch SAP AG nicht gestattet. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Die von SAP AG oder deren Vertriebsfirmen angebotenen Software-Produkte können Software-Komponenten auch anderer Software-Hersteller enthalten.

Microsoft<sup>®</sup>, WINDOWS<sup>®</sup>, NT<sup>®</sup>, EXCEL<sup>®</sup>, Word<sup>®</sup>, PowerPoint<sup>®</sup> und SQL Server<sup>®</sup> sind eingetragene Marken der Microsoft Corporation.

IBM<sup>®</sup>, DB2<sup>®</sup>, OS/2<sup>®</sup>, DB2/6000<sup>®</sup>, Parallel Sysplex<sup>®</sup>, MVS/ESA<sup>®</sup>, RS/6000<sup>®</sup>, AIX<sup>®</sup>, S/390<sup>®</sup>, AS/400<sup>®</sup>, OS/390<sup>®</sup> und OS/400<sup>®</sup> sind eingetragene Marken der IBM Corporation.

ORACLE<sup>®</sup> ist eine eingetragene Marke der ORACLE Corporation.

INFORMIX<sup>®</sup>-OnLine for SAP und Informix<sup>®</sup> Dynamic Server<sup>™</sup> sind eingetragene Marken der Informix Software Incorporated.

UNIX<sup>®</sup>, X/Open<sup>®</sup>, OSF/1<sup>®</sup> und Motif<sup>®</sup> sind eingetragene Marken der Open Group.

HTML, DHTML, XML, XHTML sind Marken oder eingetragene Marken des W3C<sup>®</sup>, World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc.

JAVASCRIPT<sup>®</sup> ist eine eingetragene Marke der Sun Microsystems, Inc., verwendet unter der Lizenz der von Netscape entwickelten und implementierten Technologie.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo und mySAP.com sind Marken oder eingetragene Marken der SAP AG in Deutschland und vielen anderen Ländern weltweit. Alle anderen Produkte sind Marken oder eingetragene Marken der jeweiligen Firmen.

## Symbole

Symbol	Bedeutung
	Achtung
	Beispiel
	Empfehlung
	Hinweis
	Syntax
	Tip

## Inhalt

<b>HTMLBusiness-Sprachbeschreibung .....</b>	<b>6</b>
Allgemeine HTMLBusiness-Regeln .....	7
Einbindung von HTMLBusiness in HTML .....	8
Kommentare in HTMLBusiness .....	10
Einbindung von mehreren HTMLBusiness-Anweisungen .....	11
Verweise auf R/3-Dynprofelder .....	12
Syntaxkonventionen .....	15
Verweise auf R/3-Arrays.....	16
Zeiger auf R/3-Felder verwenden.....	17
Auf R/3-Feldattribute zugreifen .....	18
.dim .....	19
.disabled .....	20
.exists.....	21
.label .....	22
.maxSize .....	23
.name.....	24
.type .....	25
.value .....	26
.visSize .....	27
<b>HTMLBusiness-Sprachelemente.....</b>	<b>28</b>
<b>HTMLBusiness-Schlüsselwörter.....</b>	<b>29</b>
<b>HTMLBusiness-Ausdrücke .....</b>	<b>30</b>
<b>HTMLBusiness-Funktionen .....</b>	<b>33</b>
Die Funktion archiveURL.....	35
Die Funktion assert.....	36
Die Funktion imageURL .....	37
Die Funktion includeFrame .....	39
Die Funktion mimeType .....	40
Die Funktion printf .....	43
Die Funktion strCat.....	45
Die Funktion strCmp.....	46
Die Funktion strCpy .....	47
Die Funktion striCmp .....	48
Die Funktion strLen .....	49
Die Funktion strLwr.....	50
Die Funktion strnCmp.....	51
Die Funktion strniCmp.....	52
Die Funktion strStr .....	53
Die Funktion strSub .....	54
Die Funktion strUpr.....	55
Die Funktion toLower.....	56
Die Funktion toUpper.....	57
Die Funktion wgateURL.....	58
Die Funktion write.....	62
Die Funktion writeEnc.....	63
<b>HTMLBusiness-Funktionsbeschreibung.....</b>	<b>64</b>

<b>HTMLBusiness -Anweisungen .....</b>	<b>71</b>
for-Anweisung.....	72
if-Anweisung .....	73
include-Anweisung .....	75
repeat-Anweisung.....	80
repeat .....	81
repeat with <reg> in <field>.....	82
repeat with <reg> from <expn> to <expn> .....	83
<b>Zusammenfassung der HTMLBusiness-Sprachgrammatik .....</b>	<b>84</b>
<b>Externe Faktoren .....</b>	<b>86</b>
<b>Sprachenunabhängigkeit.....</b>	<b>87</b>
Abrufen von Texten aus dem R/3-System .....	88
Verwendung von sprachenspezifischen Templates .....	89
Verwendung von Sprachressourcendateien .....	90
<b>Web-Browser-Unabhängigkeit .....</b>	<b>92</b>
Anforderungen, die einen Frame verändern .....	93
Anforderungen, die mehrere Frames verändern.....	94
<b>Lokales Speichern (Caching) auf dem Web-Browser .....</b>	<b>95</b>
<b>Einbindung von Java-Applets und ActiveX-Controls .....</b>	<b>97</b>
<b>Web-Eingabe dem R/3-System zuordnen.....</b>	<b>98</b>
<b>Syntax und Semantik .....</b>	<b>99</b>
<b>Mehrere Felder von einer HTML-Steuerung übergeben .....</b>	<b>102</b>
<b>&lt;textarea&gt;-Steuerungen verwenden.....</b>	<b>103</b>

## HTML<sup>Business</sup>-Sprachbeschreibung

### Einsatzmöglichkeiten

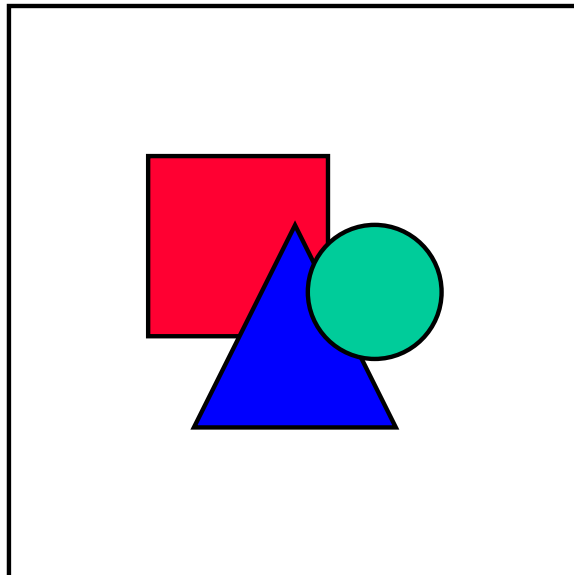
HTML<sup>Business</sup> ist eine SAP-spezifische Makrosprache, die vom Internet Transaction Server (ITS) benutzt wird, um Daten aus Transaktionsdynpros dynamisch in HTML-Templates von Internet-Anwendungskomponenten (IACs) einzumischen.

Startet ein Benutzer eine IAC vom Web-Browser aus, wird ein ITS-Service angestoßen, der einer R/3-Transaktion entspricht. Für jedes R/3-Transaktionsdynpro gibt es ein entsprechendes HTML-Template, das sowohl Standard-HTML-Quelltext als auch HTML<sup>Business</sup>-Anweisungen enthält. Zur Laufzeit werden die R/3-Dynprodaten an den ITS übergeben. Der HTML<sup>Business</sup>-Interpreter wertet die HTML<sup>Business</sup>-Anweisungen aus. Anschließend werden die Dynprodaten in die HTML-Templates eingemischt und dann im Web-Browser des Benutzers angezeigt.

### Funktionsumfang

Um HTML<sup>Business</sup> in Standard-HTML einzubinden, verwendet der ITS HTML-Befehle wie `<server>` `</server>` oder die entsprechenden invertierten Hochkommata.

Diese Befehle schließen HTML<sup>Business</sup>-Ausdrücke ein und machen dadurch mehrere Befehle oder Methoden auf HTML-Seiten überflüssig. Außerdem ist es so möglich, beliebige Autorenumgebungen zum Schreiben von HTML-Quelltext für Web-Seiten zu verwenden.



Diese Dokumentation setzt grundlegende Kenntnisse auf den Gebieten Standard-HTML, R/3-Dynproverarbeitung und ITS-Architektur voraus.

## Allgemeine HTML<sup>Business</sup>-Regeln

Dieser Abschnitt beschreibt die allgemeinen Regeln für die Verwendung von HTML<sup>Business</sup>.

### Allgemeine Syntax

[Einbindung von HTML-Business in HTML \[Seite 8\]](#)

[Kommentare in HTML-Business \[Seite 10\]](#)

[Einbindung von mehreren HTML-Business-Anweisungen \[Seite 11\]](#)

### Verweise auf Variablen

[Verweise auf R/3-Felder \[Seite 12\]](#)

[Verweise auf R/3-Arrays \[Seite 16\]](#)

[Zeiger auf R/3-Felder verwenden \[Seite 17\]](#)

[Auf R/3-Feldattribute zugreifen \[Seite 18\]](#)

## Einbindung von HTMLBusiness in HTML

### Einbindung von HTML<sup>Business</sup> in HTML

Um HTML<sup>Business</sup>-Ausdrücke in HTML-Standard Quelltext einzubinden, müssen sie immer in einen `<server> ... </server>`-Befehl oder den entsprechenden invertierten Hochkommata eingeschlossen werden.



Der folgende Quelltext führt dazu, daß das Dynprofeld `vbcom-kunde` in die Web-Seite eingemischt wird:

```
<h1> Order Status </h1>
<p> Customer Number: <server> VBCOM-KUNDE </server>
...
```

### Einbindung von HTML<sup>Business</sup> in Befehlen

HTML erlaubt keine Befehle in Befehlen. Sie können daher keine HTML<sup>Business</sup>-Befehle in einen HTML-Befehl einfügen.



Folgender Befehl ist also nicht möglich:

```
<a href="<server> screenURL </server>"> Link </a>
```

HTML-Editoren können derartige Befehle nicht verarbeiten. Deshalb bietet HTML<sup>Business</sup> zusätzlich das invertierte Hochkomma (```) zur Markierung von Ausdrücken. Sie können dieses Zeichen analog zum Befehl `<server> ... </server>` verwenden, aber auch innerhalb von Befehlen.



Obenstehender Hyperlink kann z.B. folgendermaßen geschrieben werden:

```
<a href="`screenURL`"> Link </a>
```

### Verwendung invertierter Hochkommata

Wenn Sie möchten, daß invertierte Hochkommata (```) explizit als uninterpretiertes Zeichen in einer HTML-Seite erscheinen, fügen Sie den entsprechenden Code `&#96` ein.



Die Zeile

```
<p> Hier folgt ein HTML-Business-Ausdruck: &#96VBCOM-KUNDE&#96
</p>
```

wird z.B. uninterpretiert in der Web-Seite belassen und sieht im Web-Browser folgendermaßen aus:

```
Hier folgt ein HTML-Business-Ausdruck: `VBCOM-KUNDE`
```



## Kommentare in HTMLBusiness

## Kommentare in HTML<sup>Business</sup>

Sie können HTML-Kommentare verwenden, um HTML<sup>Business</sup>-Ausdrücke auszukommentieren:

- Ein HTML-Kommentar wird durch "<!--" eingeleitet und endet mit "-->". (Ein Kommentar darf nicht nur mit ">" beendet werden.). HTML<sup>Business</sup>-Ausdrücke, die innerhalb eines HTML-Kommentars stehen, werden uninterpretiert in der HTML-Seite ausgegeben.

Die Zeile

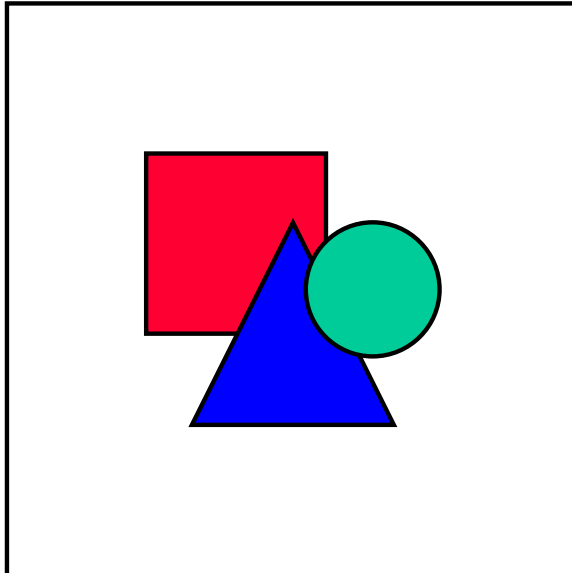
```
<!-- `VBCOM-KUNDE` -->
```

wird also nicht interpretiert.

- Sie können auch innerhalb eines HTML<sup>Business</sup>-Ausdruckes Teile auskommentieren. Dies erfolgt über die HTML-Kommentarsyntax.

Folgende Zeile enthält einen gültigen HTML<sup>Business</sup>-Ausdruck:

```
`repeat with j from 1 to <!--stepLoop.dim --> 10`
```



HTML-Kommentare müssen mit "-->" abgeschlossen werden. Das Zeichen ">" allein beendet den Kommentar nicht.

## Einbindung von mehreren HTML<sup>Business</sup>-Anweisungen

Sie können mehrere HTML<sup>Business</sup>-Anweisungen

- separat in den Serverbefehl oder invertierte Hochkommata einschließen
- in einen einzigen Serverbefehl oder zwischen einem einzigen Paar invertierter Hochkommata ( ) einschließen

Trennen Sie im letzten Fall (wie in JavaScript oder C) die HTML<sup>Business</sup>-Anweisungen mit Semikolons. Folgende Schreibweise ist richtig:

```
<p> `for (j=0; j <= array.dim; j++)
    j; write("="); array[j]; write (" , ")
end`
</p>
```

### Begrenzer in Hochkommata zwischen Anweisungen

Zwei Anweisungen, die beide getrennt in invertierte Hochkommata ( ) eingeschlossen sind, mußten vor ITS-Version 2.2 durch mindestens ein Zeichen getrennt werden.

```
`if (1)` `write("Hello world!")` `end` (Beachten Sie die
Leerzeichen!)
```

(Das galt auch für die entsprechenden Befehle </server> und <server> .)

Ab ITS 2.2. können Sie nun schreiben:

```
`if (1)` `write("Hello world")` `end`.
```

### Begrenzer zwischen aufeinanderfolgenden HTML<sup>Business</sup>-Anweisungen

Vor ITS-Version 2.2 mußte zwischen aufeinanderfolgenden HTML<sup>Business</sup>-Anweisungen entweder ein Semikolon oder ein mindestens ein einzelner Begrenzer stehen.

Ab ITS 2.2. interpretiert der HTML<sup>Business</sup>-Interpreter diese Anweisungen richtig. Es sind weder Semikolons noch andere Begrenzer erforderlich. Ein Semikolon ist auch nicht erforderlich:

- vor den Schlüsselwörtern **end**, **else**, **elsif** und **elseif**
- nach Anweisungen, die einen Block einleiten

Beispiele sind **for** (...), **if** (...), **elsif** (...), **elseif** (...) und **repeat** ( )

## Verweise auf R/3-Dynprofelder

## Verweise auf R/3-Dynprofelder

Hauptaufgabe von HTML<sup>Business</sup> ist es, dem Internet Transaction Server (ITS) den Datentransfer zwischen R/3-Dynpros und den HTML-Templates zu ermöglichen.

Es gibt zwei R/3-Feldtypen, die dem ITS zur Laufzeit bekannt sind:

- von einem R/3-Dynpro übergebene Felder
  - Nur tatsächliche Dynprofelder sind für den ITS sichtbar. Felder, die dem ABAP-Modulpool bekannt sind, aber auf keinem Dynpro erscheinen, sind für den ITS nicht sichtbar.
- über den RFC-Kanal übergebene Felder
  - Diese Felder wurden vom ABAP-Programm mittels der ITS-Macros [FIELD\\_SET \[Extern\]](#) oder [FIELD\\_TRANSPORT \[Extern\]](#) übertragen.

### Feldverweise erstellen

Die Feldersetzung ist die einfachste Form einer HTML<sup>Business</sup>-Anweisung. Bis auf reservierte ABAP-Schlüsselwörter wird jeder HTML<sup>Business</sup>-Ausdruck wie ein R/3-Feldname bezeichnet.

Um z.B. die Felddaten eines R/3-Dynpros in Ihrer HTML-Seite anzuzeigen, schließen Sie den R/3-Feldnamen einfach in `<server>`-Befehle (oder invertierte Hochkommata) ein.

Zwischen den Dynprofeldern und den HTML-Feldern besteht normalerweise eine 1:1-Beziehung. Dies gilt sowohl für das Lesen vom und Schreiben auf das Dynpro. (Große Textbereiche bilden eine Ausnahme.)

- Wenn der ITS eine Übereinstimmung für den Namen findet, überträgt er den Inhalt des R/3-Feldes in die HTML-Seite.
- Wenn keine Übereinstimmung gefunden wird, ersetzt der ITS die Variable mit einem Leerstring.

Feldinhalte können auch in einfachen Ausdrücken verwendet werden.

### Feldbezeichnersyntax

Die folgende Tabelle zeigt die vorgeschriebene Syntax für die Erstellung von Verweisen auf R/3-Dynprofelder in HTML<sup>Business</sup>. Die Attribute liefern Ihnen zusätzliche Informationen über das Feld.

Name	Syntax
Field	<code>{ ^ } identifier [ [ expression ] ] [. attribute ]</code>
Attribute	<code>label  </code>
	<code>visSize  </code>
	<code>maxSize  </code>
	<code>dim  </code>
	<code>disabled  </code>
	<code>name  </code>

Verweise auf R/3-Dynprofelder

	value
--	-------

Bezeichner sollten in HTML<sup>Business</sup> den folgenden Konventionen üblicher Programmiersprachen wie C oder JavaScript folgen:

Feldbezeichnerkomponente	Mögliche Zeichen	Werte
Erstes Zeichen	Ein Buchstabe	a...z , A...Z
	Ein Unterstrich	_
	Eine Tilde	~
Nachfolgende Zeichen	Buchstaben	a...z , A...Z
	Ein Unterstrich	_
	Eine Tilde	~
	Eine Ziffer	0...9
	Ein Bindestrich	-



Folgende Bezeichner sind richtig:

```
nCustomers
_foo_bar
~frame
date1
ordered_items
vbcom-kunde
```

Sie müssen den ganzen Feldbezeichner in einfache Hochkommata einschließen,

- wenn Sie andere Zeichen verwenden
- wenn Sie einen Bezeichner mit demselben Namen als Schlüsselwort verwenden



```
'*vbcom-kunde'
'$@#identifizier()***'
'from'
```

Bezeichner wie "VBCOM-KUNDE" benötigen aufgrund des Bindestrichs keine einfachen Hochkommata. Statt "VBCOM-KUNDE" können Sie einfach `VBCOM-KUNDE` schreiben.

Wenn Sie jedoch arithmetische Ausdrücke mit Minuszeichen verwenden möchten, muß vor und hinter dem Minuszeichen mindestens ein Leerzeichen stehen:

```
` vbcom-kunde `
```

---

**Verweise auf R/3-Dynprofelder****Die Anzahl der übergebenen Zeichen beschränken**

Sie können die Anzahl der Zeichen pro Feld beschränken, die von der HTML-Seite auf das Dynpro übertragen werden. Geben Sie hierzu nach dem Feldnamen ein Komma ein, und geben Sie dann die gewünschte Anzahl der Zeichen an. Sie müssen die Länge zwischen dem Feldnamen und dem Index angeben.

Wenn Sie auf diese Art die Anzahl der Zeichen beschränken, gehen zusätzliche Zeichen verloren. Dies ist nicht der Fall, wenn das Feld ein Array ist und wenn Sie keinen expliziten Index mit [] angegeben haben. Die Zeichen, die über die von Ihnen angegebene Grenze hinausgehen, werden dann in eine neue Zeile der Step-Loop aufgenommen. Dies ist für die Verarbeitung großer Textbereiche erforderlich.

## Syntaxkonventionen

In den Tabellen wird folgende Syntax verwendet:

- Optionale Ableitungen sind in eckigen Klammern ([]) aufgeführt.
- Geschweifte Klammern ({}), beschreiben die beliebig häufige Wiederholung des eingeschlossenen Ausdrucks.
- Ein senkrechter Strich (|) verkettet alternative Ableitungen ("OR").
- Runde Klammern (()) fassen Ableitungen logisch zusammen, falls keine Eindeutigkeit gewährleistet ist.

---

**Verweise auf R/3-Arrays**

## Verweise auf R/3-Arrays

Zusätzlich zu einfachen Feldern verarbeitet der Internet Transaction Server (ITS) auch Arrays. Im R/3-System werden diese Arrays entweder als Step-Loops auf dem Dynpro erstellt oder über RFC vom Dynpro an den ITS gesendet. Auf die einzelnen Elemente in einem Array können Sie durch die Angabe eines Index nach dem Feldnamen zugreifen. Der Index ist in eckige Klammern eingeschlossen. Diese Zuordnung gilt für Felder in einer Step-Loop in beide Richtungen. Wenn Daten über RFC übertragen werden, sind diese schreibgeschützt.

Wenn die Klammern hinter dem Feldnamen keinen Wert enthalten, werden die vom HTML-Dokument in das R/3-System übertragenen Daten zu den bestehenden Feldern hinzugefügt. Das Array darf nicht länger als die entsprechende Step-Loop auf dem Dynpro sein, da dies sonst zu einem Laufzeitfehler führt.

## Zeiger auf R/3-Felder verwenden

Mit Hilfe des Zeichens "^" können Bezeichner als Zeiger eingesetzt werden.

Hat "Feldname" z.B. den Wert "vbcom-kunde", so führt die folgende Anweisung dazu, daß der Inhalt des Feldes "vbcom-kunde" ausgegeben wird:

```
<p> Der Wert des Feldes 'Feldname` ist '^Feldname` </p>
```

Für diese Anweisung erzeugt der ITS in der HTML-Seite folgendes:

```
<p> Der Wert des Feldes VBCOM-KUNDE ist 3100 </p>
```

Mehrfache Indirektion ist möglich, z.B.:

```
^^kaum_sinnvoll[j]
```

## Auf R/3-Feldattribute zugreifen

## Auf R/3-Feldattribute zugreifen

HTML<sup>Business</sup> stellt mehrere Attribute bereit, um die Attribute eines Dynprofeldes im R/3-System zu bestimmen:

Attribut	Bedeutung
<a href="#">.dim [Seite 19]</a>	Anzahl der Werte in einem Feld (= Dimension)
<a href="#">.disabled [Seite 20]</a>	Das Feld ist auf dem Dynpro eingabebereit oder nicht.
<a href="#">.exists [Seite 21]</a>	Feld existiert auf dem Dynpro oder nicht.
<a href="#">.label [Seite 22]</a>	Beschreibender Text zu dem gleichnamigen Eingabefeld
<a href="#">.maxSize [Seite 23]</a>	Maximale Anzahl Zeichen, die in das Feld passen
<a href="#">.name [Seite 24]</a>	Vollständiger Name und Index eines Feldes
<a href="#">.type [Seite 25]</a>	Feldtyp
<a href="#">.value [Seite 26]</a>	Aktueller Wert des Feldes
<a href="#">.visSize [Seite 27]</a>	Maximale Anzahl Zeichen, die das Feld auf dem Dynpro anzeigt

## .dim

Mit dem Attribut `dim` ermitteln Sie, wie viele Werte ein Feld besitzt (Dimension eines mehrwertigen Feldes, Array-Dimension). Von Bedeutung ist dies bei Step-Loops, die als mehrwertige Felder abgebildet werden.

Angenommen, der Step-Loop besitzt eine Spalte, die aus dem Feld `VBCOM-WGKEY` gebildet wird:

- Hat diese Spalte 20 Zeilen, so liefert `VBCOM-WGKEY.dim` den Wert 20.
- Bei Spalten mit nur einer Zeile (einwertigen Feldern) wird der Wert 1 geliefert.
- Ist ein Feld nicht definiert, liefert `dim` den Wert 0.

Auf mehrwertige Felder können Sie über die Angabe eines zusätzlichen Indexes zugreifen. Dieser wird in eckigen Klammern angefügt und ist definiert über die Werte 1 bis `dim`.



Der folgende Quelltext verarbeitet einen Array, der auf dem Feld `vbcom-kunde` basiert:

```
`repeat with I from 0 to vbcom-kunde.dim`  
  <input type=text name="`vbcom-kunde[I].name`" value="`vbcom-  
kunde[I].value`">  
`end`
```

Wenn Sie auf den zweiten Wert des Feldes `vbcom-kunde` zugreifen möchten, müssen Sie folgenden Ausdruck verwenden:

Kunde Nr.2: ``vbcom-kunde[2]``

Der Wert von `.dim` muß innerhalb der definierten Array-Dimension liegen:

- Wenn Sie einen negativen Index eingeben, wird ein Laufzeitfehler ausgegeben.
- Wenn Sie Indizes größer als der durch `dim` definierte Wert eingeben, führt dies zur Ausgabe eines Leerstrings.

**.disabled**

## **.disabled**

Mit dem Attribut `.disabled` ermitteln Sie, ob ein Feld eingabebereit ist oder nur der Anzeige dient.



Mit dem folgenden Quelltext stellen Sie fest, ob das Feld `vbcom-kunde` auf dem Dynpro existiert und ob es eingabebereit ist:

```
`if ( vbcom-kunde.exists )`  
  `if ( vbcom-kunde.disabled )`  
    <! Value only -->  
    `vbcom-kunde`  
  `else`  
    <! ready for input -->  
    <input type="text" name="vbcom-kunde" value "`vbcom-  
kunde`">  
  `endif`  
`endif`
```

## .exists

Mit dem Attribut `.exists` ermitteln sie, ob ein Feld auf dem aktuellen Dynpro existiert.



Mit dem folgenden Quelltext stellen Sie fest, ob das Feld `vbcom-kunde` auf dem Dynpro existiert und ob es eingabebereit ist:

```
`if ( vbcom-kunde.exists )`  
  `if ( vbcom-kunde.disabled )`  
    <! Value only -->  
    `vbcom-kunde`  
  `else`  
    <! ready for input -->  
    <input type="text" name="vbcom-kunde" value "`vbcom-  
kunde`">  
  `endif`  
`endif`
```

## .label

### .label

Mit dem Attribut `.label` ermitteln Sie die Textbeschreibung eines Eingabefeldes:



Mit dem folgenden Quelltext legen Sie die Textbeschreibung des Feldes `vbcom-kunde` fest:

```
`vbcom-kunde.label`: <input type=text name="vbcom-kunde">
```

Ist zu einem Eingabefeld auf dem R/3-Dynpro eine Textbeschreibung vorhanden, die den gleichen Feldnamen besitzt, so können Sie auf diese Beschreibung mit dem Attribut `label` zugreifen:

```
<p> `vbcom-kunde.label`: <input type=text name="vbcom-kunde">
```

Mit diesem Attribut (zusätzlich zu den Ressourcendateien) können Sie Sprachenunabhängigkeit in HTML-Seiten realisieren. Wenn Sie das Attribut `label` wie hier beschrieben verwenden, bestimmt die Anmeldesprache am R/3-System die Texte in der HTML-Seite.

Weitere Informationen finden Sie unter [Verwendung von Sprachressourcendateien \[Seite 90\]](#).

## .maxSize

Mit dem Attribut `.maxSize` ermitteln Sie die maximale auf dem Dynpro sichtbare Länge eines Feldes:



Mit dem folgenden Quelltext können Sie die maximale Eingabelänge des Feldes `vbcom-kunde` abfragen:

```
<p> Bitte geben Sie Ihre Kundennummer ein  
  <input type=text name="vbcom-kunde"  
        maxSize=`vbcom-kunde.maxSize` </p>
```

.name

## .name

Mit dem Attribut `.name` können Sie in Verbindung mit dem Attribut `.value` gültige Feldnamen konstruieren.

Das Attribut `.name` gibt den vollständigen Namen und den Index des jeweiligen Felds aus.



Um beispielsweise einen `<input>`-Befehl für das mehrwertige Feld `vbcom-kunde` zu erstellen, können Sie schreiben:

```
`repeat with I from 0 to vbcom-kunde.dim`  
  <input type=text name="vbcom-kunde[`I`]" value="`vbcom-  
kunde[I]`">  
`end`
```

Dies ergibt folgende Ausgabe:

```
<input type=text name="vbcom-kunde[1]" value="4711">  
<input type=text name="vbcom-kunde[2]" value="8523">  
<input type=text name="vbcom-kunde[3]" value="1234">
```

Wie das vorherige Beispiel zeigt, kann die Reihe der invertierten Hochkommata und Anführungszeichen etwas verwirrend sein. Statt dessen können Sie auch die Attribute `.name` und `.value` einsetzen:

```
`repeat with I from 0 to vbcom-kunde.dim`  
  <input type=text name="`vbcom-kunde[I].name`" value="`vbcom-  
kunde[I].value`">  
`end`
```

Die Ausgabe ist dieselbe wie im obigen Beispiel, also:

```
<input type=text name="vbcom-kunde[1]" value="4711">  
<input type=text name="vbcom-kunde[2]" value="8523">  
<input type=text name="vbcom-kunde[3]" value="1234">
```

## .type

Mit dem Attribut `.type` können Sie den Typ eines Dynprofeldes festlegen.



Mit dem folgenden Quelltext stellen Sie fest, ob ein Feld ein Listenfeld (Combo box) ist:

```
`if <field>.type==ComboBox`  
  (action)  
`endif`
```

In folgenden Tabelle sehen Sie die möglichen Werte für das Attribut `.type`:

Feldattribut	Mögliche Werte
<code>.type</code>	TableView
	TableColumn
	TableSelector
	TableColTitle
	Label
	Frame
	PushButton
	RadioButton
	CheckButton
	Password
	ComboBox
	Edit
	Unknown

.value

## .value

Mit dem Attribut `.value` können Sie zusammen mit dem Attribut `.name` gültige Feldnamen konstruieren.

Das Attribut `.value` gibt den Wert eines Feldes aus.



Um einen `<input>`-Befehl für das mehrwertige Feld `vbcom-kunde` zu erstellen, können Sie schreiben:

```
`repeat with I from 0 to vbcom-kunde.dim`  
  <input type=text name="vbcom-kunde[`I`]" value="`vbcom-  
kunde[I]`">  
`end`
```

Dies ergibt folgende Ausgabe:

```
<input type=text name="vbcom-kunde[1]" value="4711">  
<input type=text name="vbcom-kunde[2]" value="8523">  
<input type=text name="vbcom-kunde[3]" value="1234">
```

Wie das vorherige Beispiel zeigt, kann die Reihe der invertierten Hochkommata und Anführungszeichen etwas verwirrend sein. Statt dessen können Sie auch die Attribute `.name` und `.value` einsetzen:

```
`repeat with I from 0 to vbcom-kunde.dim`  
  <input type=text name="`vbcom-kunde[I].name`" value="`vbcom-  
kunde[I].value`">  
`end`
```

Die Ausgabe ist dieselbe wie im obigen Beispiel, also:

```
<input type=text name="vbcom-kunde[1]" value="4711">  
<input type=text name="vbcom-kunde[2]" value="8523">  
<input type=text name="vbcom-kunde[3]" value="1234">
```

## .visSize

Mit dem Attribut `.visSize` können Sie die maximal auf dem Dynpro sichtbare Länge eines Feldes abfragen:



Mit dem folgenden Quelltext können Sie die maximal sichtbare Länge des Feldes `vbcom-kunde` ermitteln:

```
<p> Bitte geben Sie Ihre Kundennummer ein  
  <input type=text name="vbcom-kunde"  
    size=`vbcom-kunde.visSize` </p>
```

## HTML<sup>Business</sup>-Sprachelemente

Die folgenden Abschnitte geben Ihnen eine Beschreibung der Bestandteile von HTML<sup>Business</sup>:

[HTML-Business-Schlüsselwörter \[Seite 29\]](#)

[HTML-Business-Ausdrücke \[Seite 30\]](#)

[HTML-Business-Funktionen \[Seite 33\]](#)

[HTML-Business-Anweisungen \[Seite 71\]](#)

[Zusammenfassung der HTML-Business-Sprachgrammatik \[Seite 84\]](#)

## HTML<sup>Business</sup>-Schlüsselwörter

Folgende Schlüsselwörter sind reserviert in HTML<sup>Business</sup>:

archiveURL	assert	by	declare
define	else	elseif	elsif
end	for	from	if
imageURL	in	include	mimeURL
repeat	secure	times	to
wgateURL	with	write	writeEnc

Diese Schlüsselwörter können nicht als Bezeichner verwendet werden. Wenn Sie einen gleichnamigen Bezeichner einsetzen, muß dieser in normalen einfachen Hochkommata ('') eingeschlossen werden.

Zum Beispiel: `repeat with i from 'from' to 'to'`

## HTMLBusiness-Ausdrücke

**HTML<sup>Business</sup>-Ausdrücke**

HTML<sup>Business</sup>-Ausdrücke können in der in C, C++ oder JavaScript üblichen Weise geschrieben werden.

Ab ITS 2.2 gibt es neue Regeln für:

- Operatorgewichtung
- Operatorauswertungsreihenfolge

**Operatorgewichtung**

HTML<sup>Business</sup> hatte eine Gewichtung, die oftmals von dem abwich, was C-, C++- oder Java-Programmierer erwarten. Der Operator `||` hatte beispielsweise Vorrang vor dem Operator `==`. ITS-Entwickler mußten daher unnötigerweise oft Klammern setzen, um den Vorrang eindeutig darzustellen:

Anstatt beispielsweise zu schreiben:

```
`if (a==1 && b==2)` ,
```

mußte man

```
`if ((a==1) && (b==2))` .
```

schreiben. Ansonsten hätte der HTML<sup>Business</sup>-Interpreter den Ausdruck als

```
`if (a==(1 && b)== 2)`
```

interpretiert. Die Gewichtung wurde nun C, C++ und Java angepaßt.

Folgende, in fallender Gewichtung aufgelistete Operatoren stehen in HTML<sup>Business</sup> zur Verfügung:

Operator	Gewichtung
++, --	1
*, /, %	2
+, -, &	3
==, !=, >, <, >=, <=	4
&&,	5

Wird eine andere Gewichtung verlangt, müssen die Ausdrücke in Klammern gestellt werden. Operatoren mit gleicher Gewichtung werden von links nach rechts ausgewertet.

**Operatorauswertungsreihenfolge (Version 2.2)**

Vor ITS Version 2.2 wich auch die Auswertungsreihenfolge von der bei C, C++ oder Java in einigen (nicht allen) Fällen ab. Ausdrücke wurden von rechts nach links ausgewertet (anstatt von links nach rechts).

Aus Konsistenzgründen wurde dies mit Version 2.2 angepaßt. Der Ausdruck

```
`8/2*4`
```

wird nun als

``(8/2)*4` (==16)`

anstatt

``8/(2*4)` (==1).`

ausgewertet.

### Syntax der Ausdrücke

Die folgende Grammatik gibt genau an, welche Formen der Ausdrücke erlaubt sind:

- Operatoren identischer Gewichtung sind gruppiert und werden von links nach rechts ausgewertet.
- Wenn die Auswertungsreihenfolge der Operatoren durchbrochen werden soll, ist Klammerung erforderlich.

Ausdruck	Syntax
<b>expression</b>	simpleexpr [compop simpleexpr]
<b>simpleexpr</b>	term { addopr simpleexpr}
term	factor { mulopr factor}
factor	(!   ++   --) factor ( expression )   function call   assignment   lvalue [++   --]   constant
<b>function call</b>	internalfn ( argument {, argument})   externalfn ( expression {, expression})
<b>internalfn</b>	write   writeEnc   wgateURL   archiveURL   imageURL   <b>mimeURL</b>   assert
<b>mulopr</b>	* / % &&
<b>addopr</b>	+ - &
<b>compop</b>	==   !=   >   <   >=   <=

Informationen über die in der Tabelle benutzte Schreibweise finden Sie in den [Syntaxkonventionen \[Seite 15\]](#).



Folgende Ausdrücke sind richtig:

`vbcom-kunde`  
`nCustomers % 10`

## HTMLBusiness-Ausdrücke

```
!fExists
a > b*2+1
name != "Walt"&" "&"Whitman"
(x -y) * (a+b) & " US$"
cond1 && (cond2 || cond3) && cond4
```

## HTML<sup>Business</sup>-Funktionen

HTML<sup>Business</sup> stellt für einige Anwendungsfälle nützliche Standardfunktionen zur Verfügung, die HTML-Fragmente erzeugen. Diese Funktionen können nur innerhalb des HTML<sup>Business</sup>-Interpreters entwickelt werden, so daß der Bestand der Makros für die HTML-Generierung nicht erweitert werden kann.

Um Ihnen mehr Spielraum zu bieten, können Sie Ihre eigenen Funktionen schreiben, jedoch sind dazu Kenntnisse der darunterliegenden Konzepte erforderlich.

Es gibt folgende Standardfunktionen:

[Die Funktion archiveURL \[Seite 35\]](#)

[Die Funktion assert \[Seite 36\]](#)

[Die Funktion imageURL \[Seite 37\]](#)

[Die Funktion mimeURL \[Seite 40\]](#)

[Die Funktion printf \[Seite 43\]](#)

[Die Funktion strCat \[Seite 45\]](#)

[Die Funktion strCmp \[Seite 46\]](#)

[Die Funktion strCpy \[Seite 47\]](#)

[Die Funktion striCmp \[Seite 48\]](#)

[Die Funktion strLen \[Seite 49\]](#)

[Die Funktion strLwr \[Seite 50\]](#)

[Die Funktion strnCmp \[Seite 51\]](#)

[Die Funktion strnCmp \[Seite 52\]](#)

[Die Funktion strStr \[Seite 53\]](#)

[Die Funktion strSub \[Seite 54\]](#)

[Die Funktion strUpr \[Seite 55\]](#)

[Die Funktion toLower \[Seite 56\]](#)

[Die Funktion toUpper \[Seite 57\]](#)

[Die Funktion wgateURL \[Seite 58\]](#)

[Die Funktion write \[Seite 62\]](#)

[Die Funktion writeEnc \[Seite 63\]](#)

[HTMLBusiness-Funktionen \[Seite 64\]](#)

Weitere Informationen zum Schreiben Ihrer eigenen HTML<sup>Business</sup>-Funktionen finden Sie unter:

[HTMLBusiness-Funktionsbeschreibung \[Seite 64\]](#)



## Die Funktion archiveURL

### Einsatzmöglichkeiten

Greift auf Funktionen im iXOS-Archiv zu.

### Syntax

```
archiveURL(command, archiveID=<expression>, docID=expression)
```

### Beschreibung

Analog zu wgateURL kann die Funktion archiveURL verwendet werden, um auf das iXOS-Archivsystem zuzugreifen. Sie geben die entsprechende iXOS-Funktion über die Parameter command, archiveID und docID an. Weitere Informationen über das iXOS-Archiv finden Sie in der entsprechenden Produktinformation.

Die einzelnen Parameterbeschreibungen für die von Ihnen aufgerufene Funktion entnehmen Sie der Produktdokumentation zur zugehörigen Archiv-Web-DII. Die relevante Archiv-Web-DLL muß auf dem Web-Server korrekt installiert sein.

Die von archiveURL verwendete URL wird in der globalen Servicebeschreibung global.svc unter dem Parameter ~URLarchive festgelegt.

## Die Funktion assert

# Die Funktion assert

## Einsatzmöglichkeiten

Erzeugt eine Standardmeldung, wenn bei einem HTML-Feld ein Fehler auftritt.

## Syntax

```
assert(<field name>)
```

## Beschreibung

Mit der Funktion "assert()" können generisch Fehlermeldungen aus dem R/3-System in Webseiten abgebildet werden.

Tritt im R/3-System bei der Prüfung von Benutzereingaben ein Fehler auf (ungültige Eingabe), so wird die aktuelle Cursorposition verwendet, um festzustellen, welches Feld den Fehler verursacht hat. Auf der ITS-Seite steht im Systemfeld "~MessageLine" dann die entsprechende Fehlermeldung.

Wenn Sie in einem HTML-Template für ein Feld die Funktion assert verwenden und dieses Feld in R/3 einen Fehler erzeugt, ersetzt der ITS Ihre assert-Anweisung durch die Standardfehlermeldung. Mit der Funktion assert können Sie eine Meldung (oder ein Bild) neben einem Feld anzeigen, immer wenn dieses Feld Fehlerdaten enthält.

Der ITS ersetzt assert normalerweise durch den ~ErrorMarker-Wert aus der Servicebeschreibung global.srvc. Dieser Parameter kann eine HTML-Anweisung enthalten (z.B. einen Hyperlink zu einem Bild). Der Vorschlagswert für ~ErrorMarker ist: "@@ Error @@".



```
<form ...>
...
Materialnummer <input type="text" name="matnr">`assert(matnr)`
Menge      <input type="text"
              name="quantity">`assert(quantity)`
      <p>`~MessageLine`</p>
</form>
```

Da nur eine direkte Ersetzung erfolgt, werden HTML<sup>Business</sup>-Anweisungen in diesem Parameter nicht analysiert.

## Die Funktion imageURL

### Einsatzmöglichkeiten

Erzeugt auf der Basis von ITS-Sprache und Theme dynamische Links zu Bildern.

### Syntax

```
imageURL(~type=<expression>, ~name=<expression> {,~theme=<expression>},
{,~language=<expression>})
```

### Parameter

Parameter	Bedeutung
~name	Name des Bildes
~type	Typ des Bildes
~language	Aktuelle Anmeldesprache
~theme	Theme

### Beschreibung



Ab ITS-Version 1.1 (R/3-Version 3.1H) wird die Funktion imageURL nicht mehr verwendet. Verwenden Sie für neuere ITS- und R/3-Versionen an Stelle von imageURL [die Funktion mimeTypeURL \[Seite 40\]](#). Die Funktion imageURL wird jedoch weiterhin unterstützt, so daß Sie Ihre existierenden HTML-Templates weiterhin verwenden können.

Über die Funktion **imageURL** ist es möglich, auf Bilder zuzugreifen. Diese Bilder sind nicht im ITS-Verzeichnis, sondern im Verzeichnis des HTTP-Servers abgelegt.

Wenn Sie Links zu Bildern angeben, ist es nicht ausreichend, eine einfache URL anzugeben. Die URL muß Parameter mit Angaben zum Typ des Bildes sowie zu dem entsprechenden Service und der entsprechenden Theme enthalten. Über die Funktion **imageURL** können Sie auch sprachenabhängig auf Bilder zugreifen. Hiermit wird die Verwendung mehrsprachiger Bilder unterstützt.

Geben Sie bis auf den Namen alle Parameter bezüglich des Unterverzeichnisses in der ITS-[Verzeichnisstruktur \[Extern\]](#) an. Die erzeugte URL hängt von diesen Werten und dem in `global.srv` definierten Parameter `~URLimage` ab. Die Funktion **imageURL** erzeugt die grundlegende URL wie folgt:

```
<~URLimage>/<~type>/<~language>/<~theme>/<~name>
```

oder spezifisch:

```
/<HTTP server root directory>/SAP/ITS/GRAPHICS/<type>/<language>/<theme>/<name>
```

Wenn die Parameter `~Language` und `~Theme` nicht angegeben werden, werden die Werte aus der Servicebeschreibung oder die aktuelle Anmeldesprache verwendet. Die den Parametern zugewiesenen Werte werden ohne Überprüfung vom ITS geholt, um die URL zu erstellen. So

## Die Funktion imageURL

können Sie weitere (mit dem Bilderverzeichnis verknüpfte) Unterverzeichnisse anlegen, die über die Funktion `imageURL` aufgerufen werden können.

Werte, die nicht definiert oder leer sind, werden aus der Verzeichnisstruktur ausgenommen.



Einige Beispiele (wobei `~language "EN"` ist und `~theme` nicht definiert ist):

1. ``

Ergibt:

```

```

2. ``

Ergibt:

```

```

3. ``

Ergibt:

```

```

4. ``

Ergibt:

```

```

## Die Funktion includeFrame

### Einsatzmöglichkeiten

Setzt Anwendungen mit mehreren Frames in eine einzelne Seite um.

### Syntax

```
includeFrame(~framename=<frame name>)
```

### Parameter

Parameter	Bedeutung
~framename	Name des Frames

### Beschreibung

Bei Anwendungen mit mehreren Frames können Sie diese Funktion verwenden, um die Frames in eine einzelne Seite umzusetzen.

Beispielsweise könnten Sie mit dem Befehl `<frameset> ... </frameset>` eine einfache Anwendung mit zwei Frames wie folgt implementieren:

```
<frameset rows=80,* FRAMEBORDER=0 BORDER=0 FRAMESPACING=0>
<frame name="FRAME_1" SRC="`wgateURL(~FrameName="FRAME_1")`"
scrolling="no">
<frame name="FRAME_2" SRC="`wgateURL(~FrameName="FRAME_2")`">
</frameset>
```

Um die aus mehreren Frames bestehende Anwendung in eine HTML-Seite mit einem Frame umzusetzen, entfernen Sie die Struktur `<frameset> ... </frameset>` und verwenden stattdessen die Funktion `includeFrame` mit dem Befehl `<table> ... </table>`:

```
<table>
  <TR><TD>`includeFrame(~framename="FRAME_1")`</TD></TR>
  <TR><TD>`includeFrame(~framename="FRAME_2")`</TD></TR>
</table>
```

Anschließend werden **FRAME\_1** und **FRAME\_2** zwar noch in 2 Zeilen, aber als einzelne Seite und nicht als zwei getrennte Frames angezeigt.

## Die Funktion mimeTypeURL

## Die Funktion mimeTypeURL

### Einsatzmöglichkeiten

Erzeugt auf der Basis von ITS-Sprache und Theme dynamische Links zu Bildern.

### Syntax

```
mimeTypeURL ( [ [~service]=expression, ] [ [~theme]=expression, ] [
  [~language]=expression, ] [~secure=expression,] [~name]=expression)
```

### Parameter

Parameter	Bedeutung
~service	Gibt den Namen des Service an; bei Auslassung wird der Name des aktuellen Service verwendet. Besondere Servicenamen sind "global", welcher für alle von mehreren Servern verwendete Dateien benutzt werden sollte, und "system", welcher für die ITS-Systemnachrichten verwendet wird.
~theme	Gibt die zu verwendende Theme an; bei Auslassung wird die aktuelle Theme verwendet (in der Servicedatei definiert). Durch Angabe von ~theme="" wird der gesamte Theme-Teil des erstellten Pfads weggelassen.
~language	Gibt die zu verwendende Sprache an; bei Auslassung wird die aktuelle Sprache verwendet (in der Servicedatei oder über die Loginseite definiert). Sie können den Sprachteil in dem erstellten Pfad auslassen, indem Sie ~language="" angeben.
~secure	Gibt an, ob eine relative oder absolute URL-Adresse verwendet werden soll: <ul style="list-style-type: none"> <li>• <b>~secure</b> wird nicht verwendet: mimeTypeURL() legt eine relative URL wie in Versionen vor ITS 2.2 an.</li> <li>• <b>~secure="on"</b>: mimeTypeURL() legt eine absolute URL an und verwendet den Protokolltyp HTTPS. Verwenden Sie den Wert ON, wenn Sie von einer (unsicheren) HTTP-Kommunikation zu einer (sicheren) HTTPS-Kommunikation während einer laufenden Verbindung wechseln möchten.</li> <li>• <b>~secure="off"</b>: mimeTypeURL() legt eine absolute URL an und verwendet den Protokolltyp HTTP. Verwenden Sie den Wert OFF, wenn Sie von einer (sicheren) HTTPS-Kommunikation zu einer (unsicheren) HTTP-Kommunikation während einer laufenden Verbindung wechseln möchten.</li> </ul> Beispiele für die Verwendung dieses Parameters finden Sie unter <a href="#">Die Funktion wgateURL [Seite 58]</a> .
~name	Gibt den Namen (und optionale Unterverzeichnisse) der Datei an, auf die verwiesen werden soll. Beispiele hierfür sind <b>ok.gif</b> oder <b>buttons/roundones/cancel.gif</b> .

## Die Funktion mimeURL

Parameternamen können weggelassen werden, was kürzere, aber weniger verständliche Funktionsaufrufe ergibt. Wenn Parameternamen ausgelassen werden, ist die Standardreihenfolge: ~service, ~theme, ~language, ~secure, ~name

Alle Parameter außer ~name sind optional und werden von dem aktuellen Session-Kontext abgeleitet, sofern sie nicht bereits als Argument definiert sind. Geben Sie außer dem Dateinamen alle anderen Parameter bezüglich des Unterverzeichnisses in der [Verzeichnisstruktur \[Extern\]](#) an. Die Funktion mimeURL() konstruiert den URL-Pfad mit folgender Struktur:

```
/<HTTP server root directory>/SAP/ITS/MIMES/<service>/<theme>/<language>/<name>
```

Wenn die Parameter ~service, ~theme und ~language nicht angegeben werden, werden die entsprechenden Werte aus der Servicebeschreibung verwendet. Wenn Sie den Parameter ~language ohne einen Wert angeben (~language=""), verwendet mimeURL die Dateien in dem übergeordneten Theme-Verzeichnis. Hiermit können Sie auf sprachenunabhängige Dateien zugreifen.

## Beschreibung

Mit der Funktion mimeURL können Sie Links zu statischen Dateien wie z.B. Bildern und Hilfedateien angeben, die sich zur Laufzeit in den HTML-Templates befinden sollen. Diese Dateien werden nicht im ITS-Verzeichnis, sondern im Verzeichnis des HTTP-Servers abgelegt.

Hierfür ist es nicht ausreichend, eine einfache URL anzugeben. Die URL muß Parameter enthalten, die die statische Datei einem bestimmten Service sowie einer bestimmten Theme und Sprache zuordnen.

Die Funktion mimeURL ersetzt die Funktion imageURL und sollte an deren Stelle verwendet werden. Sie bietet flexibleren und einfacheren Zugriff auf service-, sprach- und theme-abhängige Dateien wie Bilder, Ton- und andere Multimediadaten (daher der Name mimeURL).

Außer servicespezifischen Dateien können Sie die mit der Funktion mimeURL globale Dateien über den Parameter ~service="global" aufrufen.

Die den Parametern zugewiesenen Werte werden ohne Überprüfung vom ITS geholt, um die URL zu erstellen. So können Sie weitere mit dem MIMES-Verzeichnis verknüpfte Unterverzeichnisse anlegen, die über die Funktion mimeURL aufgerufen werden können.



Die folgenden Beispiele illustrieren die Verwendung von mimeURL.

Wenn ~URLmime auf /sap/its/mimes gesetzt, der aktuelle Service vw01, die aktuelle Sprache EN und die aktuelle Theme 99 ist, ergibt der Aufruf von mimeURL die folgende Ausgabe:

```
1. 
```

Ergibt:

```

```

```
2. 
```

Ergibt:

```

```

```
3. 
```

Ergibt:

```

```

**Die Funktion mimeTypeURL**

4. ``  
Ergibt:  
``
5. ``  
Ergibt:  
``
6. ``  
Ergibt:  
``
7. ``  
Ergibt:  
``

## Die Funktion printf

### Einsatzmöglichkeiten

Gibt eine Zeichenkette im angegebenen Format zurück.

### Syntax

```
string printf (in string format,...)
```

### Parameter

- Format*    Formatzeichenkette (Template), aus der die endgültige Kette gebildet wird. Die Formatzeichenkette enthält "%" -Formatangaben, die durch als ...-Argumente übergebene Parameter ersetzt werden.
- ...        Variable Anzahl von Argumenten

### Beschreibung

Gibt eine Zeichenkette zurück, die aus der angegebenen Formatzeichenkette gebildet wird, wobei die Formatangaben aus einer variablen Argumentliste aufgelöst werden (wie die C-Funktion printf()). Formatangaben haben folgende Form:

**%<flags><width><.pre><conversion>**,

wobei:

## Die Funktion printf

<b>&lt;flags&gt;</b> null oder mehr der folgenden Angaben enthalten kann:	
-	Linksbündig im Feld
+	Anzeige führender Zeichen
<space>	Raum für Präfix anstatt für Zeichen
0	Auffüllen mit führenden Nullen
<b>&lt;width&gt;</b>	(optional) Anzahl der Zeichen, die erweitert werden
<b>&lt;pre&gt;</b>	(optional nach.) Verwendet für
diouxX	Mindestzahl Stellen
s	Höchstzahl Stellen
<b>&lt;conversion&gt;</b> eine der folgenden Angaben enthält:	
%	"%"-Zeichen
d	Ganze Dezimalzahl mit Vorzeichen
l	Ganze Zahl mit Vorzeichen
u	Ganze Dezimalzahl ohne Vorzeichen
x, X	Ganze Hexadezimalzahl ohne Vorzeichen
o	Oktalzeichen ohne Vorzeichen
c	Character-Anzeige
s	Zeichenkette

## Rückgabewert

Gibt eine Zeichenkette zurück, die aus der Formatzeichenkette erstellt wird, bei der alle "%" - Formatangaben ersetzt werden.

## Beispiel

```
`printf ("Lawrence Smith", format="%-20s will be opening %s in %s on %s
%02d, %04d.",
"a menswear store", "Billingsley", "January", 1, 1999)`
```

## Die Funktion strCat

### Einsatzmöglichkeiten

Hängt eine Zeichenkette an eine andere an.

### Syntax

```
string strCat (inout string destString,  
              in string sourceString)
```

### Parameter

*DestString* Variable, die das Ziel der Append-Operation angibt

*SourceString* Ausdruck, der die Quelle der Append-Operation angibt.

### Beschreibung

Diese Funktion verkettet Zeichenketten und stellt das Ergebnis in die Zielzeichenkette. ``a = a & b`` ist gleichbedeutend mit dem Aufruf ``strCat (a, b)``.

### Rückgabewert

Gibt den Inhalt von *destString* zurück.

### Beispiel

```
`if (toLower(strCat(~language, "glish")) == "english")`  
English Version:  
`end`
```

## Die Funktion strCmp

# Die Funktion strCmp

## Einsatzmöglichkeiten

Vergleicht zwei Zeichenketten auf Übereinstimmung.

## Syntax

```
int strCmp (in string string1,  
           in string string2)
```

## Parameter

*string1* Zeichenkette auswertender Ausdruck

*string2* Zeichenkette auswertender Ausdruck

## Beschreibung

Diese Funktion vergleicht zwei Zeichenketten. ``a == b`` ist gleichbedeutend mit dem Aufruf ``strCmp (a, b)``.

## Rückgabewert

Diese Funktion gibt zurück:

- 0, wenn *string1* gleich *string2* ist.
- <0, wenn *string1* kleiner als *string2* ist.
- >0, wenn *string2* kleiner als *string1* ist.

## Beispiel

```
`if (strCmp(toLower(~language), "en") == 0)`  
English Version:  
`end`
```

## Die Funktion strCpy

### Einsatzmöglichkeiten

Legt eine Kopie einer beliebigen Zeichenkette an.

### Syntax

```
string strCpy (inout string destString,  
              in    string sourceString)
```

### Parameter

*destString* Variable, die das Ziel der Kopieroperation angibt.

*sourceString* Ausdruck, der als Quelle für die Kopieroperation verwendet wird.

### Beschreibung

Diese Funktion kopiert einen Zeichenkettenwert in eine andere Ablage. ``a = b`` ist gleichbedeutend mit ``strCpy (a, b)``.

### Rückgabewert

Gibt den Inhalt von *destString* zurück.

### Beispiel

```
`if (toLower(strCpy(l, ~language)) == "en")`  
English Version:  
`end`
```

## Die Funktion striCmp

# Die Funktion striCmp

## Einsatzmöglichkeiten

Vergleicht zwei Ausdrücke auf Übereinstimmung, wobei Groß- und Kleinschreibung außer acht gelassen wird.

## Syntax

```
int striCmp (in string string1,  
            in string string2)
```

## Parameter

*string1* Zeichenkette auswertender Ausdruck

*string2* Zeichenkette auswertender Ausdruck

## Beschreibung

`toLower(a) == toLower(b)` ist gleichbedeutend mit dem Aufruf `striCmp (a, b)`.

## Rückgabewert

Dies Funktion gibt zurück:

- 0, wenn *string1* gleich *string2* ist, wobei Groß-/Kleinschreibung außer acht gelassen wird.
- <0, wenn *string1* kleiner als *string2* ist, wobei Groß-/Kleinschreibung außer acht gelassen wird.
- >0, wenn *string2* kleiner als *string1*, wobei Groß-/Kleinschreibung außer acht gelassen wird.

## Beispiel

```
`if (striCmp(~language, "en") == 0)`  
English Version:  
`end`
```

## Die Funktion strLen

### Einsatzmöglichkeiten

Gibt die Anzahl der Zeichen einer Kette zurück.

### Syntax

```
int strLen (in string string)
```

### Parameter

*String* Kette, deren Zeichen gezählt werden sollen

### Rückgabewert

Gibt die Anzahl der Zeichen in der Kette zurück.

### Beispiel

```
`if (strnCmp(~language, "en", strLen("en")) == 0)`  
English Version:  
`end`
```

---

## Die Funktion strLwr

### Die Funktion strLwr

#### Einsatzmöglichkeiten

Gibt eine Kopie einer Zeichenkette in Kleinschreibung zurück.

#### Syntax

```
string strLwr (in string string)
```

#### Parameter

*String* Kette, für die eine Kopie in Kleinschreibung gewünscht wird.

#### Rückgabewert

Gibt eine Kopie der Zeichenkette in Kleinbuchstaben zurück.

#### Beispiel

```
`if (strnCmp(strLwr(~language), "en", 1) == 0)`  
English Version:  
`end`
```

## Die Funktion strnCmp

### Einsatzmöglichkeiten

Vergleicht die ersten n Zeichen zweier Zeichenketten auf Übereinstimmung.

### Syntax

```
int strnCmp (in string string1,  
            in string string2,  
            in int    numberOfChars)
```

### Parameter

*string1*            Zeichenkette auswertender Ausdruck  
*string2*            Zeichenkette auswertender Ausdruck  
*numberOfChars*    Anzahl der zu vergleichenden Zeichen

### Rückgabewert

Die Funktion gibt zurück:

- 0, wenn die Zahl der ersten *numberOfChars*-Zeichen von *string1* gleich der Zahl der ersten *numberOfChars*-Zeichen von *string2* ist.
- <0, wenn die Zahl der ersten *numberOfChars*-Zeichen von *string1* kleiner als die Zahl der ersten *numberOfChars*-Zeichen von *string2* ist.
- >0, wenn die Zahl der ersten *numberOfChars*-Zeichen von *string2* kleiner als die Zahl der ersten *numberOfChars*-Zeichen von *string1* ist.

### Beispiel

```
`if (strnCmp(toLower(~language), "en", 1) == 0)`  
English Version:  
`end`
```

## Die Funktion strnCmp

# Die Funktion strnCmp

## Einsatzmöglichkeiten

Vergleicht die ersten *n* Zeichen zweier Zeichenketten auf Übereinstimmung, wobei Groß-/Kleinschreibung außer acht gelassen wird.

## Syntax

```
int strnCmp (in string string1,  
            in string string2,  
            in int    numberOfChars)
```

## Parameter

*string1*            Zeichenkette auswertender Ausdruck  
*string2*            Zeichenkette auswertender Ausdruck  
*numberOfChars*    Anzahl der zu vergleichenden Zeichen

## Rückgabewert

Die Funktion gibt zurück:

- 0, wenn die ersten *numberOfChars*-Zeichen von *string1* und *string2* gleich sind, wobei Groß-/Kleinschreibung außer acht gelassen wird.
- <0, wenn die ersten *numberOfChars*-Zeichen von *string1* kleiner als die von *string2* sind, wobei Groß-/Kleinschreibung außer acht gelassen wird.
- >0, wenn die ersten *numberOfChars*-Zeichen von *string2* kleiner als die von *string1*, sind, wobei Groß-/Kleinschreibung außer acht gelassen wird.

## Beispiel

```
`if (strnCmp(~language, "en", 1) == 0)`  
English Version:  
`end`
```

## Die Funktion strStr

### Einsatzmöglichkeiten

Sucht nach einer beliebigen Zeichenfolge innerhalb einer Zeichenkette.

### Syntax

```
int strStr (in string string,  
           in string stringPattern)
```

### Parameter

*String* Kette, in der gesucht werden soll

*stringPattern* Zeichenfolge, nach der gesucht werden soll

### Rückgabewert

Die Funktion gibt zurück:

- 0, wenn *string* nicht *stringPattern* enthält.
- >0, wenn *string* den Wert *stringPattern* enthält. Hierbei gibt der Wert ungleich null die Anfangsposition der gefundenen Zeichenfolge an.

### Beispiel

Der Ausdruck `strStr("Hello world", "world")` gibt die Stelle (Position 7) an, an der "world" in "Hello world" beginnt.

## Die Funktion strSub

# Die Funktion strSub

## Einsatzmöglichkeiten

Gibt eine Zeichenfolge aus einer beliebigen Zeichenkette zurück.

## Syntax

```
string strSub (in string string,  
              in int    position,  
              in int    length)
```

## Parameter

*string* Zeichenkette auswertender Ausdruck  
*position* Anfangsstelle der Zeichenfolge in der Zeichenkette [1..n]  
*length* Anzahl der Zeichen, die in der Zeichenfolge ab der Stelle *position* zurückgegeben wird

## Beschreibung

Die Funktion gibt eine Zeichenfolge aus einer beliebigen Zeichenkette zurück. Sie können die Anfangsstelle und Länge angeben. strSub gibt die *length*-Zeichen an der gewünschten Stelle *position* zurück.

Wenn Sie eine Länge von 0 angeben, gibt die Funktion alle Zeichen von Anfang bis Ende der Kette zurück. Wenn *length* größer als die vorhandenen Zeichen in der Quelle *string* ist, werden alle Zeichen bis zum Kettenende genommen.

## Rückgabewert

Gibt eine Zeichenfolge (Kopie) einer beliebigen Zeichenkette zurück.

## Beispiel

```
`name = "Walt Whitman" `  
The last name of `name` is  
`if (0 != (j = strstr (name, " ")))`  
    `strSub (name, j + 1, 0)`  
`else`  
    undefined.  
`end`
```

## Die Funktion strUpr

### Einsatzmöglichkeiten

Gibt eine Kopie einer Zeichenkette in Großbuchstaben zurück.

### Syntax

```
string strUpr (in string string)
```

### Parameter

*String* Zeichenkette, für die eine Kopie in Großbuchstaben gewünscht wird.

### Rückgabewert

Gibt eine Kopie einer Zeichenkette in Großbuchstaben zurück.

### Beispiel

```
`if (strnCmp(strUpr(~language), "EN", 1) == 0)`  
English Version:  
`end`
```

---

## Die Funktion toLower

### Die Funktion toLower

#### Einsatzmöglichkeiten

Gibt eine Kopie einer Zeichenkette in Kleinbuchstaben zurück.

#### Syntax

```
string toLower (in string theString)
```

#### Parameter

*theString* Zeichenkette auswertender Ausdruck

#### Beschreibung

Die Funktion gibt eine Kopie in Kleinbuchstaben der eingegeben Zeichenkette zurück. toLower ändert die eingegebene Kette nicht.

#### Rückgabewert

Gibt eine Kopie einer Zeichenkette in Kleinbuchstaben zurück, die als *theString* übergeben wird.

#### Beispiel

```
`if (toLower(~language) == "en")`  
English Version:  
`end`
```

## Die Funktion toUpper

### Einsatzmöglichkeiten

Gibt eine Kopie einer Zeichenkette in Großbuchstaben zurück.

### Syntax

```
string toUpper (in string theString)
```

### Parameter

*theString* Zeichenkette auswertender Ausdruck

### Beschreibung

Die Funktion legt eine Version der eingegebenen Zeichenkette in Kleinbuchstaben an und gibt sie zurück. ToUpper ändert das Original nicht.

### Rückgabewert

Gibt eine Kopie einer Zeichenkette in Großbuchstaben zurück, die als *theString* übergeben wird.

### Beispiel

```
`if (toUpper(~language) == "EN")`  
English Version:  
`end`
```

## Die Funktion wgateURL

## Die Funktion wgateURL

### Einsatzmöglichkeiten

Erzeugt dynamisch URLs, die mit dem aktuellen Web-Server übereinstimmen.

### Syntax

```
wgateURL(parameter = expression {, parameter = expression})
```

### Parameter

Die folgende Tabelle zeigt die möglichen Parameter der Funktion wgateURL:

Parameter	Aufgabe
~OkCode	Diesem Parameter wird der Funktionscode angegeben, der in der R/3-Transaktion ausgelöst werden soll.
~target	Diesem Parameter wird der Name des Ziel-Frames in einem <form>-Befehl zugewiesen. Dieser Parameter hat eine besondere Bedeutung. Einzelheiten finden Sie unter <a href="#">Web-Browser-Unabhängigkeit [Seite 92]</a> . (Ab ITS-Version 2.2 ersetzt ~target den Parameter ~forceTarget.)
~forceTarget	Dieser Parameter wird nicht mehr unterstützt und sollte gelöscht werden. Verwenden Sie ab ITS-Version 2.2 für alle Browser ~target.
~FrameName	Bestimmt die URL eines Frames in einem Frameset-Dokument.
~secure	Gibt an, ob eine relative oder absolute URL-Adresse verwendet werden soll: <ul style="list-style-type: none"> <li>• <b>~secure</b> wird nicht verwendet: wgateURL() legt eine relative URL wie in Versionen vor ITS 2.2 an.</li> <li>• <b>~secure="on"</b>: wgateURL() legt eine absolute URL an und verwendet den Protokolltyp HTTPS. Verwenden Sie den Wert ON, wenn Sie von einer (unsicheren) HTTP-Kommunikation zu einer (sicheren) HTTPS-Kommunikation während einer laufenden Verbindung wechseln möchten.</li> <li>• <b>~secure="off"</b>: wgateURL() legt eine absolute URL an und verwendet den Protokolltyp HTTP. Verwenden Sie den Wert OFF, wenn Sie von einer (sicheren) HTTPS-Kommunikation zu einer (unsicheren) HTTP-Kommunikation während einer laufenden Verbindung wechseln möchten.</li> </ul>
Screen field name	Füllt Dynprofelder im Ziel-Frame.

## Beschreibung

Diese Funktion erzeugt dynamisch URL-Adressen, die mit den Systeminformationen für den aktuellen HTTP-Server übereinstimmen.

Um möglichst sicher und einfach die Kommunikation mit der ITS-Software zu ermöglichen, wurde in HTML<sup>Business</sup> die Funktion `wgateURL` aufgenommen. Mit dieser Funktion können Sie HTML-Templates unproblematisch von einem Server auf einen anderen portieren. Zur Laufzeit erzeugt `wgateURL` dynamisch URLs, die mit den Systeminformationen für den aktuellen HTTP-Server übereinstimmen. Sie müssen daher URLs nicht fest in HTML-Templates einprogrammieren.

Neben dem Einmischen von Systeminformationen wie dem Feld `~state` kodiert die Funktion `wgateURL` auch alle Parameter URL-konform. Weitere Einzelheiten finden Sie unter [Die Funktion writeEnc \[Seite 63\]](#).

Verwenden Sie `wgateURL` in den folgenden drei Fällen:

- In `<form>`-Befehlen müssen Sie die URL des Programms angeben, das den Inhalt des Formulars verarbeitet. Dieser Pfad hängt von der Art der Installation Ihres ITS ab und muß deshalb dynamisch bestimmt werden.
- Wenn Sie eine Transaktion mit mehreren Frames verwenden, liefert diese Funktion den Ziel-Frame. In diesem Fall muß die URL des ITS erzeugt werden und zusätzliche Parameter eingemischt werden.
- In einem Frameset-Dokument liefert diese Funktion die URL für einen Frame.

## Anker in der URL anlegen

Über den Parameter `~anchor` kann ein HTML-Anchor in die WGate-URL eingebaut werden:

```
<form method="POST" action="`wgateURL(~anchor="footer")`">
```

Dies wird expandiert zu:

```
<form method="POST" action="/scripts/wgate.dll/vw01#footer">
```



Das folgende Beispiel soll die Funktionsweise von `wgateURL` verdeutlichen:

```
<ul>
  `repeat with j from 1 to xlist-matnr.dim`
  <li>
    <a href="`wgateURL(matnr=xlist-matnr[j], quantity=xlist-
      kwmeng[j])`">
      `xlist-matbez[j]` </a> </li>
  `end`
</ul>
```

Dieses HTML-Template würde folgendermaßen expandiert werden:

```
<ul>
  <li>
    <a
      href="http://pn0208/scripts/wgate.dll/vw01?~State=4711&matnr=9
```

## Die Funktion wgateURL

```
132&quantity=2">
  Microsoft Word Update </a> </li>
<li>
  <a
href="http://pn0208/scripts/wgate.dll/vw01?~State=4711&matnr=9
133&quantity=1">
  Microsoft Excel Update </a> </li>
</ul>
```



Stellen Sie sicher, daß der Wert des Attributes `href` in Anführungszeichen (") eingeschlossen ist.

## Parameter `~secure`

Sie können mit der Systemvariablen `~http_https` abfragen, ob der aktuelle Zugriff HTTP oder HTTPS verwendet. (Für HTTP, `~http_https = "off"`, und for HTTPS, `~http_https = "on"`). Im folgenden sehen Sie Beispiele für den Einsatz des Parameters `~secure`:

- `<a href=" wgateURL(~okcode="back")`"> wird expandiert zu <a href="/scripts/wgate/vw01/~sd..f?~okcode=back">`
- `<a href=" wgateURL(~okcode="back", ~secure="on")`"> wird expandiert zu <a href="https://myHost:444/scripts/wgate/vw01/~sd..f?~okcode=back">`
- `<a href=" wgateURL(~okcode="back", ~secure="off")`"> wird expandiert zu <a href="http://myHost:1080/scripts/wgate/vw01/~sd..f?~okcode=back">`
- `<a href=" wgateURL(~okcode="back", ~secure=~http_https)`">` wird entweder zur HTTP- oder HTTPS-Variante expandiert, abhängig vom Protokolltyp des aktuellen Zugriffs.

Der Parameter `~secure` erfordert zusätzliche Informationen in der Datei `.srvc` (normalerweise `global.srvc`):

Systemparameter	Bedeutung
<code>~hostSecure</code>	Hostname und Domäne des Web-Servers für sichere Zugriffe (HTTPS). Wenn nichts angegeben wird, greift der Standardwert <code>~hostUnsecure</code> .
<code>~portSecure</code>	Portnummer für sichere Zugriffe (HTTPS). Wenn nichts angegeben wird, greift der Standardwert 443 (Standard-HTTPS-Port).
<code>~hostUnsecure</code>	Hostname und Domäne des Web-Servers für unsichere Zugriffe (HTTP). Wenn nichts angegeben wird, greift der Standardwert <code>~http_host</code> aus dem HTTP-Header des aktuellen Zugriffs.
<code>~portUnsecure</code>	Portnummer für unsichere Zugriffe (HTTP). Wenn nichts angegeben wird, greift der Standardwert 80 (Standard-HTTP-Port).

Diese zusätzlichen Systemparameter werden der globalen Servicedatei während der Installation der ITS-Version 2.2 hinzugefügt.



## Die Funktion write

# Die Funktion write

## Einsatzmöglichkeiten

Schreibt die Ausgabe in die HTML-Seite, ohne Leerzeichen oder andere Trennzeichen einzufügen.

## Syntax

```
write ( <expression> {, <expression>} )
```

## Beschreibung

Die Funktion `write` in HTML<sup>Business</sup> ähnelt derselben Funktion in JavaScript. Beide Funktionen schreiben Feldinhalte und andere Informationen in das HTML-Dokument. Der einzige Unterschied zwischen `write` und einer einfachen Ersetzung ist, daß `write` individuelle Argumente ausgibt, ohne zwischen den Werten Leerzeichen oder andere Trennzeichen einzufügen.



Zwei Beispiele:

```
<p> `write (i, ". ", xlist-matnr)` </p>
```

```
<p> `write (j * 2, 123, j > 3)` </p>
```

## Die Funktion writeEnc

### Einsatzmöglichkeiten

Schreibt die Ausgabe in die HTML-Seite und wandelt dabei alle nicht-alphanumerischen Zeichen in die entsprechende Hexadezimalkodierung um.

### Syntax

```
writeEnc ( <expression> {, <expression>} )
```

### Beschreibung

Die Funktion writeEnc verhält sich analog zu write. Im Gegensatz zu write wird die Ausgabe jedoch URL-kodiert. Alle nicht-alphanumerischen Zeichen (Leerzeichen und Sonderzeichen) werden in die entsprechende Hexadezimalkodierung umgewandelt. Mit dieser Funktion können Sie URLs korrekt aus Feldern aufbauen, die Leerzeichen und Sonderzeichen enthalten.



Folgendes Beispiel soll den Unterschied zwischen write und writeEnc verdeutlichen. Der HTML-Quelltext

```
<a href="http://pn0208/scripts/any.dll?matbez=`write (matbez[j])`&quantity=1">
```

wird im ungünstigsten Fall expandiert zu:

```
<a href="http://pn0208/scripts/any.dll?matbez=Grosse Stuehle&quantity=1">
```

Leerzeichen, wie zwischen "Grosse Stuehle", dürfen in URLs aber nicht enthalten sein. Wird dagegen "writeEnc" wie folgt verwendet:

```
<a href="http://pn0208/scripts/any.dll?matbez=`writeEnc(matbez[j])`&quantity=1">
```

ergibt sich richtig (die Zeichenketten "Grosse" und "Stuehle" wurden vom Quelltext durch ein Leerzeichen getrennt):

```
<a href="http://pn0208/scripts/any.dll?matbez=Grosse%20Stuehle&quantity=1">
```

## HTML<sup>Business</sup>-Funktionsbeschreibung

### Allgemeine Regeln

Zur Unterstützung des Designs von HTML-Templates bei der Implementierung von Internet-Anwendungskomponenten (Internet Application Components=IACs) können Sie Ihre eigenen HTML<sup>Business</sup>-Funktionen schreiben, die in syntaktischer Hinsicht den Programmiersprachen ABAP oder C ähnlich sind.

Eine einfache HTML<sup>Business</sup>-Funktion `concatenate_string`, die zwei Zeichenketten verbindet, hat beispielsweise folgende Form:

```
`function concatenate_string (string1,string2)
  result = string1 & string2;
  return (result);
end;`
```

Sie übergeben die beiden zu verbindenden Zeichenketten in den Parametern `string1` und `string2` und geben dann `result` zurück.

Sie können die Funktion `concatenate_string` folgendermaßen aufrufen:

```
...
`write (concatenate_string("A","B"));`
...
```

Dies ergibt folgende Ausgabe: **AB**.

Der Vorteil der Definition von HTML<sup>Business</sup>-Funktionen liegt darin, daß Sie sie beliebig oft in anderen Templates wiederverwenden können. Sie brauchen daher nicht jedes Mal den Quelltext neu zu definieren.

SAP liefert keine Standardbibliothek mit wiederverwendbaren HTML<sup>Business</sup>-Funktionen aus. Es ist Aufgabe Ihres Entwicklungsteams, Funktionen zu definieren, die Ihren anwendungsspezifischen Bedürfnissen genügen.

### HTML<sup>Business</sup>-Funktionen definieren

Die Funktionsdefinition gibt an:

- Den Namen der Funktion
- Die Zahl und Namen der Parameter, die die Funktion erhalten soll
- Einen Funktionskörper mit den Anweisungen, die festlegen, was die Funktion tun soll
- Einen Rückgabewert

HTML<sup>Business</sup>-Funktionen müssen vor ihrem Einsatz definiert sein. Sie können eine Funktionsdefinition an einer beliebigen Stelle im Quelltext platzieren, jedoch **nicht** innerhalb einer anderen HTML<sup>Business</sup>-Anweisung. So können Sie etwa eine Funktionsdefinition nicht in eine `if`-Anweisung einbinden.

### HTML<sup>Business</sup>-Funktionssyntax

Die Grundsyntax einer HTML<sup>Business</sup>-Funktionsdefinition lautet:

```
function <function name> (<parameter list>)
  <function body>
```

```
    return ( <return value> );
end;
```

In der folgenden Tabelle sehen Sie alle Syntaxkomponenten:

Komponente	Bedeutung
<function name>	Eindeutige Kennung, die sich aus einer ununterbrochenen Folge von alphanumerischen Zeichen und zulässigen Sonderzeichen zusammensetzt.  HTML <sup>Business</sup> -Funktionsnamen <b>müssen</b> mit einem Buchstaben beginnen. Anschließend können Sie eine beliebige Kombination von Großbuchstaben (A bis Z), Kleinbuchstaben (a bis z) und Zahlen (0 bis 9) verwenden.  Leerzeichen sind nicht zulässig, aber Sie können mit dem Unterstrichzeichen _ die einzelnen Wörter verbinden, die den Funktionsnamen bilden.
<parameter list>	Die Namen der zu übergebenden Parameter  Wenn der Aufrufer keinen Parameter angibt, wird der Standardausdruck verwendet,
<function body>	Gültige HTML <sup>Business</sup> -Anweisung
<return value>	Gültige HTML <sup>Business</sup> -Anweisung

### HTML<sup>Business</sup>-Funktionen aufrufen

Sie können HTML<sup>Business</sup>-Funktionen als Teil einer gültigen HTML<sup>Business</sup>-Anweisung aufrufen. Die Syntax dafür lautet:

```
~
...
<function name> (<parameter expression>)
...
~
```

Der <parameter expression> besteht aus einer Liste mit Parameternamen, die durch Kommata getrennt sind. Die Syntax für einen einzelnen Parameternamen lautet:

```
<parameter name>=<any valid HTMLBusiness expression>
```

Beim Aufruf einer HTML<sup>Business</sup>-Funktion wird jeder Parameterausdruck geprüft, und der Ergebniswert wird der passenden Parameterkennung zugeordnet.

- Wenn Parameter mit dem Parameternamen identifiziert werden, wird der passende Parameternamen in der Funktionsdefinition nachgeschaut. Wenn kein entsprechender Name gefunden wird, tritt ein Laufzeitfehler auf.
- Wenn Parameter nicht mit dem Parameternamen identifiziert werden, werden sie als positionale Parameter betrachtet und einer nach dem anderen in die entsprechende Parameterkennung kopiert; dies geschieht durch Kopieren des nten Parameters in die nte Parameterkennung.

Sie können die Funktion `concatenate_string` also auf zwei verschiedene Weisen aufrufen:

## HTMLBusiness-Funktionsbeschreibung

- Mit Parameternamen wie in:

```
concatenate_string(string1="prefix",string2="suffix") ;
```

- Mit positionalen Parametern wie in:

```
concatenate_string("prefix","suffix") ;
```

Eine Funktion kann eine andere Funktion aufrufen, vorausgesetzt, die andere Funktion wurde definiert. Im folgenden Beispiel ruft die Funktion `func_1` die Funktion `func_2` auf, jedoch ergibt dies einen Kompilierzeitfehler, denn die Funktion `func_2` ist zum Zeitpunkt des Aufrufs nicht definiert.

```

~
function func_1 (x)
  write(func_2(2*x));  <!--compile error -->
end;

function func_2 (y)
  write(y);
end;

func_1(10);
~

```

## HTML<sup>Business</sup>-Funktionen rekursiv aufrufen

Eine HTML<sup>Business</sup>-Funktion kann sich selbst aufrufen. Die folgende Beispielfunktion berechnet Fakultäten:

```

~
function factorial(f,x=1)
  if (x==f)
    return (x);
  else
    return (x*factorial(f,x+1));
  end;
end;

repeat with i from 1 to 10;
  factorial(i);
end;
~

```

## Rückgabewerte in HTML<sup>Business</sup>-Funktionen

HTML<sup>Business</sup>-Funktionen können Werte an den Aufrufer zurückgeben.

Wenn die HTML<sup>Business</sup>-Funktion einen Zeichenkettenwert zurückgibt, der dann in einem Ausdruck verwendet wird, der einen numerischen Wert zum Ergebnis hat, wird eine automatische Typkonvertierung durchgeführt.

Bei der folgenden Funktion `dup_string`, die Zeichenketten verdoppelt, sind alle unten aufgeführten Anweisungen möglich:

```

~
function dup_string (s)
  return (s&s);
end;
~

```

HTMLBusiness-Funktionsbeschreibung

```
dup_string(„a“);           <-- outputs "aa" -->
write(dup_string(„b“));    <-- outputs "bb" -->
dup_string(dup_string(„c“)); <-- outputs "cccc" -->
2*dup_string(„1“);        <-- outputs "22" -->
~
```

**Umfang der Variablen in HTML<sup>Business</sup>-Funktionen**

Wenn ein Parameter einer HTML<sup>Business</sup>-Funktion den gleichen Namen hat wie eine globale Variable, enthält dieser Parameter den Wert, der vom Aufrufer innerhalb der Funktion hinterlegt wurde. Der Wert der globalen Variablen wird jedoch wiederhergestellt, wenn die Verarbeitung von der Funktion an den Aufrufer zurückgeht, wie in folgendem Beispiel:

```
~
...
function write_numbers ( x1,x2,x3 )
  global_variable = x1;
  write(„global_variable is „,global_variable,“\n“);
  write(„x1 is „,x1,“\n“);
  write(„x2 is „,x2,“\n“);
  write(„x3 is „,x3,“\n“);
end;

global_variable = 1000;
x1 = 100;
x2 = 200;
x3 = 300;
write(„Before the call\n“);
write("global_variable is ",global_variable," \n");
write("x1 is ",x1," \n");
write("x2 is ",x2," \n");
write("x3 is ",x3," \n");
write("Calling the function . . .\n");
write_numbers(10,20,30);
write("After the call\n");
write("global_variable is ",global_variable," \n");
write("x1 is ",x1," \n");
write("x2 is ",x2," \n");
write("x3 is ",x3," \n");
...
~
```

Dies ergibt folgende Ausgabe:

```
Before the call
global_variable is 1000
x1 is 100
x2 is 200
x3 is 300
Calling the function . . .
global_variable is 10
x1 is 10
x2 is 20
x3 is 30
```

## HTMLBusiness-Funktionsbeschreibung

```

After the call
global_variable is 10
x1 is 100
x2 is 200
x3 is 300

```

Standard-HTML in HTML<sup>Business</sup>-Funktionen

HTML<sup>Business</sup>-Funktionen können Standard-HTML-Quelltext enthalten.

Im folgenden Beispiel gibt die HTML<sup>Business</sup>-Funktion `write_html_text` den HTML-Text `<H1> This is standard HTML </H1>` zehnmal im Web-Browser aus.

```

~
function write_html_text()
~  <!--Now entering HTML mode -->
  <H1> This is standard HTML </H1>
~end;~

~repeat with I from 1 to 10;
  write_html_text();
end;
~

```

## Feldverweise

Sie können keine Parameter an Felder über Verweise übergeben. Betrachten Sie einmal folgendes Beispiel:

```

~
...
function dump_table(t)
  <TABLE>~
  repeat with i from 1 to t.dim;
    <TR><TD>`t[i]`</TD></TR>~
  end;
  </TABLE>~
end;

mytable[1] = „Line 1“;
mytable[2] = „Line 2“;
mytable[3] = „Line 3“;
dump_table(mytable);
...
~

```

In diesem Fall gibt HTML<sup>Business</sup> nur die erste Zeile der Tabelle aus, da `mytable`, als Parameter an die Funktion `dump_table` übergeben, als der Wert der ersten Zeile des mehrwertigen Feldes `mytable` geprüft wird, der `Line 1` heißt.

Um den Namen zur Laufzeit aufzulösen, müssen Sie den Namen der Variablen als Zeichenkette übergeben und den HTML<sup>Business</sup>-Operator `^` verwenden, wie in folgendem Beispiel:

```

~
...
function dump_table(t)
  <TABLE>~

```

```

repeat with i from 1 to ^t.dim;
  <TR><TD>`^t[I]`</TD></TR>`
end;
`</TABLE>`
end;

mytable[1] = „Line 1“;
mytable[2] = „Line 2“;
mytable[3] = „Line 3“;
dump_table(mytable);
...
`

```

## Anwendungsbeispiel

Eine nützliche Anwendung besteht darin, HTML<sup>Business</sup>-Funktionen zu schreiben, die HTML-Quelltexte erzeugen. Sie können beispielsweise die Funktion `screenfield` folgendermaßen definieren:

```

`
function ( ~screenfield )
  if ( ^~screenfield.disabled )
    <!-- Screenfield is not ready for input -->
    write(^screenfield.label);
  else
    write("<INPUT TYPE=\"");
    if ( ^screenfield.type=="RadioButton" )
      write("RADIO\"");
      write(" NAME=\"", ^screenfield.group, "\"");
      write(" VALUE=\"", ^screenfield.name, "\"");
    else
      if ( ^screenfield.type=="CheckBox" )
        html_type = "CHECKBOX";
      else
        html_type = "TEXT";
      end;
      write(html_type, "\"");
      write(" NAME=\"", ^screenfield.name, "\"");
      write(" VALUE=\"", ^screenfield.value, "\"");
    end;
  end;
end;
`

```

Sie können die Funktion `screenfield` auf zwei verschiedene Weisen aufrufen:

- Mit Parameternamen wie in:  

```
screenfield (~screenfield="VBAK-VBELN");
```
- Mit positionalen Parametern wie in:  

```
screenfield ("VBAK-VBELN");
```

Funktionsaufrufe können Ausdrücke prüfen, daher können Sie immer schreiben:

```

this_field = „VBAK-VBELN“;
dnprofield (this_field);

```

---

**HTMLBusiness-Funktionsbeschreibung****HTML<sup>Business</sup>-Funktionen**

Sie können HTML<sup>Business</sup>-Funktionen in einem Template definieren und dann in jedes beliebige andere Template einfügen.

In folgendem Fall wird davon ausgegangen, daß die Funktion `screenfield` im Template `util.html` definiert ist. Um `screenfield` zu verwenden, muß in einem anderen HTML<sup>Business</sup>-Template `util.html` eingefügt sein:

```
-  
include(~service="util",~theme="",~name="util.html");  
  
screenfield(~screenfield="VBAK-VBELN");  
screenfield(~screenfield="VBAK-BELNR");  
screenfield(~screenfield="VBAK-DATUM");  
-
```

Obwohl die Anweisung `include` den Parametern Standardwerte zuweist und keine Fehlermeldung erzeugt, wenn Parameter fehlen, **müssen** Sie die Parameter `~service`, `~theme` und `~name` explizit aufführen, da der HTML<sup>Business</sup>-Interpreter zwischen Kompilierzeit-Includes und Laufzeit-Includes unterscheidet.

Kompilierzeit-Includes werden immer zur Kompilierzeit des entsprechenden Templates eingefügt. Dies ist erforderlich, damit die im eingefügten Template definierten Funktionen bekannt sind.

## HTML<sup>Business</sup> -Anweisungen

HTML<sup>Business</sup> stellt folgende Anweisungen bereit:

[for-Anweisung \[Seite 72\]](#)

[if-Anweisung \[Seite 73\]](#)

[include-Anweisung \[Seite 75\]](#)

[repeat-Anweisung \[Seite 80\]](#)

## for-Anweisung

## for-Anweisung

### Einsatzmöglichkeiten

Wiederholt eine Ersetzung in einer `for`-Schleife.

### Syntax

```
for ( expression ; expression ; expression ) statement end
```

### Beschreibung

Die `for`-Variante kann wie in C oder JavaScript verwendet werden. Im Unterschied zu C ist es nicht möglich, mehrere Ausdrücke durch Kommata getrennt aufzuführen.



Die Verwendung der `for`-Schleife kann mit den folgenden Beispielen verdeutlicht werden:

```
`for (j = 10; j > 0; j--)` <td> `end`  
`for (j = 1; j <= array.dim ; j++)`  
  Array[ `j` ]=`array[j]`  
`end`  
`for (a = "a"; a != "aaaa"; a = a & "a") a end`
```

Informationen darüber, wie HTML<sup>Business</sup> Ausdrücke auswertet, finden Sie unter [Ausdrücke \[Seite 30\]](#).

Informationen zur Verwendung von `for` zum Programmieren von Wiederholungsschleifen finden Sie unter [repeat-Anweisung \[Seite 80\]](#).

## if-Anweisung

### Einsatzmöglichkeiten

Führt bedingte Ersetzungen eines HTML<sup>Business</sup>-Ausdrucks durch.

### Syntax

```
if ( expression) statement
{ [elsif | elseif] ( expression) statement }
[else statement ]
end
```

### Beschreibung

Mit der **if**-Anweisung in HTML<sup>Business</sup> können Sie bedingte Ersetzungen in Ihrem HTML-Dokument durchführen. Bei einer **if**-Anweisung testet der ITS einige Bedingungen, bevor die Ersetzung durchgeführt wird.

Die **if**-Anweisung in HTML<sup>Business</sup> lehnt sich an die Syntax und Semantik üblicher Programmiersprachen (C, JavaScript) an. Ein beliebiges Schachteln von HTML<sup>Business</sup>-Anweisungen ist möglich: **if**-Anweisungen können andere **if**-Anweisungen enthalten. Dasselbe gilt auch für die **repeat**-Anweisung.

Die Schlüsselwörter **elsif** und **elseif** können alternativ verwendet werden.

In der Beschreibung der Syntax entspricht der Begriff *expression* einem beliebigen Ausdruck, der zu den Werten 0 (Bedingung nicht erfüllt, FALSE) oder ungleich 0 (Bedingung erfüllt, TRUE) evaluiert.



Beispiele für derartige Ausdrücke:

```
VBCOM-KUNDE >= 1000
VBCOM-KUNDE
j % 3 == 0
j / (4 - 1) != a * (b + (c + 2))
s == "Walter" & " " & "Weissmann"
(x > 2 && x <99) || (s > "abc")
```

Beispiele für **if**-Anweisungen:

```
`if (VBCOM-KUNDE) VBCOM-KUNDE else` undefinierte Kundennummer
`end`
`if (j % 3 == 0)` <TR> `else` <TD> `end`
`if (1) write("This is always true!")
elseif (0) write("This is never true!")
else write("The impossible occurred!")
```

## if-Anweisung

```
end`  
`if (x > 0 && x < 100)`  
  x ist groesser 0 und kleiner 100  
  `if (y > 0 && y < 100)`  
    y ist groesser 0 und kleiner 100  
  `elsif (Y > 100)`  
    y ist groesser 100  
  `end`  
elsif (x <= 0)`  
  x ist kleiner als 0!  
`else`  
  x ist groesser 99!  
`end`
```



Die Operatoren <, <=, > und >= sind für Zeichenketten nicht definiert. Wenn Sie diese Operatoren mit Zeichenketten verwenden, werden die Zeichenketten automatisch in numerische Werte konvertiert.

## include-Anweisung

### Einsatzmöglichkeiten

Importiert Quelltext aus verschiedenen Quelldateien in das aktuelle HTML-Template.

### Syntax

```
include([[~service=]expression,] [[~theme=]expression,]
[[~language=]expression,] [~name=]expression)
```

### Parameter

Parameter	Bedeutung
<code>~service</code>	Name des Service. Bei keiner Angabe wird der aktuelle Service verwendet.
<code>~theme</code>	Theme. Bei keiner Angabe wird die aktuelle Theme verwendet (in der Datei.srvc des Service definiert). Sie können <code>~theme=""</code> angeben, um den ganzen Theme-Teil des URL-Pfades wegzulassen.
<code>~language</code>	Sprache. <b>HINWEIS:</b> Wenn dieser Parameter fehlt, wird keine Sprache verwendet (im Gegensatz zum Parameter <code>~language</code> für mimeURL). Wenn Sie die aktuelle Sprache verwenden möchten, geben Sie <code>~language=~language</code> an.
<code>~name</code>	Name der Datei, die eingefügt werden soll. Die einzufügende Datei wird über folgenden Pfad geholt: <code>&lt;itsRootDir&gt;\&lt;virtual ITS&gt;\templates\&lt;~service&gt;\&lt;~theme&gt;\&lt;filename(~name)&gt;[_&lt;~language&gt;].&lt;extension(~name)&gt; html)</code>

### Beschreibung

Mit der `include`-Anweisung können Sie Quelltext aus einem anderen HTML-Template in das aktuelle HTML-Template einfügen.

Die Anweisung `include` hat folgende Vorteile:

- Konsistenter Stil für IACs  
IACs verwenden oft die gleichen Stilelemente wie etwa Titelleisten oder Copyright-Hinweise. Sie können diese Elemente in einer einzelnen Datei ablegen und in die entsprechenden Templates einfügen. Über diese Datei können auch Aktualisierungen an den Stilelementen leicht global vorgenommen werden.
- Wiederverwendung von Quelltexten  
Mit der Anweisung `include` können Sie ein Java- oder Visual-Basic-Skript in mehreren Templates wiederverwenden. Dies ist vor allem nützlich, wenn Sie oft verwendete Routinen wie z.B. Gültigkeitsprüfungen bei Benutzereingaben implementieren.
- Wiederverwendung von Funktionsdeklarationen

## include-Anweisung

Include-Dateien können auch Funktionsschnittstellendeklarationen enthalten. Dies ist vor allem bei externen Funktionsbibliotheken nützlich. Sie können alle Bibliotheksfunktionen in einer Datei deklarieren und die Datei in allen Templates über die Bibliothek einfügen.

### Benannte oder positionale Parameter

Sie können namens- oder positionsbezogene Parameter verwenden. Bei positionaler Klammerung (z.B., ohne `~language=` vor dem Sprachenwert), geht der ITS von folgendem aus:

- `~name` ist der erste nicht benannte Parameter von **rechts**
- `~service` ist der erste nicht benannte Parameter von **links**
- `~theme` ist der zweite nicht benannte Parameter von **links**
- `~language` ist der dritte nicht benannte Parameter von **links**



Beispiel zum Einfügen von Quelltext aus dem gleichen Verzeichnis wie das einfügende Template:

```
`include ("purchasing_titlebar")`
```

Beispiel zum Einfügen von Quelltext aus dem Verzeichnis des globalen Templates (mit der gleichen Theme):

```
`include ("global", "purchasing_titlebar")`
```

Beispiel zum Einfügen von Quelltext für externe Funktionsdeklarationen:

```
`include ("system", "", "sapxjstring.html")`
```

### Kompilier- und Laufzeitauswertung

Der ITS löst Ihre `include`-Anweisungen entweder beim Kompilieren oder zur Laufzeit auf.

Wenn alle `include`-Parameter Konstanten auswerten oder implizit durch den aktuellen Kontext definiert sind, kann der ITS sie zum Kompilierzeitpunkt auflösen. Diese Option sollten Sie vorziehen, da sich hierbei folgende Vorteile ergeben:

- ``include ("purchasing_titlebar.html")``

Service, Theme und Sprache werden durch das einfügende Template selbst definiert.

- ``include ("purchasing"&"_titlebar"&".html")``

Service, Theme und Sprache sind implizit definiert, und `"purchasing"&"_titlebar"&".html"` wertet eine Konstante aus.

- ``include ("system", "", "sapxjstring.html")``

Allen Parametern werden explizit Konstantenwerte zugeordnet.

Im folgenden sehen Sie Beispiele für die Auflösung zur Laufzeit:

- ``include ("purchasing" & screen_element & ".html")``

`screen_element` ist ein nicht-konstanter Subausdruck, der erst zur Laufzeit definiert wird.

- ``include (~service=~service, "purchasing_titlebar.html")``

## include-Anweisung

Der HTML-Business-Interpreter unterscheidet nicht zwischen Variablen, die zur Kompilier- oder Laufzeit definiert wurden, wenn er Ausdrücke vorauswertet. `~service` wird daher nicht als konstanter Ausdruck betrachtet, selbst wenn ein völlig fehlendes `~service`-Argument als Konstante und als impliziert definiert behandelt werden würde.

## Einfügen von Sprachressourcen in Templates mit Includes

Wenn Sie ein Template aus einem anderen Service in Ihrem aktuellen Service einfügen, werden auch alle Sprachressourcen, die mit dem entsprechenden Template verbunden sind, berücksichtigt.

- Wenn ein Include-Template auf die Sprachressourcendatei `<resource file>_<language>.htrc` verweist, versucht der HTML<sup>Business</sup>-Interpreter, den Namen innerhalb des Include-Templates aufzulösen.
- Wenn ein Include-Template auf die Sprachressourcendatei `<resource file>_<language>.htrc` verweist und der Name nicht innerhalb des Include-Templates aufgelöst werden kann, muß der HTML<sup>Business</sup>-Interpreter versuchen, den Namen aufzulösen.



Das Template `templateA.html` des Service A enthält das Template `templateB.html` des Service B:

- Wenn `templateB.html` des Service B auf die Sprachressourcendatei `resource1_en.htrc` verweist, löst der HTML<sup>Business</sup>-Interpreter den Namen innerhalb des Templates `templateB.html` des Service B auf.
- Wenn der Name der Sprachressourcendateien nicht von innerhalb des Include-Templates `templateB.html` aufgelöst werden kann, muß er von innerhalb `templateA.html` des Service A aufgelöst werden.

Ein Beispiel für ein Kompilierzeit-Include:

```
`include(~service="paw1",~theme="99",
~name="sapmpw01_100")`
```

`templateA.html` des Service A enthält `templateB.html` des Service B und `templateC.html` des Service C:

`templateA.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`write (#resource1)`
`include(~service="ServiceB", ~theme="99", ~name= "B" )`
`include (~service="ServiceC", ~theme="99",~name="C")`
```

`templateB.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`write (#resource1)`
`write (#resource2)`
```

`templateC.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`write (#resource1)`
`write (#resource2)`
```

Die Inhalte der entsprechenden Sprachressourcendateien sehen so aus:

Ressourcendatei	Variable	Wert
-----------------	----------	------

## include-Anweisung

templateA_en.htrc	#resource1	A1
	#resource2	A2
templateB_en.htrc	#resource1	B1
templateC_en.htrc	#resource1	C1

Wenn die Anmeldesprache Englisch (en) ist und Sie `templateA.html` ausführen, erhalten Sie folgende Ausgabe:

```
A1
B1
A2
C1
A2
```

Ein anderes Beispiel für ein Kompilierzeit-Include:

```
mytheme=99`
`include(~service="paw1",~theme=mytheme,~name="sapmpw01_100
")`
```

`templateA.html` des Service A enthält `templateB.html` des Service B, `templateB.html` des Service B enthält `templateC.html` des Service C:

`templateA.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`write (#resource1)`
`mytheme=99`
`include (~service="ServiceB", ~theme=mytheme,~name="B" )`
```

`templateB.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`mytheme=99`
`include (~service="ServiceC", ~theme=mytheme, ~name="C")`
`write (#resource1)`
`write (#resource2)`
```

`templateC.html` enthält den folgenden HTML<sup>Business</sup>-Quelltext:

```
`write (#resource1)`
`write (#resource2)`
```

Die Inhalte der entsprechenden Sprachressourcendateien sehen so aus:

Ressourcendatei	Variable	Wert
templateA_en.htrc	#resource1	A1
	#resource2	A2
templateB_en.htrc	#resource1	B1
templateC_en.htrc	#resource1	C1

Wenn die Anmeldesprache Englisch (en) ist und Sie `templateA.html` ausführen, erhalten Sie folgende Ausgabe:

```
A1
C1
A2
```

B1  
A2

### Einschränkungen

- Sie können die Anweisung `include` nicht in einer Sprachressourcendatei verwenden.
- Der eingefügte Dateiname muß die Erweiterung `.html` haben.

## repeat-Anweisung

**repeat-Anweisung**

Mit den Anweisungen `repeat` und `for` in HTML<sup>Business</sup> können Sie wiederholte Ersetzungen in Ihren HTML<sup>Business</sup>-Ausdrücken durchführen. Bei `repeat`- und `for`-Anweisungen führt der ITS eine Ersetzung mehrmals durch. Mit diesen Anweisungen können Step-Loops in HTML-Seiten eingemischt und beliebig strukturiert werden.

Die folgende Tabelle zeigt einige Beispiele für die Verwendung:

<b>repeat</b>	<b>for</b>
<code>repeat 10 times` &lt;p&gt;&lt;/p&gt; end`</code>	<code>for(i=1;i&lt;=10;i++)` &lt;p&gt;&lt;/p&gt; end`</code>
<code>repeat with i in FIELD_OP` &lt;option value=" i "&gt; end`</code>	(Keine äquivalente for-Anweisung.)
<code>repeat with i from 1 to FIELD_OP.dim` &lt;option value=" FIELD_OP[i] "&gt; FIELD_BZ[i] end`</code>	<code>for(i=1; i&lt;=FIELD_OP.dim; i++)` &lt;option value=" FIELD_OP[i] "&gt; FIELD_BZ[i] end`</code>

Einzelheiten über die in der Tabelle verwendete Syntax finden Sie unter [Syntax \[Seite 15\]](#).

Weitere Informationen zu diesen Anweisungen finden Sie unter:

[repeat \[Seite 81\]](#)

[repeat with <reg> in <field> \[Seite 82\]](#)

[repeat with <reg> from <expn> to <expn> \[Seite 83\]](#)

Weitere Informationen zu diesen Anweisungen finden Sie unter:

[for-Anweisung \[Seite 72\]](#)

## repeat

### Einsatzmöglichkeiten

Wiederholt eine Ersetzung mehrfach.

### Syntax

```
repeat expression times statement end
```

### Beschreibung

Soll ein bestimmter Bereich in der HTML-Seite mehrfach wiederholt werden und spielt der Schleifenindex keine Rolle, so kann dies einfach mit Hilfe dieser Anweisung bewirkt werden.

Informationen darüber, wie HTML<sup>Business</sup> Ausdrücke auswertet, finden Sie unter [Ausdrücke \[Seite 30\]](#).



```
`repeat 10 * c times` <td> `end`
```

repeat with <reg> in <field>

## repeat with <reg> in <field>

### Einsatzmöglichkeiten

Wiederholt eine Ersetzung für jeden Wert in einem Array.

### Syntax

```
repeat with <register> in <field statement> end
```

### Beschreibung

Sollen alle Werte eines mehrwertigen Feldes (Arrays) z.B. in einer Liste ausgegeben werden, so bietet sich diese Schleifenkonstruktion an. `repeat with <register> in` bewirkt, daß alle aktuellen Werte des mehrwertigen Feldes nacheinander in das Register gestellt werden.

Informationen darüber, wie HTML<sup>Business</sup> Ausdrücke auswertet, finden Sie unter [Ausdrücke \[Seite 30\]](#).



```
<ol>
  `repeat with value in array`
  <li>`value`</li>
  `end`
</ol>
```

## repeat with <reg> from <expn> to <expn>

### Einsatzmöglichkeiten

Wiederholt eine Ersetzung durch Verwendung eines Schleifenindex.

### Syntax

```
repeat with <register> from <expression> to <expression> [by
<expression>] statement end
```

### Beschreibung

Mit der Schleife [repeat with <reg> in <field> \[Seite 82\]](#) kann immer nur über eine einzelne Spalte eines Step-Loops iteriert werden.

Sollen mehrere Spalten parallel abgelaufen werden, müssen Sie die einzelnen Feldwerte über einen Index ansprechen. Diesen Laufindex erhalten Sie durch Verwendung von **repeat with . . . from**. Hier nimmt das Register nacheinander die Werte beginnend mit dem auf "from" folgenden Ausdruck bis zum auf "to" folgenden Ausdruck an. Über die Verwendung von "by" kann auch eine Schrittweite ungleich 1 spezifiziert werden.

Informationen darüber, wie HTML<sup>Business</sup> Ausdrücke auswertet, finden Sie unter [Ausdrücke \[Seite 30\]](#).



```
<table>
  `repeat with index from 1 to xlist-posnr.dim`
    <tr> <td> `xlist-posnr[i]` </td>
      <td> `xlist-matnr[i]` </td>
      <td> `xlist-arktx[i]` </td>
      <td> `xlist-kwmeng[i]` </td> </tr>
  `end`
</table>
```

Soll diese Tabelle in umgekehrter Reihenfolge ausgegeben werden, wird folgender HTML-Quelltext verwendet:

```
<table>
  `repeat with index from xlist-posnr.dim to 1 by "-1" `
    <tr> <td> `xlist-posnr[i]` </td>
      <td> `xlist-matnr[i]` </td>
      <td> `xlist-arktx[i]` </td>
      <td> `xlist-kwmeng[i]` </td> </tr>
  `end`
</table>
```

## Zusammenfassung der HTMLBusiness-Sprachgrammatik

Zusammenfassung der HTML<sup>Business</sup>-Sprachgrammatik

Für HTML<sup>Business</sup>-Ausdrücke wurde die nachfolgende Grammatik implementiert.

Nonterminal	Ableitung
htmlbusiness	([ html   script[;] ] htmlbusiness)   eof
html	<b>bytestream</b>
script	(declaration   expression   conditional   loop)
declaration	<b>declare</b> externalfn {, externalfn } <b>in</b> module
externalfn	identifizier
module	constant
function	funcname ( argument {, argument})
argument	[identifizier =] expression
expression	simpleexpr [compop simpleexpr]
simpleexpr	term { addopr simpleexpr}
term	factor { mulopr factor}
factor	(!   ++   --) factor ( expression )   assignment   lvalue [++   --]   constant
function call	internalfn ( argument {, argument})   externalfn ( expression {, expression})
internalfn	<b>write</b>   <b>writeEnc</b>   <b>wgateURL</b>   <b>archiveURL</b>   <b>imageURL</b>   <b>mimeURL</b>   <b>assert</b>
mulopr	<b>*</b> / <b>%</b> <b>&amp;&amp;</b>
addopr	<b>+</b> - <b>&amp;</b> <b>  </b>
compop	<b>==</b>   <b>!=</b>   <b>&gt;</b>   <b>&lt;</b>   <b>&gt;=</b>   <b>&lt;=</b>
lvalue	field   register
field	{ <b>^</b> } identifizier [ [ expression ] ] [. attribute ]

Zusammenfassung der HTMLBusiness-Sprachgrammatik

attribute	<b>label</b>   <b>visSize</b>   <b>maxSize</b>   <b>dim</b>   <b>disabled</b>   <b>name</b>   <b>value</b>
assignment	value = expression
conditional	<b>if</b> ( expression) htmlbusiness { ( <b>elsif</b>   <b>elseif</b> ) ( expression) htmlbusiness } <b>[else</b> htmlbusiness ]
loop	<b>repeat</b> expression <b>times</b> htmlbusiness <b>end</b>   <b>repeat with</b> register <b>in</b> field htmlbusiness <b>end</b>   <b>repeat with</b> register <b>from</b> expression <b>to</b> expression <b>by</b> expression htmlbusiness <b>end</b>   <b>for</b> ( expression ; expression ; expression) htmlbusiness <b>end</b>
register	identifier
identifier	{ ~   _   - } char { char   <b>digit</b>   _   ~   - }
constant	<b>digit</b> { <b>digit</b> }   " <b>bytestream</b> " #identifier
char	<b>a..z</b>   <b>A..Z</b>

(Informationen über die in der Tabelle verwendete Syntax finden Sie unter [Syntax \[Seite 15\]](#).)

---

**Externe Faktoren**

## Externe Faktoren

Es gibt einige externe Faktoren, die Sie bei der Arbeit mit HTML<sup>Business</sup> berücksichtigen sollten:

[Sprachenunabhängigkeit \[Seite 87\]](#)

[Web-Browser-Unabhängigkeit \[Seite 92\]](#)

[Lokales Speichern \(Caching\) auf dem Web-Browser \[Seite 95\]](#)

[Einbindung von Java-Applets und ActiveX-Controls \[Seite 97\]](#)

## Sprachenunabhängigkeit

Folgende Möglichkeiten stehen zum Erstellen sprachenunabhängiger HTML-Templates im Zusammenhang mit der ITS-Programmierung zur Verfügung:

[Abrufen von Texten aus dem R/3-System \[Seite 88\]](#)

[Verwendung von sprachenspezifischen Templates \[Seite 89\]](#)

[Verwendung von Sprachressourcendateien \[Seite 90\]](#)

## Abrufen von Texten aus dem R/3-System

## Abrufen von Texten aus dem R/3-System

Die einfachste Variante ist die Verwendung der Sprachinformationen aus dem R/3-System. Auf Grund der Anmeldung am R/3-System werden auf den Dynpros bereits sprachenspezifische Texte dargestellt. Diese Texte lassen sich in einem HTML-Template über das `.label`-Attribut auswerten und in die Seite einmischen.

Folgendes Beispiel soll dies verdeutlichen:



```
<table>
  <tr> <th> `text-posnr.label` </th>
      <th> `text-matnr.label` </th>
      <th> `text-arktx.label` </th>
      <th> `text-kwmeng.label` </th> </tr>
  `repeat with index from 1 to xlist-posnr.dim`
    <tr> <td> `xlist-posnr[i]` </td>
        <td> `xlist-matnr[i]` </td>
        <td> `xlist-arktx[i]` </td>
        <td> `xlist-kwmeng[i]` </td> </tr>
  `end`
</table>
```

Diese Variante ist unzureichend, wenn nicht alle Texte, die auf der HTML-Seite erscheinen sollen oder die eventuell hinzugefügt werden - z.B. die jeweiligen kundenspezifischen Lieferbedingungen - auf dem Dynpro plziert bzw. vorgesehen werden können. In diesem Fall muß zusätzlich eine der nachfolgenden Varianten eingesetzt werden:

[Verwendung von sprachenspezifischen Templates \[Seite 89\]](#)

[Verwendung von Sprachressourcendateien \[Seite 90\]](#)

## Verwendung von sprachenspezifischen Templates

Mit sprachenspezifischen Templates wird für jede unterstützte Sprache ein eigenes Set von Templates geschrieben. Die Namen dieser HTML-Templates tragen eine Kennung der entsprechenden Sprache, die zur Laufzeit auf die Anmeldesprache am R/3-System abgestimmt wird.

Die Namenskonvention für sprachenspezifische HTML-Templates lautet:

`<service>_<language>.html`

also z.B. `SAPLEC30_1000_D.html`.

Die HTML-Templates zu dem Service müssen in einem Verzeichnis abgelegt sein, das den Namen des Service trägt:

`... \2.0 \<virtual-ITS> \Templates \Service`

Also z.B. `C:\Program Files\SAP\ITS\2.0\<virtual-ITS>\Templates\ECS3`.

Ein typisches Template-Verzeichnis für die Sprachen Deutsch und Englisch könnte dann wie folgt aussehen:

`saplec30_1000_d.html`

`saplec30_1000_e.html`

`saplec30_2000_d.html`

`saplec30_2000_e.html`



Sie können unterschiedlich gestaltete HTML-Templates zur selben R/3-Transaktion ablegen; so muß nicht für jede Variante eine eigene Transaktion definiert werden. Sie können z.B. HTML-Templates in einem Design, andere Templates in einem zweiten Design etc. anlegen.

Es genügt vielmehr, für jede Variante einen eigenen Service zu definieren, z.B. `ECS3_SAP` und `ECS3_UpToDate`. Jeder Service ruft dann dieselbe Transaktion (`~transaction`-Anweisung in der `srvc`-Datei). Da jeder Service ein eigenes Template-Verzeichnis besitzt, können unterschiedliche HTML-Template-Varianten abgelegt werden.

Nachteilig an dem Variantenschema ist die mehrfache Ablage der ausgestalteten HTML-Seiten. Sollen gestalterische Änderungen an den HTML-Templates vorgenommen werden, müssen Sie die unterschiedlichen Sprachvarianten einzeln anpassen.

## Verwendung von Sprachressourcendateien

### Verwendung von Sprachressourcendateien

Wenn Sie dieselben HTML-Templates für alle unterstützten Sprachen verwenden möchten, können Sie sogenannte Sprachressourcendateien einsetzen. Hierbei legen Sie für jede Sprache eine eigene Sprachressourcendatei an. Die Ressourcendateien werden zusammen mit den HTML-Templates abgelegt und gemäß folgender Konvention benannt:

```
<service>_<sprache>.htrc
```

also z.B. **ECS3\_D.htrc**

Die Sprachressourcendateien zu einem Service müssen im gleichen Verzeichnis abgelegt sein wie die HTML-Templates:

```
...\2.0\<virtual-ITS>Templates\Service
```

Also z.B. C:\Program Files\SAP\ITS\2.0\<virtual-ITS>\Templates\ECS3

Werden Sprachressourcendateien eingesetzt, folgen die HTML-Templates einer geänderten Namenskonvention. In diesem Fall muß die Sprachkennung im Dateinamen weggelassen werden. Die HTML-Templates werden sonst nicht erkannt. Wenn die Kennung weggelassen wird, lautet der Dateiname wie folgt:

```
<Modulepool>_<screennumber>.html
```



Zur Verwendung von Sprachressourcendateien anstelle der folgenden sprachenspezifischen Templates

```
saplec30_1000_d.html
```

```
saplec30_1000_e.html
```

```
saplec30_2000_d.html
```

```
saplec30_2000_e.html
```

müssen Sie die folgenden Dateien implementieren (zwei Templates und zwei Ressourcendateien):

```
saplec30_1000.html
```

```
saplec30_2000.html
```

```
ecs3_d.htrc
```

```
ecs3_e.htrc
```

### Aufbau einer Sprachressourcendatei

Eine Sprachressourcendatei enthält eine Anzahl von Ressourcen-Keys, von denen jeder einen Namen und einen Wert besitzt. Der Name ist ein Platzhalter, und der Wert ist die Übersetzung des Keys in die gegebene Sprache.

Der Ressourcen-Key wird als Platzhalter in einem HTML-Template verwendet und gibt an, wo der übersetzte Text (der Ressourcenwert) zur Laufzeit eingefügt werden soll.

## Verwendung von Sprachressourcendateien



Eine Sprachressourcendatei für Deutsch kann z.B. unter anderem die folgenden Schlüssel enthalten:

Schlüssel	Text
Title	Statusabfrage fuer Bestellungen [Ask for order status]
Ok	OK
Cancel	Abbrechen [Cancel]
Exit	Beenden [Exit]
4711	Sind Sie sicher? [Are you sure?]
#4712	Diesen Key gibt es nicht! [This key does not exist!]
frontcolor	0x000000
backcolor	0xffffffff

### Verwendung der Ressourcen-Keys

In HTML-Templates können die Ressourcen-Keys überall dort verwendet werden, wo sonst auch eine HTML<sup>Business</sup>-Konstante stehen könnte. Dabei muß dem Ressourcen-Key das Nummernzeichen (#) vorangestellt werden:



```
<html>
  <head>
    <title>`#title`</title>
  </head>
  <body bgcolor="`#backcolor`" text="`#frontcolor`">
    <form action="`wgateURL()`" method="post">
      ...
      <input type=submit name="~OkCode=/00" value=" `#ok` ">
      <input type=submit name="~OkCode=/NEX" value=" `#exit` ">
    </form>
  </body>
</html>
```

## Web-Browser-Unabhängigkeit

### Web-Browser-Unabhängigkeit

Sollen dieselben HTML-Templates für unterschiedliche Web-Browser verwendet werden, so muß bei der Angabe von URLs die Funktion `wgateURL` eingesetzt werden.

Ursache für die Probleme sind das unterschiedliche Verhalten des Netscape Navigator und des Microsoft Internet Explorer bezüglich des HTTP-Headers "`Window-Target`".

- Beim Netscape Navigator können Sie den Ziel-Frame einer Anforderung vom HTTP-Server aus dynamisch setzen.
- Der Microsoft Internet Explorer ignoriert diese Angabe. Hier kann nur beim `<form>`-Befehl und `<a>`-Befehl statisch ein Ziel-Frame definiert werden. Dies ist aber nicht in allen Fällen ausreichend.

Diese Unterschiede werden durch die HTML<sup>Business</sup>-Funktion `wgateURL` verborgen, die abhängig vom anfragenden Web-Browser die geeignete Variante wählt.

Um Browser-unabhängige HTML-Templates zu schreiben, müssen Sie bei der Implementierung der HTML-Templates einplanen, welche Anforderungen in welchem Frame bzw. welchen Frames enden sollen. Der entsprechende Ziel-Frame wird dann als Parameter "`~target`" an `wgateURL` übergeben.

Die beiden folgenden Beispiele sollen dies verdeutlichen:

- [Anforderungen, die einen Frame verändern \[Seite 93\]](#)
- [Anforderungen, die mehrere Frames verändern \[Seite 94\]](#)

## Anforderungen, die einen Frame verändern

Wird durch eine Anforderung - ausgelöst durch einen Hyperlink oder die Submit-Taste ("get" oder "post") - genau ein Frame beeinflusst, muß der Parameter "~target" den Namen dieses Frames erhalten.



Im Service ECS3 werden zwei Frames eingesetzt. Der linke Frame zeigt eine Auswahl von Bestellanforderungen, jede als Hyperlink realisiert. Beim Auswählen eines Hyperlinks wird der rechte Frame mit dem Namen "subscreen\_210" verändert und zeigt zur gewählten Bestellanforderung die zugehörigen Posten an.

Der Hyperlink im linken Frame sollte dann folgendes Aussehen haben:

```
<ul>
`for (j=1; j <= xorder-vbeln.dim; j++)`
  <li> <a href="`wgateURL(xlist-vbeln=xorder-vbeln[i],
    ~OkCode="SLCT",
    ~target="subscreen_210")`"> `xorder-vbeln[i]`
`end`
</ul>
```

Je nach Web-Browser wird diese Konstruktion geeignet aufgelöst.

Bei URLs in <form>-Befehlen ist analog zu verfahren.



Der Wert des Attributes HREF im <a>-Befehl und des Attributes ACTION im <form>-Befehl muß in Anführungszeichen eingeschlossen werden.

Es darf nicht explizit ein Attribut TARGET angegeben werden.

## Anforderungen, die mehrere Frames verändern

### Anforderungen, die mehrere Frames verändern

Obwohl das Web aufgrund einer Anforderung nicht mehr als eine HTML-Seite auf einmal senden kann, führt eine Aktion in einer Seite häufig dazu, daß dynamisch der Inhalt mehrerer Frames gleichzeitig erneuert werden muß.

Um dies zu ermöglichen, kann das aktuelle Frameset-Dokument erneut an den Web-Browser gesendet werden. Dieses fordert dann alle darin definierten Frame-Seiten erneut vom Server an. Auf diese Weise ist es also möglich, entweder genau einen Frame oder aber alle Frames auf einmal aufzufrischen.

Wenn in einer Transaktion ungewiß ist, wie viele Frames aufgrund einer Aktion aufgefrischt werden müssen, oder wenn immer mehr als ein Frame erneuert werden muß, so müssen alle Frames neu beim Server angefordert werden.

Um dieses Verhalten browser-unabhängig in einer HTML-Seite zu realisieren, muß der Parameter `~target` mit dem Wert `"_parent"` oder `"_top"` belegt werden. Dies führt dazu, daß der Server das Frameset-Dokument nochmals in den obersten Frame (`_top`) oder in den Frame sendet, der dem anfragenden Frame übergeordnet ist (`_parent`).



Im Service CreateSO - eine Einkaufsanwendung - werden drei Frames eingesetzt.

Ein Frame zeigt die Produktdetails und besitzt eine Drucktaste, mit der sich das aktuell angezeigte Produkt zu dem in einem weiteren Frame angezeigten Warenkorb hinzufügen läßt. Nach dem Hinzufügen in den Warenkorb wird im Produktdetailframe wieder die Produktübersicht angezeigt. Es muß also aufgrund einer Aktion/einer Anforderung mehr als ein Frame erneuert werden.

Der entsprechende HTML-Quelltext in der Produktdetailseite lautet demnach wie folgt:

```
...
<form action="`wgateURL(~target = "_parent")`" method="post">
...
...
  <input type=submit name="~OkCode="SLCT" value=" Insert ">
</form>
```

## Lokales Speichern (Caching) auf dem Web-Browser

Je nach Einstellung kann ein Web-Browser einmal gelesene HTML-Seiten in einem Speicher halten (Caching). Dieses Caching-Verhalten kann in begrenztem Umfang vom Benutzer beeinflusst werden. Grundsätzlich muß also davon ausgegangen werden, daß der Benutzer ein beliebiges Verhalten konfiguriert hat.

Der HTTP-Server muß daher das Caching-Verhalten so beeinflussen, daß bei Interaktionsschritten auch tatsächlich mit dem R/3-System kommuniziert wird und nicht eine Seite aus dem lokalen Cache des Web-Browsers angezeigt wird.

Probleme verursacht ungewolltes Caching bei der Methode "get". Diese wird beim Einsatz von Hyperlinks verwendet und besitzt die Semantik einer reinen Leseoperation. Hierbei muß nicht unbedingt mit dem HTTP-Server kommuniziert werden, wenn der Web-Browser auf das Ergebnis einer früheren Anforderung zurückgreifen kann. Im ungünstigsten Falle bedeutet dies, daß der HTTP-Server von der Benutzeraktion nicht angesprochen wird.



Über einen Hyperlink

"<http://pn0208:1080/scripts/wgate.dll?~Service=ECS3>"  
soll der Service ECS3 angesprochen werden.

Wird dieser Hyperlink das erste Mal gewählt, wird die Anforderung vom Web-Browser an den Server gesendet, der mit einer entsprechenden Antwortseite und dem Start des Service reagiert. Danach liegt diese Seite im Cache des Web-Browsers. Ist der Web-Browser nun entsprechend konfiguriert, wird er beim nächsten Wählen des Hyperlinks die Antwortseite direkt aus dem Cache laden und nicht mehr mit dem Server kommunizieren. Dadurch wird aber auf der Serverseite auch kein Service gestartet. Sobald der Benutzer die im Cache gehaltene Seite ausgefüllt hat und an den Server schicken möchte, beantwortet dieser die Anforderung mit der Fehlermeldung

```
"Session not found!"
```

Diese Meldung ist richtig, denn es wurde kein Service gestartet.

Wird ein Formular (<form>-Befehl) verwendet und eine Anforderung über die Methode `post` an den Server gesendet, besteht dieses Problem nicht. Die Methode `post` besitzt Update-Semantik, und der Web-Browser kommuniziert bei einer Anforderung - konkreter, bei einer "Submit"-Funktion - daher immer mit dem Server.



In Web-Transaktionen ist es unbedingt erforderlich, Aktionen, die Update-Semantik besitzen, über die Methode `post` auszulösen. Es dürfen hierfür keine Hyperlinks eingesetzt werden. Verwenden Sie statt dessen ein Formular mit der Methode "`post`".

### Einsatz eines Verfallsdatums

Um beide Anforderungsmethoden gleichermaßen einsetzen zu können, müssen vom Server ausgehende Seiten je nach Typ mit einem Verfallsdatum versehen werden. Über dieses kann gesteuert werden, wann eine Seite vom Web-Browser als veraltet erkannt wird und nicht mehr

## Lokales Speichern (Caching) auf dem Web-Browser

aus dem Cache heraus verwendet werden kann. Der Web-Browser nimmt dann beim Ansprechen einer Seite erneut mit dem Server Kontakt auf.

Für die Definition des Verfallsdatums wird der HTTP-Header "**Expires**" verwendet. Er wird mit dem Wert "0" belegt, was den Web-Browser anweist, die entsprechende Seite nicht in den Cache zu stellen.

Dieser HTTP-Header wird aber nur bei Seiten verwendet, die auf Grund einer "get"-Methode - also z.B. eines Hyperlinks - angefragt werden. Seiten, die durch "post" gelesen werden, erhalten kein Verfallsdatum.

Um eine einwandfreie, automatische Caching-Steuerung zu gewährleisten, darf das Caching-Verhalten von HTML-Templates nicht durch nachträgliches Einfügen von `<meta http-equiv...>`-Befehlen beeinflusst werden.

## Frameset-Dokument

Soll das Starten eines Service indirekt über ein Frameset-Dokument erfolgen, so sollten Sie diese Seite statisch mit einem Verfallsdatum versehen, um zu gewährleisten, daß der Service tatsächlich gestartet wird.



Folgendes Beispiel zeigt einen indirekten Servicestart über eine statische Frameset-Seite:

```
<a href="http://pn0208:1080/ecs1/index.html"> Service ECS1
</a>
```

Inhalt von `index.html`:

```
<html><head><title> Order Status </TITLE>
  <meta http-equiv="Expires" content="0"> <!--
IMPORTANT!! -->
</head>
<frameset rows=90,*>
  <frame name="PRISMA" src="/ecs1/uptodate.html"
scrolling="no" >
  <frame name="R3" src="/scripts/wgate.dll?~Service=ECS1">
</frameset>
</html>
```

Bei statischen Einstiegsseiten und wenn eine Seite über die Methode "get" angesprochen wird, muß im `<head>`-Bereich der HTML-Seite der Befehl `<meta http-equiv="Expires" content="0">` angegeben werden. Beachten Sie die Großkleinschreibung von "**Expires**".

## Einbindung von Java-Applets und ActiveX-Controls

Java-Applets und ActiveX-Controls können zu Anzeigezwecken problemlos in HTML-Templates integriert werden.



Als kurzes Beispiel der Aufruf eines Java-Applets zur Anzeige eines Kreisdiagramms:

```
<applet code="PieChart.class" codebase="/ECS3" width=350 height=90>
  <param name=columns    value="3">
  <param name=scale      value="1">
  <param name=bgcolor    value="white">
  <param name=c1_color   value="red">
  <param name=c2_color   value="blue">
  <param name=c3_color   value="green">
  <param name=c1_label   value="Unconfirmed and Undelivered
    `write(100 * (ordered - confirmed) / ordered)`">
  <param name=c2_label   value="Confirmed but Undelivered
    `write(100 * (confirmed - delivered) / ordered)`">
  <param name=c3_label   value="Delivered
    `write(100 * delivered / ordered)`">
  <param name=c1_value   value="`write(100 * (ordered -
confirmed)
    / ordered)`">
  <param name=c2_value   value="`write(100 * (confirmed -
delivered) / ordered)`">
  <param name=c3_value   value="`write(100 * delivered /
ordered)`">
</applet>
```

In diesem Beispiel wurde die Codebase des Applets auf "/ECS3" gesetzt. Im Fall des Internet Information Servers muß "PieChart.class" dann im Verzeichnis "inetsrv/wwwroot/ECS3" liegen.

Analog kann die Einbindung von ActiveX-Controls erfolgen.

## Web-Eingabe dem R/3-System zuordnen

### Web-Eingabe dem R/3-System zuordnen

Dieser Teil beschreibt die Syntax und Semantik der Namen-/Wertepaare, die vom Web-Browser an das R/3-System übergeben werden.

Das einfache Konzept der Namen-/Wertepaare im Web reicht für die Nutzung in einer R/3-Umgebung nicht aus. Daher wurde eine etwas komplexere Syntax eingeführt, die vom Internet Transaction Server (ITS) für die Zuordnung der Eingabefelder aus dem Web zum R/3-System verwendet wird. Die wichtigste Erweiterung der Syntax sind die eckigen Klammern, die eine Step-Loop- oder Array-Dateneingabe ermöglichen.

Die vom Web-Browser übergebenen Parameter haben immer noch das Format `Name/Wert` und sind weiterhin URL-kodiert, d.h. im Format `Name=Wert&Name=Wert&...`. Die einzige Änderung ist, daß `Name` beispielsweise `mail-text:80[ ]` lauten kann.

Fünf Varianten der Eingabesteuerung müssen unterstützt werden:

- Eingabe eines einzelnen Wertes  
Hier sind keine Änderungen der `Name=Wert`-Syntax erforderlich. Der **Name**-Teil wird dem entsprechenden R/3-Dynprofeld unverändert zugeordnet. Ein typisches Beispiel ist ein Eingabefeld wie **firstName**.
- Eingabe mehrerer Werte  
Es muß möglich sein, Step-Loops in den Dynpros mit Eingaben aus dem Web auszufüllen. Das zugrundeliegende Konzept ist die Angabe der Step-Loop-Zeilenummer zusammen mit dem Namen.
- Beschränkung der Eingabelänge  
In manchen Fällen muß möglicherweise die Anzahl der Zeichen beschränkt werden, die an das R/3-Dynprofeld übergeben werden, oder es muß ein automatischer Eingabeumbruch ausgeführt werden (beispielsweise bei `<textarea>`-Eingabesteuerungen).
- Übergabe mehrerer Eingabeelemente von einer einzigen Web-Eingabesteuerung
- Unterstützung von Image-Maps

#### Siehe auch:

[Syntax und Semantik \[Seite 99\]](#)

[Mehrere Felder von einer HTML-Steuerung übergeben \[Seite 102\]](#)

[<textarea>-Steuerungen verwenden \[Seite 103\]](#)

## Syntax und Semantik

Die folgende Tabelle beschreibt die Syntax, mit der alle fünf Fälle der Eingabesteuerung dargestellt werden.

Eingabe	Ableitung
Name/Wert	Feldname [ Länge] [Index] "=" Wert ["," Name/Wert]
Feldname	{ "~"   "_"   "-" } Buchstabe { Buchstabe   Ziffer   "_"   "~"   "-" }
Länge	":" numKonstante
Index	"["numKonstante t "]"
Wert	URL-kodierte Zeichenfolge
NumKonstante	Ziffer { Ziffer }
Ziffer	"0".. "9"
Buchstabe	"A".. "Z"   "a".. "z"

Beachten Sie, daß eckige Klammern "[" ein wahlfreies Attribut angeben.

"Länge" und "Index" haben die folgende Semantik:

Länge	Beschränkt die Anzahl der Zeichen, die aus dem "Wert"-Teil entnommen und in das entsprechende R/3-Dynprofeld gestellt werden, auf <i>NumKonstante</i> Zeichen. Alle weiteren Zeichen gehen verloren, es sei denn, dieses Feld ist ein Feld für mehrere Werte und entspricht einem Step-Loop und wird im Append-Modus ausgefüllt (siehe "Index" unten). In diesem Fall werden alle zusätzlichen Zeichen automatisch in die nächste Zeile des Step-Loops umgebrochen.
Index	Wenn Index angegeben wird, wird das Feld zu einem Feld mit mehreren Werten oder einem "Step-Loop". <i>NumKonstante</i> legt fest, zu welcher Zeile der Wert gehört. Wenn keine <i>NumKonstante</i> angegeben wird (Klammernpaar ohne Inhalt), wird der Wert im Append-Modus hinzugefügt (d.h. der Wert wird am Ende der bereits angegebenen Werte hinzugefügt).



Einige Beispiele zur Klärung:

**firstName=Heidi**

legt das Feld "firstName" im Bild auf den Wert "Heidi" fest.

**mail-text[12]=Viele+Grüße**

## Syntax und Semantik

legt die Step-Loop-Zeile 12 auf den Wert "Viele Grüße" fest. Beachten Sie, daß "+" die URL-Codierung für ein Leerzeichen ist.

```
mail-text[]="Heidi+Becker"
```

fügt den Text "Heidi Becker" am Ende der Zeilen hinzu, die gegenwärtig im Feld "mail-text" definiert sind. Wenn mail-text bereits den folgenden Text enthält:

```
mail-text[1] = "Hallo Peter"
mail-text[2] = "Ich komme um 19 Uhr"
mail-text[3] = "Viele Grüße"
```

sieht das Ergebnis folgendermaßen aus:

```
mail-text[1] = "Hallo Peter"
mail-text[2] = "Ich komme um 19 Uhr"
mail-text[3] = "Viele Grüße"
mail-text[4] = "Heidi Becker"
```

```
lastName:4=Becker
```

legt das Dynprofeld "lastName" auf den Wert "Beck" fest.

```
mail-text:10[2]=Ich+komme+um+19+Uhr
```

legt die Step-Loop-Zeile 2 auf den Wert "Ich komme " fest, während "um 19 Uhr" gelöscht wird. Die Zeile wird hinter dem letzten Leerzeichen abgeschnitten, wenn das folgende Wort nicht vollständig in die angegebene Begrenzung paßt. Da ein Indexwert angegeben ist, wird der Teil "um 19 Uhr" nicht, wie im nächsten Beispiel, in die nächste Zeile umgebrochen.

```
mail-text:10[]=Ich+komme+um+19+Uhr
```

Wenn der Briefftext gegenwärtig mail-text[1]="Hallo! " lautet, sieht das Ergebnis folgendermaßen aus:

```
mail-text[1] = "Hallo! "
mail-text[2] = "Ich komme "
mail-text[3] = "um 19 Uhr"
```

Der automatische Wortumbruch findet statt, da kein Indexwert ("[]") festgelegt wurde.

```
mail-text:10[]
```

```
=Ich+komme+um+19+Uhr%d%a%d%aViele+Grüße%d%aHeidi+Becker
```

Wenn der Briefftext gegenwärtig aus mail-text[1]="Hallo!" besteht, wird dieser Text wie folgt dargestellt:

```
mail-text[1] = "Hallo! "
mail-text[2] = "Ich komme "
mail-text[3] = "um 19 Uhr"
mail-text[4] = ""
```

```
mail-text[5] = "Viele "  
mail-text[6] = "Grüße"  
mail-text[7] = "Heidi "  
mail-text[8] = "Becker"
```

Mit den `<textarea>`-Steuerungen für Zeilenumbruch/Neue Zeile werden die Zeilen genauso abgeschnitten bzw. die Wörter so umgebrochen, wie dies beim Erreichen der Eingabelängenbegrenzung der Fall ist.

```
firstName=Heidi,LastName=Becker
```

Mit dieser Syntax wird das Feld `firstName` auf den Wert `Heidi` festgelegt, das Feld `lastName` auf den Wert `Becker`.

---

**Mehrere Felder von einer HTML-Steuerung übergeben**

## Mehrere Felder von einer HTML-Steuerung übergeben

Ein typisches Beispiel ist ein Dynpro, in dem sich zwei Eingabefelder befinden: ein Feld für die Menge und ein weiteres Feld für die Einheit. Gleichzeitig ist nur eine Auswahlliste auf der HTML-Seite vorhanden, aus der beide Werte in einem Schritt ausgewählt werden müssen:

```
<select name="unit=ml,amount=">
  <option value="10">10 Milliliter
  <option value="25">25 Milliliter
  <option value="50">50 Milliliter
</select>
...
```

Es ist **nicht** möglich, den folgenden Code zu verwenden, um zwei Felder gleichzeitig zu übergeben. Die Verkettung von Namen-/Wertepaaren ist also nur im "Name"-Teil einer Steuerung gültig:

```
...
<select name="productno=4711,amount=">
  <option value="10,unit=ml">10 Milliliter
  <option value="25,unit=ml">25 Milliliter
  <option value="50,unit=ml">50 Milliliter
</select>
...
```

## <textarea>-Steuerungen verwenden

Wenn Sie eine <textarea>-Steuerung verwenden möchten, um Eingaben für einen R/3-Step-Loop zu erhalten, können Sie die folgende Anweisung in Ihr HTML-Template aufnehmen:

```
<textarea name="mail-text:80[]" cols="80"></textarea>
```

Diese Anweisung stellt sicher, daß maximal 80 Zeichen pro Step-Loop in das Bild gestellt werden.

Hiermit erhalten Sie den aktuellen Inhalt des Step-Loops im Bild als Standardwert für die <textarea>-Steuerung:

```
<textarea name="mail-text:80[]" cols="80">`repeat with r in mail-  
text;  
write (r, "\r\n"); end`</textarea>
```

Fügen Sie keine zusätzlichen Zeilenumbrüche ein, da dies in einigen Web-Browsern zu unbeabsichtigten Ergebnissen führen kann.



Stellen Sie sicher, daß in Ihrem Bild genügend Step-Loop-Zeilen für alle Werte des entsprechenden Feldes vorhanden sind. Andernfalls tritt ein Fehler auf.