ShopPilot®
by IBO Internet Software

Shop Server Pages

SSP 3.0

für EP 4.xx

Referenz für Programmierer

IBO Internet Software e.K.

Wehrstr. 6
41199 Mönchengladbach

Tel. +49 (0) 2166 9989 530
Fax +49 (0) 2166 9989 535

hop@shoppilot.de
www.shoppilot.de

Copyright 2018 by IBO Internet Software

1 Verzeichnis

1 Verzeichnis 2
2 Änderungshistorie 5
3 SSP-Referenz 6
3.1 Vorbemerkung 6
3.2 Einleitung 6
3.3 Aktive Elemente für ShopPilot Vorlagen 7
3.3.1 Identifier 7
3.3.2 Macros 7
3.3.2.1 loopitem 7
3.3.2.2 enditem 8
3.3.2.3 include 8
3.3.2.4 module 8
3.3.3 SSP-Identifier 9
3.4 SSP Sprachdefinition 11
3.4.1 Datentypen 11
3.4.2 Eingebaute read only Variablen 11
3.5 API-Funktionen 12
3.5.1 Hilfsfunktionen 12
3.5.2 Ausgabe des Identifiers 15
3.5.3 Variablen im Warenkorb speichern 15
3.5.4 Eingebettete Vorlagen 16
3.5.4.1 Aufbau des Suchstrings für search und cserach 17
3.5.5 Tabs generieren 17
3.5.6 Funktionen für den Zugriff auf Variablen 17
3.5.6.1 Artikel 18
3.5.6.2 Artikelvarianten 20
3.5.6.3 Seiten 20
3.5.6.4 Listen 21
3.5.6.5 Warenkorb 21
3.5.6.6 Kundendaten 24
3.5.6.7 Haendlerdaten 25
3.5.6.8 Subkatalogdaten 25
3.5.6.9 Abgeschickte Formulare 26
3.5.7 Anwendung von SSP-Identifiern 27
3.5.7.1 Identifier in SSP-Funktionen 27
3.5.7.2 Mehrfachbenutzung 27
3.5.8 Callbacks 29
3.5.8.1 invoice_addpos 29
3.5.8.2 invoice_addpos_final 29
3.5.8.3 @text mailtext = invoice_addpos_mail() 30
3.5.8.4 num <anzahl> begin_read_template() 30
3.5.8.5 after_read_template(num <key>, text <file>, num <count>) 30
3.5.8.6 @templates_new modify_search_templates(<text> templ_notfound, <text> templ_found, <text> templ_found_asn) 30
3.5.8.7 text <fuereinenartikel> set_simpleradio_text(text <artikelnr>, text <beschreibung>, text <preis>, num <index>) 31
3.5.8.8 text <jscode> js_placecart_insertion() 32
3.5.8.9 num <price> modify_int_price(num <price>, num <index>) 33
3.5.8.10 num <price> modify_db_price(num <price>, text <artnr>) 33
3.5.8.11 modify_price_quantity_scale 33
3.5.8.12 Shopcontol Callback 34
3.5.8.13 get_catalog_markup 34
3.5.8.14 exclude_zahlart 35

Copyright 2018 by IBO Internet Software

3.5.8.15 AddSQLCondition 35
3.5.8.16 modify_search_templates 36
3.5.8.17 searchdb_limit_results 36
3.5.8.18 check_order 36
3.5.8.19 check_udc_action 37
3.5.8.20 modify_mailtext 39
3.5.8.21 change_mydelwk 39
3.5.8.22 on_syserr 40
3.5.8.23 check_write_user 40

3.6 Automatisch eingebundene Standard-ipls 44
3.6.1.1 stdall.ipl 44
3.6.1.2 stdcart.ipl 44
3.6.1.3 stdplacecart.ipl 44

3.7 Anwendungsbeispiele 45
3.7.1 Artikelspezifischer Rabatt 45
3.7.2 Staffelrabatt nach Warenwert 47
3.7.3 Artikelabhängige Versandkosten 50
3.7.4 Zusätzliche Artikelbilder 52
3.7.5 Seitenhierarchie als Pfad 52
3.7.6 Verwandte Artikel aus Suchergebnis 53
3.7.7 Angabe des Rechnungsdatums 53
3.7.8 Artikel gruppieren 53
3.7.9 Datei-Upload 55
3.7.10 XTabs - Erweiterte Tabs 56
3.7.10.1 Definieren der Tabs 56
Ausgeben der Tabs 57
3.7.11 Produktkonfigurator 58
3.7.11.1 Prinzip eines einfachen Produktkonfigurators 58
3.7.12 Anzahl der auszugebenden Artikeln pro Seite 60
3.7.13 Artikel sortieren 61

3.8 Datenbankzugriff 62
3.8.1 Grundsätzliche Vorgehensweise 63
3.8.1.1 Anwendung von begin_read_template 63
3.8.1.2 Daten einlesen 63
3.8.1.3 Daten anzeigen 64
3.8.1.4 Daten ändern 64
3.8.1.5 Daten einfügen 65
3.8.1.6 Daten löschen 65
3.8.2 Funktionen für Shopobjekte 65
3.8.2.1 Artikel 65
3.8.2.1.1 db_query_item 65
3.8.2.1.2 db_update_item 66
3.8.2.1.3 db_insert_item 66
3.8.2.1.4 db_delete_item 66
3.8.2.2 Subkatalog Artikel 66
3.8.2.2.1 db_query_iscat 67
3.8.2.2.2 db_update_iscat 67
3.8.2.2.3 db_insert_iscat 67
3.8.2.2.4 db_delete_iscat 67
3.8.2.3 Seiten 67
3.8.2.3.1 db_query_page 67
3.8.2.3.2 db_update_page 67
3.8.2.4 Bestellungen 67
3.8.2.4.1 db_query_orders 67
3.8.2.4.2 db_update_orders 68
3.8.2.4.3 db_delete_orders 68
3.8.2.5 Bestellpositionen 68
3.8.2.5.1 db_query_oitems 68
3.8.2.6 Kunden/Benutzer 68
3.8.2.6.1 db_query_usr 68

Copyright 2018 by IBO Internet Software

3.8.2.6.2 db_update_usr 69
3.8.2.6.3 db_delete_usr 69
3.8.3 Freie Datenbankabfragen mit SQL 69
3.8.3.1 readSQLData 69
3.8.3.2 get_last_sqlerr 70
3.8.4 Anwendungsregeln anhand eines Beispiels 70
3.8.4.1 Übersicht 70
3.8.4.2 News-Übersicht 71
3.8.4.3 Einen Eintrag editieren 72
3.8.4.4 Einen neuen Eintrag anlegen 73
3.8.4.5 Die Reihenfolge der Einträge bestimmen 74
3.9 Nutzung von Ajax/Json 76
3.9.1 Aufbau 76
3.9.2 Der Kern 77
3.9.3 Hinzfügen einer neuen Ajaxanwendung 78

Copyright 2018 by IBO Internet Software

2 Änderungshistorie

Datum	Version	Änderungen
1.10.2002	1.01	Initiale Version
9.04.2003	1.02	Kundendaten auslesen **ssp::get_var_user** Warenkorbsumme **ssp::get_wkprodsum** Beispiel: **Verwandte Artikel aus Suchergebnis**
15.8.2003	1.03	**ssp::embedded** spmacro jetzt auch auf Meldungsseiten und Navigation möglich
30.12.2003	1.04	ssp::get_var_varticle – Werte von Varianten auslesen Gesamtsumme **ssp::get_wktotalsum**
10.02.2004	1.05	**ssp::get_var_merch** erweitert Formularwerte auslesen: **ssp::get_var_form** neu SSP-Identifier mehrfach für einen Artikel benutzen Identifier in SSP-Funktionen
15.10.2004	2.10	Erweiterung für Datenbankzugriff **ssp::embedded** neuer Type "show" benötigt mindestens 2.10.03
30.12.2004	2.10	Zugriffsfunktionen für Zahlungsart und Lieferzone, benötigt mindestens 2.10.04
20.01.2005	2.11	Neue Funktionen: get_cart_rabatt() get_cart_myrabatt(), **get_var_cart()** jetzt auch Parameter image möglich
28.05.2005	2.20	xtabs, stdcart.ipl
05.08.2005	2.20.01d	**invoice_addpos_final()**
28.05.2006	2.50	**Erweiterung __xxspath__ get_var_minder(), stdplacecart.ipl**
06.07.2006	2.50	**Erweiterung stdall.ipl, Callbacks für Preismodifikation** modify_int_price, modify_db_price
26.02.2007	2.80	set_var_form(), modify_price_quantity_scale(), db_delete_usr(), Callback: ShopControl::printateob(), Callback: invoice_addpos_final ssp::get_cart_versand() ssp::get_cart_zahlartbetrag () Callback: get_catalog_markup()
21.01.2008	2.80.05	qform(), qdb(), pageurl(), is_netto(), ssp::getChildCategoriesRecursive()
1.07.2008	3.00.00	xcreated und xmodified für Artikel und Varianten SSP Identifier mit Parameter möglich __modul::funktion(parameter)__ Neue Funktion Callback: modify_search_templates Speichern und Lesen von beliebigen Werten im Warenkorb: ssp::cart_write_var() und ssp::cart_read_var()
28.10.2008	3.00.01	Neue Funktion Callback::exclude_zahlart Zahlarten ausschließen (nicht anzeigen)
14.1.2009	3.00.03	Neue Funktion Callback:: AddSQLCondition Artikelabfrage aufgrund einer zusätzlichen Bedingung verändern
20.3.2009	3.01.01	neue Funktion get_var_list zum Auslesen von Listen
10.6.2009	3.02.01	get_var_cart kann mit Parameter 'pgid' die Seitennummer für die Detailseite zurückgeben
28.1.2010	3.02.02	ssp::get_all_zones() gibt eine Array mit den Zonenbezeichnungen für alle Länder zurück, Parameter für SSP-Identifier können auch "," und "-" enthalten.
10.12.2010	3.02.05	ssp::get_all_zones() erweitert, js_taxproplist() erstellt ein Javascript Array mit den steuerlichen Eigenschaften der Lieferzonen. Das Callback mydelwk() ermöglicht zu Verhindern, dass bestimmte Artikelpositionen aus dem Warenkorb gelöscht werden.
26.12.2011	3.04.01	set_var_cart(); $ssp::KZGS; Callback::modify_detail_template()
03.12.2012	3.05.02	get_cmdln_arg(),Callback:: override_template(), get_merch_conf_values()
29.07.2016	3.08.01	Callback::override_zahlartradio_item()

Copyright 2018 by IBO Internet Software

3 SSP-Referenz

3.1 Vorbemerkung

Die Shop Server Pages stehen erst am Anfang der Entwicklung. Wesentliche Teile dieser Dokumentation sind stabil und umfassend getestet. Es kann jedoch bei einzelnen Funktion oder Verfahrensweisen noch zu Änderungen kommen.

Für die sachgerechte Anwendung sind grundlegende Kenntnisse der Programmierung mit Perl oder einer verwandten Programmiersprache (z.B. PHP) hilfreich.

Es wird ausdrücklich davor gewarnt Zugriffe auf interne Funktionen oder Variablen der ShopPilot Scripte durchzuführen, da diese zur Fehlfunktion der gesamten Software führen kann. Wir sind bemüht die hier als Schnittstelle aufgeführten Funktionen so auszubauen, dass damit möglichst viele Anwendungsfälle abgedeckt werden.

Aber genug der Warnungen. Im Grunde ist die Anwendung von SSP sehr einfach. Und vielfach konnten wir schon erleben, dass auch bisherige Nicht-Programmierer SSP nach kurzer Zeit selbständig zur Verbesserung ihres Shops einsetzen konnten. Wichtig ist, dass man Sachen einfach mal spielerisch (in LiveView) ausprobiert, um ein Gefühl dafür zu bekommen. Das Verständnis wächst dann automatisch.

3.2 Einleitung

Ein wesentlicher Punkt für den Erfolg eines Shops ist die den Produkten, den Kunden und der Firma angemessene Darstellung des Angebots. Missstände in diesem Bereich führen oft zu nicht den Erwartungen entsprechenden Ergebnissen. Das Shop-System sollte hier alle Möglichkeiten unterstützen und keine Einschränkungen haben. Die Darstellung der Shop-Seiten durch ShopPilot beruht auf Vorlagen. Vorlagen sind normale HTML-Dokumente, die je nach Anforderungen um ShopPilot spezifische Elemente erweitert werden können.

Diese Elemente sind:

- Identifier
- Makros
- SSP-Identifier / Shop Server Pages (SSP) Code

Identifier (Platzhalter) erlauben auch ohne Programmierkenntnisse die freie Gestaltung einer Shop-Seite. Makros bieten zusätzliche Möglichkeiten, wie die automatische Wiederholung (Iterationen) von Seitenausschnitten und andere häufig benötigte Funktionen. Durch SSP ist schließlich eine funktionale Änderung und Erweiterung möglich. Mit SSP ist es z.B. möglich eigene Identifier (SSP-Identifier) mit eigener Funktionalität zu erstellen und diese dann wie die eingebauten zu verwenden.

Identifier wirken in einer HTML-Seite als Platzhalter. Sie werden durch entsprechende Daten ersetzt. Diese Daten können aus Artikel- oder Seiteniteration stammen oder sind bei den Standard-Identifiern global verfügbare Daten (z.B. Suchfunktion = __suchen__)

Beispiel für benutzerdef. Identifier: `__STRUMPFGROESSE__` → `36`

Identifier werden direkt in den HTML-Text eingegeben, hierzu kann jeder beliebige HTML-Editor dienen. Für die Darstellung von Texten sind auf einfache Weise beliebige Formatierungen der Identifier und damit der ersetzten Texte möglich. Neben einer Anzahl von eingebauten Identifiern kann man zusätzliche Identifier im ShopPilot Work Center über die Anlage von benutzerdefinierte Felder definieren.

Makros dienen der Steuerung der Darstellung. Sie werden durch einen HTML-Kommentar eingebettet.
Beispiel Iteration für 4 Artikel:

```
<!--spmacro:loopitem4-->Größe: __STRUMPFGROESSE__<BR><!--spmacro:enditem-->
```

→

```
Größe: 36
Größe: 37
Größe: 34
Größe: 38
```

Copyright 2017 by IBO Internet Software

Mit Hilfe eines speziellen Makros (spmacro:module) lassen Sich **Shop Server Pages** definieren. Hiermit können komplexe Funktionen realisiert werden. Durch ein Callback-Modul kann die Verarbeitung im Kern der ShopPilot Scripte modifiziert werden.

3.3 Aktive Elemente für ShopPilot Vorlagen

3.3.1 Identifier

Identifier werden während der Laufzeit ersetzt.
Man unterscheidet zwischen aufzählungsabhängigen Identifiern und konstanten Identifiern.
Aufzählungsabhängig sind z.B. Identifier für Artikel, Seiten oder Artikel im Warenkorb. Hier wird davon ausgegangen, dass ein oder mehrere Artikel dargestellt werden sollen.
Die Reihenfolge der Identifiern einer Vorlage braucht nicht gleich zu sein, obwohl dies oft der Fall ist. Der Beginn eines neuen Artikels wird daran erkannt, dass einer der verwendeten Identifier zum zweiten mal innerhalb der Vorlage auftaucht. In diesem Fall wird der Artikelzähler (Index) erhöht. Gleichfalls ergibt sich daraus die Regel, dass ein Identifier in einer Aufzählung nicht zweimal für den gleichen Artikel verwendet werden darf.
Eine Ausnahme von dieser Regel ist über Verwendung von SSP-Identifern mit der Rückgabe "$ssp::noinc" möglich.

3.3.2 Macros

3.3.2.1 loopitem

loopitem[<zeilen>[.<spalten>]]

Anwendung: Artikelseiten, Warenkorb, Bestellformular, Bestelldatenvorlage, HTML-Mails, Verzeichnisseiten, wenn diese Artikel enthalten

Das Argument <zeilen> ist optional und gibt an, wie oft eine Wiederholung pro Seite maximal durchgeführt werden soll. Fehlt <zeilen> wird so oft wiederholt, wie Artikel vorhanden sind. Möchte man mehrere Artikel zusammengefasst darstellen, so kann man mit einem Dezimalpunkt abgetrennt eine weitere Zahl angeben. Diese steht für die Spalten. Die Gesamtzahl der auf einer Seite dargestellten Artikel entspricht dem Produkt aus <zeilen> mal <spalten>. Obwohl hier mit Spalten bezeichnet, kann der Parameter <spalten> auch für die Darstellung in zwei unterschiedlich formatierten Zeilen sorgen, z.B. im Warenkorb: Zwei Zeilen mit unterschiedlichem Hintergrund zur besseren Lesbarkeit. Dies hängt ausschließlich vom Aufbau des zu wiederholenden Tabellenfragments ab.

Beispiel mit drei Artikeln:

```
<table>
<!--spmacro:loopitem3-->
<tr><td>
Dies ist eine Artikelnummer: __artnr__
</td></tr>
<!--spmacro:enditem-->
</table>
```

Dies ist eine Artikelnummer: 003435
Dies ist eine Artikelnummer: 003437
Dies ist eine Artikelnummer: 0a1434

Beispiel mit 4 Artikel, davon zwei in einer Zeile.

```
<table>
<!--spmacro:loopitem4.2-->
<tr><td>
Dies ist eine Artikelnummer: __artnr__
</td><td>
Noch eine: __artnr__
</td></tr>
<!--spmacro:enditem-->
</table>
```

Dies ist eine Artikelnummer: 003435 Noch eine: 003437
Dies ist eine Artikelnummer: 0a1434 Noch eine: 0a1435

Copyright 2017 by IBO Internet Software

3.3.2.2 enditem

enditem

Anwendung: Artikelseiten, Warenkorb, Bestellformular, Bestelldatenvorlage, HTML-Mails

Zeigt das Ende eines Wiederholungsbereiches an.

3.3.2.3 include

include(<filename>)

Anwendung: Alle Seiten

<filename> ist der relative Pfad zu einer Datei, die an der entsprechenden Position in den Text der Vorlage eingefügt werden soll.

Es werden zwei verschiedene Dateitypen anhand der Endung unterschieden.
*.ipl (Include Perl-Datei) - der relative Pfad beginnt hier im cgi-Verzeichnis, solche Dateien beinhalten in erster Linie Perl-Module. Durch die Ablage im cgi-Verzeichnis ist ihr Inhalt vor Zugriffen von außen geschützt.

*.htm, *.html, *.htm? (HTML-Datei) - der relative Pfad beginnt hier im Catalog-Verzeichnis, solche Dateien beinhalten in erster Linie darstellbaren HTML-Code.

Beispiel:

```
<!--spmacro:include(header.html)-->
```
oder
```
<!--spmacro:include(scripts/mein_modul.ipl)-->
```

3.3.2.4 include mit Bedingung

include(<filename>,<condition>)

Anwendung: Alle Seiten

Dieses include arbeitet wie das normale include, wenn condition wahr ist. Also einen Wert <> 0 hat.
Ist condition hingegen 0 oder leer. wird das include nicht ausgeführt. condition kann eine Variable oder Funktion sein. Bei einer Funktion wird nur der Funktionsname angegeben, ohne Klammern und Argumente.

Beispeil nur bei Liveview-Betrieb inkludieren.:
```
<!--spmacro:include(scripts/mein_modul.ipl,$ssp::liveview)-->
```

Beispiel nur für Händler1 inkludieren.:
```
sub  IstDasHaendler1 {
   …
   return 1;
}
<!--spmacro:include(scripts/mein_modul.ipl,IstDasHaendler1)-->
```

Funktionen und Variablen müssen vor der Benutzung deklariert und definiert sein.

3.3.2.5 module

module(<name>) <code>

Anwendung: Alle Seiten

<name> gibt den Namen an unter dem das Modul intern gespeichert wird.
Er dient gleichzeitig zum Aufruf von Elementen aus dem Modul.
Das nachfolgende Element <code> enthält den gesamten Code des Moduls.

Beispiel:

```
<!--spmacro:module(print)
sub hello{
```

Copyright 2017 by IBO Internet Software

```
	main::mprint("hello world!");
}
-->

__print::hello__
```

hello world!

3.3.3 SSP-Identifier

Der Aufruf von Modul-Funktionen im HTML-Text funktioniert analog zu den Identifier. Sie geben das Modul und den Funktionsnamen in doppelten Unterstrichen an. Modulname und Funktionsname werden durch zwei Doppelpunkte getrennt.

__<modul>::<funktion>__
oder
__<modul>::<funktion>(paramter)__

Gelangt der ShopPilot Kern beim Interpretieren der Vorlagendatei an eine solche Stelle, wird die entsprechende Funktion im Modul aufgerufen und der Identifier durch einen Ausgabetext (der oben mit mprint zur Verfügung gestellt wird) ersetzt.

Wird die Funktion innerhalb einer Artikelauflistung oder loopiten-Makos benutzt, werden bei normalen Artikeln zwei Parameter an die aufgerufene Funktion übergeben:

Artikelindex (0..Anzahl-1)
AnzahlArtikel (Anzahl –1)

Die Parameter werden der Modulfunktion bei Ihrem Aufruf zur Verfügung gestellt (Im Beispiel $index erhält $_[0] und $anzahl erhält $_[1]).

Diese Parameter haben zwei Aufgaben:

1. Sie ermöglichen den Zugriff auf die Attribute des aktuellen Artikels, indem der entsprechende Index übergeben wird.
2. Bei Vorlagen, die mehr Artikelzeilen als darstellbare Artikel enthalten, können die überzähligen Identifier ignoriert werden. Dies geschieht mit der Bedingung (Artikelindex <= AnzahlArtikel), nur wenn diese Bedingung wahr ist erfolgen entsprechende Aktion zu Ausgabe eines Ersetzungstextes, im anderen Fall wird der Funktions-Identifier entfernt (durch einen Leerstring ersetzt).

Beispiel:

```
<!--spmacro:module(print)
sub index {
	my $index = $_[0];
	my $anzahl =  $_[1];
	if ($index <= $anzahl) {
		my $preis = ssp::get_var_article('preis',$index);
		my $out = sprintf("Dies ist Artikel %d zum Preis von %5.2d von
	insgesamt %d Artikeln\n",$index + 1,$preis,$anzahl + 1);
		main::mprint($out);
	}
	return 0;
}
-->
__print::index__
```

Dies ist Artikel 1 zum Preis von 22.40 von insgesamt 7 Artikeln.

Bei Variantenartikeln kommen noch zwei Parameter für die Varianten hinzu, so dass insgesamt 4 Parameter übergeben werden. Die Bedeutung ist analog zu den ersten beiden Parametern, nur dass es hier um eine Auflistung von Varianten zu einem Artikel geht.

Artikelindex (0..Anzahl-1)
Anzahl Artikel (Anzahl –1)
Variantenindex (0..Anzahl-1)

Copyright 2017 by IBO Internet Software

Anzahl Varianten (Anzahl –1)

```
<--spmacro:module(print)
sub vindex {
       my $index   =  $_[0];
       my $anzahl  =  $_[1];
       my $vindex  =  $_[2];
       my $vanzahl =  $_[3];
       if ($vindex <= $vanzahl) {
my $preis ssp::get_var_varticle('preis',$vindex);
my $out = sprintf("Dies ist Variante %d zum Preis von %d von insgesamt %d Varianten
\n",$vindex +1 ,$preis,$vanzahl +1);
       main::mprint($out);
       }
       return 0;
}
```

Dies ist Variante 1 zum Preis von 12.40 von insgesamt 3 Varianten.

Zusätzlicher Parameter. Es ist auch möglich einen zusätzlichen Parameter zu übergeben. Der Parameter wird als Konstante in runden Klammern hinter die Funktion geschrieben und steht beim Funktionsaufruf als erster Parameter zur Verfügung. Die anderen Parameter verschieben sich entsprechend. Als mögliche Zeichen für den Parameterstring sind derzeit Zahlen und Buchstaben "," und "-" zulässig.

Beispiel:

```
__print::hello(Hallo)__


<--spmacro:module(print)

sub hello {
       my $parameter=  $_[0];
       my $index   =  $_[1];
       my $anzahl  =  $_[2];
       if ($index <= $anzahl) {
              main::mprint($parameter);
       }
}
-->
```

Der SSP-Identifier `__print::hello(Hallo)__` wird durch "Hallo" ersetzt.

Copyright 2017 by IBO Internet Software

3.4 SSP Sprachdefinition

3.4.1 Datentypen

Die Scriptsprache ist von Perl abgeleitet. Perl kennt grundsätzlich nur den Datentyp String. Mit diesem Datentyp werden alle anderen Datentypen dargestellt. Um die Funktionsparameter jedoch richtig zu interpretieren ist das Wissen um den Inhalt notwendig. Wir wollen deshalb folgende Konventionen verwenden. Jeder Parameter erhält in der folgenden Beschreibung einen Vorsatz der die Verwendung anzeigt.

@ Liste eines der nachfolgenden Elemente. Eine Liste aus "text"- Konstanten sieht z.B. so aus: ("abc","def,"ghj")
text beliebige Zeichenkette (Beispiel "sdfsfs2352fd")
num Numerischer Wert, auch als Dezimalzahl. Achtung Dezimaltrenner ist der Punkt. (Beispiel 3 , 3.45 , 3237723.45)
flag Boolescher Wert (Beispiel 1 , 0)
ref Referenz enthält den Namen einer Variablen
int ganzzahliger Wert
hash Ein Datenfeld, deren einzelne Werte über einen Schlüssel vom Typ text abrufbar sind. (Beispiel %feld = (key1 => ‚Wert 1', key2 => ‚Wert 2'). Abruf des Wertes für key2: $feld{key2}
hashref Referenz auf eine Variable vom Typ hash.

3.4.2 Eingebaute read only Variablen

Diese Variablen dürfen nur gelesen werden (read only), eine Veränderung ist nicht zulässig.

$ssp::xxpath (read only)
Pfad zum cgi-Script unabhängig von lokalem oder Serverbetrieb

$ssp::xxspath (read only)
Pfad zum cgi-Script über SSL (nur für Serverbetrieb)

$ssp::xxsession (read only)
Aktuelle uniqe Session-ID des Besuchers

Diese Variablen ermöglichen per SSP korrekte Links auf Funktion im HTML-Code auszugeben.

```
main::mprint("$ssp::xxpath?show20,$ssp::xxsession");  # Aufruf von Seite 20
```

$ssp::host (read only)
Der Hostname oder die IP-Adresse des Besuchers

$ssp::pgid (read only)
Die aufgerufene Seitennummer oder der Suchstring

$ssp::liveview (read only)
Ist der Wert nicht 0, wird LiveView benutzt. Alle Formulare mit Dateiupload funktionieren nicht mit dem LiveView Server. Diese Variable kann benutzt werden, um hier eine Fehlermeldung auszugeben.

$ssp::catalog_home (read only)
Relativer Dateipfad zum Katalogverzeichnis. Wird beim Öffnen von Vorlagedateien im Katalogverzeichnis benutzt.

$ssp::catalog_abs (read only)
Absolute URL zum Katalogverzeichnis, zur Ausgabe von z.B. Bild-URL´s in ausgegebenen HTML-Seiten.

$ssp::undefined (read only)
Wert der Funktionsrückgabe, wenn kein Wert ermittelt werden kann.

$ssp::noinc (read only)
Rückgabewert einer SSP-Identifier Funktion, der verhindert dass der index für die (Artikel)Position hochgezählt wird.

$ssp::KZGS (read only)
Enthält das Kennzeichen für Artikel, die Gutscheine sind. Dies sind die ersten Buchstaben der Artikelnummer (Gutscheinnummer).

Copyright 2017 by IBO Internet Software

3.5 API-Funktionen

3.5.1 Hilfsfunktionen

<num> ret = main::extr(text <value>)
Extract Number
Ermittelt die erste Zahl im String.
Beispiel: "ab 20" -> 20

<text> ret = main::fp(num <value>)
Format Price
Gibt den Wert gemäss dem eingestellten Währungsformat aus.
Beispiel: 12.6 -> 12,60 EUR
8,70 CHF

ssp::trace(text <value>)
Trace
Erzeugt eine Ausgabezeile in der Datei shoplog.txt. Die Zeile beginnt mit "T,,". Danach folgt der als Argument mitgegebene Text. Enthält der Text Zeilenumbrüche, werden diese herausgefiltert, so dass immer nur eine einzige Ausgabezeile generiert wird.
Diese Funktion dient zur Ausgabe von Zwischenergebnissen und ist für die Diagnose und Fehlersuche gedacht. Bei produktiven Shops sollte die Funktion entfernt werden, um die Logdatei nicht unnötig aufzublähen.

text key = ssp::get_key()
Gibt die Session-ID des angemeldeten Benutzers zurück.

ssp::set_key(text <key>)
Setzt die Session-ID des angemeldeten Benutzers auf den Wert <key>. Achtung: Die Anwendung dieser Funktion kann zu einem Verlust des Warenkorbs führen.

text mailprog = ssp::get_mailprog()
Gibt den Aufrufpfad des verwendeten Mail-Programms zurück.
Dies ist derselbe Wert, der in der project.ini unter sendmail eingetragen ist.

text quoted = ssp::qform(text <unquoted>)
Der Wert des Parameters <unquotet> wird zur Ausgabe in HTML gequotet.
Die Zeichen <, >, ’ und “ werden durch die entsprechenden HTML-Entites ersetzt.

text quoted = ssp::qdb(text <unquoted>)
Der Wert des Parameters <value> wird zur Ausgabe als Parameter in einer SQL-Abfrage gequotet.
Die Zeichen ’ und \ werden durch ’’ bzw. \\ ersetzt.

<hash> ssp::getCurrentPageInfo()
Gibt ein Hash zurück, das Informationen zur angezeigten Seite im Shop enthält. Die einzelnen Informationen

- pagetype <int> Seitentyp
- pageid <int> Seitennummer
- msgtype <int> Seitentyp für Meldungsseiten
- rawpageid <text> Seitennummer oder Suchstring bei Artikelsuche (z.B "searchXYZ:11111111" wenn in allen Feldern nach "XYZ" gesucht wurde)

Folgende Werte für pagetype sind möglich:

-1 (FEHLER: unbekannte Seite)
0 (Interne Seite)
1 (Verzeichnisseite)
2 (Artikelseite)
3 (Detailseite)
11 (showwk Warenkorb anzeigen)
12 (delwk Warenkorb löschen)
13 (placecart Warenkorb befüllen)
14 (showcart Warenkorb anzeigen)
15 (chgcart ändern)
16 (chgzahl Zahlungsart)
17 (chgzone Versandzone)
18 (showidfr Bestellhinweise)
19 (showidfr Bestellformular)
20 (impressum)

Folgende Werte für msgtype sind möglich:

1 Fehlermeldung

Copyright 2017 by IBO Internet Software

2 Danke-Seite
6 Keine Favoriten

Folgende Seitennummern (pageid) sind möglich:

0		Verzeichnisseite
1+		Interne Seite (pagetype == 0)
	1:	Login OK
	2:	Login Fehler
	3:	Login Eingang
	4:	Bestellhinweise
	5:	Favoriten
	6:	Warenkorb
	7:	Suchen
	8:	Meldungen
	9:	(nicht vergeben)
10+		Verzeichnisseite | Artikelseite | Detailseite
1200000000+		Informationsseite
1300000000+		Formular

Beispiel zur Verwendung:

```
sub test {
	my %pageinfo = ssp::getCurrentPageInfo();
	if ($pageinfo{pagetype} == 3) {
		main::mprint("Dies ist eine Artikeldetailseite");
	}
}
```

@text ssp::getTaxes()
Gibt ein Feld mit den Umsatzsteuersätzen zurück.
Der erste Wert ist die UST1, der zweite die UST2 und der dritte die UST3.

Beispiel zur Verwendung:

```
sub printtaxes {
	my @taxes = ssp::getTaxes();
	my $out = "";
	for (my $i = 0; $i < scalar(@taxes); $i++) {
		$out .= "Umsatzsteuer Nummer $i beträgt $taxes[$i]\%<br>";
	}
	main::mprint($out);
}
```

text ssp::pageurl(num <without_sessionid>)
Gibt eine gegen XSS gequotete URL der aktuellen Seite als String zurück. Es besteht die Möglichkeit den String mit oder ohne SessionID zu erhalten. Ist der Parameter without_sessionid gesetzt (=1), wird die SessionID aus der URL entfernt.

ssp::getChildCategoriesRecursive(num <catid>, hashref <pHashOut>)
Füllt die angegebene Hash-Array-Referenz <pHashOut> mit den Seitennummern der unterhalb der Seite <catid> liegenden Seiten sowie der Seite <catid> selbst. Schlüssel und Werte des Hashes sind für alle Seiten gleich. Der Aufruf der Funktion mit <catid> = 0 ist nicht möglich!

Beispiel bei folgender Kategoriestruktur:

0			Startseite
	10		Kategorie 1
		20	Kategorie 11
		30	Kategorie 111
		40	Kategorie 112
	50		Kategorie 2

Folgender Code:

```
my %seiten;
## 10: Seite, deren Unterseiten ausgelesen werden sollen
## \%seiten: Referenz auf das Hash-Array %seiten
ssp::getChildCategoriesRecursive(10, \%seiten);
foreach my $key (keys(%seiten)) {
	main::mprint "Seite $key: $seiten{$key}<br>";
```

Copyright 2017 by IBO Internet Software

```
}
```

würde diese Ausgabe erzeugen:

```
Seite 10: 10
Seite 20: 20
Seite 30: 30
Seite 40: 40
```

num ssp::getItemsPerPage()
Gibt die Anzahl der auszugebenden Artikel pro Seite für den aktiven Benutzer zurück.

ssp::setItemsPerPage(num <wert>)
Setzt die Anzahl der auszugebenden Artikel pro Seite für den aktiven Benutzer auf den Wert <wert>.

@num ssp::getItemsPerPageOptions()
Gibt ein nach Anzahl sortiertes Array der gültigen Auswahlmöglichkeiten für den Benutzer zurück.
Auf diese Weise kann die Auswahl im Shop dynamisch gebildet werden, ohne dabei versehentlich ungültige Auswahlmöglichkeiten mit auszugeben.

text ssp::get_suchkategorie(@text <felder>)
Erzeugt den Wert für den Parameter "suchkategorie" auf Basis der angegebenen <felder>.
In <felder> können auch benutzerdefinierte Datenfelder für Artikel und Varianten angegeben werden.
Bei Feldern, die auch für Varianten angelegt sind, werden automatisch auch die Varianten durchsucht.
Beispiel:

```
      my $suchkategorie = ssp::get_suchkategorie(qw/ITEMID DESCR ERG3 FELD1/);
```

ssp::do_not_print_headers()
Diese Funktion ermöglicht es die von ShopPilot normalweise ausgegeben HTTP-Header zu unterdrücken, um eigene Header z.B. für eine CSV-Datei auszugeben. Die Funktion muss dazu aufgerufen werden, bevor die Ausgabe des Templates beginnt. Dies kann man z.B. erreichen indem die Funktion im Modulrumpf eines SSP – Moduls aufgerufen wird (also nicht innerhalb einer Funktion).

Beispiel:

```
# Textdatei zum Download anbieten
<!—spmacro:modul(beispiel)
ssp::do_not_print_headers();

print qq{Content-Disposition: attachment; filename="test.txt"\n};
print "Content-type: text\plain\n\n";

print Inhalt ....

-->
```

Copyright 2017 by IBO Internet Software

<arg> = ssp::get_cmdln_arg(num <nr>)
Liefert die ersten 5 Argumente (0-4) der Kommandozeile zurück, die per Komma getrennt sind.
0: Kommando (evtl. mit Zusatzoptionen)
1: Sessionid (evtl. auch HaendlerID)
2: evtl. Artikelnr.
3: evtl. Offset
4: evtl. zusätzliche Argumente in a=b& Schreibweise

Beispiel:
iboshop.cgi?showd,424424242424243,abcnr,,adsource=google&mode=retarget

```
get_cmdln_arg(2)  -> abcnr

get_cmdln_arg(4)  -> adsource=google&mode=retarget
```

<args> = ssp::get_merch_conf_values()
Gibt die im Workcenter eigestellten Werte für Mindermenge Versandkosten und Rabatt in in einem Array (6 Werte) zurück. Die Reihenfolge ist

- Betrag bis zu dem Mindermengenkosten berechnet werden
- Die Höhe des Mindermengenzuschlags
- Betrag bis zu dem Versandkosten berechnet werden.
- Die Höhe der Versandkosten (es können noch andere Versandkosten hinzukommen)
- Betrag ab dem eine Rabatt gewährt wird
- Rabattwert in Prozent (es können noch andere Rabatte hinzukommen)

Beispiel:

```
my     ($MinderKBis,
        $MinderKBetrag,
        $VersandKBis,
        $VersandKBetrag,
        $RabattAb,
        $RabattSatz) = ssp::get_merch_conf_values();
```

Url = get_shop_base_url(hid)
Gibt die Basis-URL des Katalogs zurück, abhängig von der Händler-Id (hid). Kann z.B. benutzt werden, um die URL für das Canonical-Tag zu bilden.

3.5.2 Ausgabe des Identifiers

main::mprint(text <wert>)
Ersetzt den Identifier der Modul-Funktion durch den angegebenen Text. Wird mprint mehrfach innerhalb einer Modul-Funktion aufgerufen wird der Text aneinandergehängt.

Beispiel:

```
<!--spmacro:module(print)
sub hello{
       main::mprint("hello");
       main::mprint(" world!");
       return 0;
}
-->
__print::hello__
```

hello world!

3.5.3 Variablen im Warenkorb speichern

main::cart_put_formvars(text <label>)

Copyright 2017 by IBO Internet Software

speichert die per HTTP POST oder HTTP GET übergebenen zusätzlichen Variablen unter dem angegebenen Label im Warenkorb ab. Diese Daten können zu einem späterem Zeitpunkt mit cart_get_formvars ausgelesen werde. Per Konvention wird Länge des Labels mit drei Buchstaben festgelegt. Ein vierter Buchstabe wir automatisch hinzugefügt und dient zur Unterscheidung von den eingebauten Kennzeichen. Es muss natürlich darauf geachtet werden, dass alle anwendungsspezifischen Labels eindeutig sind.

main::cart_get_formvars(text <label>)

Liest die unter einem Label abgespeicherten Daten in die Formularvariablen zurück. Der Effekt ist identisch, mit dem Erhalt dieser Variablen über das abgeschickte Formular.

Beispiel:
```
<!--spmacro:module(Partner)
# auf einen Wert setzen oder der Wert kommt über ein Formular
# ssp::set_var_form("PARTNER_ID",4711);
# holt die abgespeicherter partnerid, die in einer vorhergehenden Seite mit
# cart_put_formvars("par") abgespeichert.
#
sub partnerid {
       main::cart_get_formvars("par");
       my $id = ssp::get_var_form("PARTNER_ID");
       main::mprint $id; ## Partner-Id ausgeben
       return 0;
}
-->
```

ssp::cart_write_var(text <label>, text <value>)
Speichert einen beliebigen Wert unter dem angegebenen Label im Warenkorb ab.

<wert> ssp::cart_read_var(text <label>)
Liest einen Wert, der unter dem angegebenen Label im Warenkorb gespeichert wurde, wieder aus.

Beispiel:
```
<!--spmacro:module(Name)

sub write_to_wk {
       #Formularwert auslesen oder beliebigen anderen Wert verwenden und im
       Warenkorb #speichern.
       my $name = ssp::get_var_form(“Name”);
       #Speichern des Wertes im Warenkorb um diesen auf allen Seiten #verfügbar zu
       machen.
       ssp::cart_write_var(“name”, $name);
}

sub read_from_wk {
       #Im Warenkorb gespeicherten Wert wieder auslesen und ausgeben.
       my $name = ssp::cart_read_var(“name”);
       main::mprint $name;
}

-->
```

3.5.4 Eingebettete Vorlagen

ssp::embedded(text type,text path_to_minitemplate, [text arg1, …]);

Erzeugt auf Basis eines Mini-Templates und der jeweiligen Funktion (durch type festgelegt) eine Ausgabe die in eine andere Seite eingebettet werden kann.
Der Type gibt an welche Funktion ausgeführt werden soll:

Funktion	type	mini-template	Argument
Warenkorb anzeigen	showwk	Pfad zur Vorlage	nicht notwendig
Suchergebnis anzeigen	search	Pfad zur Vorlage	Suchstring
Suchergebnis	csearch	Pfad zur Vorlage	Suchstring

Copyright 2017 by IBO Internet Software

anzeigen			
Artikelseite anzeigen	show	Pfad zur Vorlage	Seitennummer oder "byname<Titel>"

Beispiel Cross Artikel:

```
<!--spmacro:module(embed)
sub cross {
my $str = ssp::get_var_article('ZUBEHOER');
       ssp::embedded("search","mini_cross_artikel.txt",$str . ":00000001");
       return 0;
}
-->
```

Beispiel Artikel der Newsseite:

```
<!--spmacro:module(news)
sub show {
   ssp::embedded("show","mini_news.txt","bynameOE_Artikelliste");
}
-->
```

3.5.4.1 Aufbau des Suchstrings für search und cserach

Artikelnummern kommagetrennt
Doppelpunkt
Suchfelder oder Suchkategorie (normalerweise nur in der Artikelnummer)
Doppelpunkt
Preisbereich "von"
Doppelpunkt
Preisbereich "bis"
Doppelpunkt
Sortierfeld
Doppelpunkt
Sortierrichtung

```
Beispiel "132321,a23424,FGH_V34,676-4488d:100000000:10:100:1:1"
```

Der String braucht nicht alle Elemente zu enthalten, sondern kann nach der Suchkategorie oder später enden. Die Codierung der Strings für Suchkategorie, Suchfeld und Suchrichtung ist dem Dialog für die Einstellung der Standardsuche erklärt.

3.5.5 Tabs generieren

ssp::xtabs(text <template_aktiv>, text <template_inaktiv>)

Erzeugt auf Basis zweier Mini-Templates ein HTML-Konstrukt, welches eine Hauptkategorie des Shops repräsentiert. Dies kann beispielsweise zur Darstellung eines Navigationsmenüs über Reiter dienen. Die Funktion stellt fest, ob der aktuelle Tab die aktive Seite darstellt oder nicht. Davon hängt ab, wie das aktuelle Element dargestellt wird. Der Parameter <template_aktiv> gibt den Pfad zur Templatedatei für die aktiven Elemente an und der Parameter <template_inaktiv> den Pfad zur Templatedatei für die inaktiven Elemente.
Vor der erstmaligen Benutzung der XTabs müssen diese mittels **ssp::init_xtabs()** initialisiert werden!

ssp::init_xtabs()
Initialisiert die Variablen für die Anwendung von ssp::xtabs().

3.5.6 Funktionen für den Zugriff auf Variablen

Der Zugriff auf Variablen geschieht über Zugriffsfunktionen. Die entsprechende Variable wird dabei über ihren Namen ausgewählt. Der Name entspricht dabei i.d.R. dem Identifier, jedoch ohne Unterstriche. Der Zugriff auf benutzerdefinierte Variablen ist vollständig mit dieser Methode möglich. Bei den eingebauten Variablen gibt es gewisse Einschränkungen und der Name kann von der Bezeichnung des Identifier abweichen.

Beispiel: Auslesen eines Wertes aus einem benutzerdefinierten Feld.

```
ssp::get_var_article('BENUTZERFELD1',0);
```

Copyright 2017 by IBO Internet Software

Weiterhin sind verschieden Zugriffsfunktionen für Artikelseiten und Warenkorb vorhanden.
Alle Zugriffsfunktionen befinden sich im Modul **ssp**.

3.5.6.1 Artikel

<wert> ssp::get_var_article (text <name>,int <index>)
Der Rückgabewert ist der Wert der Variablen. Ist die angegebene Variable nicht definiert wird "undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.

Folgende eingebauten Variablen sind möglich:

Bedeutung	**Name**
Seitennummer	**ipagenr***
Artikelnummer	**itemid***
Beschreibung	**idescription**
Artikelpreis	**preis**
Artikelbild	**iimage**
Detailbild	**xdimage**
Detailtext	**xddescription**
Umsatzsteuersatz	**ustsatz**
Ergänzung 1	**ergaenz1**
Ergänzung 2	**ergaenz2**
Ergänzung 3	**ergaenz3**
Ergänzung 4	**ergaenz4**
Staffelpreis 1	**stafp1**
Staffelpreis 2	**stafp2**
Staffelpreis 3	**stafp3**
Staffelpreis 4	**stafp4**
Staffelmenge 1	**istafm1***
Staffelmenge 2	**istafm2***
Staffelmenge 3	**istafm3***
Staffelmenge 4	**istafm4***
Zusatztext	**izusatz***
Name Liste1	**iliste1**
Name Liste2	**iliste2**
Erstellungszeitpunkt	**xcreated**
Änderungszeitpunkt	**xmodified**
Benutzerdefinierte Felder	**durch den Namen (Großbuchstaben)**

* die Bezeichnung kann sich noch ändern

Copyright 2017 by IBO Internet Software

Beispiel: Alter Preis

```
<!--spmacro:module(preis)
sub alterpreis {
      ...
      my $p = ssp::get_var_article ('preis',0)*1.08;
      main::mprint "Alter Preis: $p ";
      return 0;
}
```

-->Beispiel: Ampel für Lagerbestand
Voraussetzung: Es ist ein benutzerdefiniertes Feld LBS für Artikel angelegt, falls automatische Abbuchung genutzt werden soll, ist statt LBS das Feld LBSAUTO zu verwenden.

```
<!--spmacro:module(item)
# Lagerbestand für Artikel
# Anzeige mit: __item::lbs__

sub lbs {
      my $index = $_[0];
      my $anzahl =  $_[1];
      if ($index <= $anzahl) {
            my $x = ssp::get_var_article('LBS',$index);
            if (length($x)) {main::mprint(&geticon($x));}
      }
}

sub geticon {
      my $l = $_[0];
      if ($l <= 0) {# kein Lagerberstand
            return "<img src=\"bilder/rot.gif\" alt=\"Dieser Artikel ist z.Zt.
nicht mehr ab Lager lieferbar\">";
      }
      if ($l < 20) {# geringer Lagerberstand
            return "<img src=\"bilder/gelb.gif\" alt=\"Dieser Artikel ist nur in
geringer Stückzahl vorhanden\">";
      }
      if ($l >= 20) {# verfügbar
            return "<img src=\"bilder/gruen.gif\" alt=\"Dieser Artikel ist ab
Lager lieferbar\">";
      }
}
-->
```

Beispiel: Ausgabe des Erstellungsdatums eines Artikels
Voraussetzungen: keine

```
<!--spmacro:module(item)
    ## Ausgabe des Erstellungszeitpunktes eines Artikels
    ## Anzeige mit: __item::erstelltam__
    sub erstelltam {
        my $index = shift;
        my $anzahl = shift;
        if ($index <= $anzahl) {
            # Erstellungszeitpunkt auslesen. Rückgabe ist ein Timestamp!
            my $created = ssp::get_var_article('xcreated', $index);
            my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
localtime($created);
            # Ausgabe des Zeitpunktes in lesbarer Form (dd.mm.yyyy / hh:mm:ss)
            my $created_str = sprintf("am %02d.%02d.%04d um %02d:%02d:%02d",
                $mday,          ## Tag im Monat
                $mon+1,         ## Monat Januar wäre die '0'
                $year+1900,     ## $year ist nur die Anzahl Jahre seit 1900
                $hour, $min, $sec);
            main::mprint("Dieser Artikel wurde erstellt<br><b>$created_str</b>.
<br>Der interne Timestamp lautet <br><b>$created</b>.");
        }
    }
-->
```

Copyright 2017 by IBO Internet Software

3.5.6.2 Artikelvarianten

<wert> ssp::get_var_varticle(text <name>,int <index>)
Artikelvarianten auslesen

Folgende eingebauten Variablen sind möglich:

Bedeutung	**Name**
Artikelnummer	**itemid* ****
Artikelpreis	**preis**
Beschreibung	**idescription**
Erstellungszeitpunkt	**xcreated**
Änderungszeitpunkt	**xmodified**

* die Bezeichnung kann sich noch ändern
** Bezeichnung "id" veraltet.
Die Werte beziehen sich auf den aktuellen Artikel (aktueller Index).
Neben den vordefinierten Werten sind auch alle benutzerdefinierten Felder für Varianten möglich.

3.5.6.3 Seiten

<wert> ssp::get_var_page (text <name>,int <pagenr>)

Seiteninformationen auslesen

Folgende eingebauten Variablen sind möglich:

Bdeutung	Name
Seitennummer	**ppagenr***
Vaterseite	**parent_pagenr***
Beschreibung	**pdescription***
Vorlage	**template***
Bild	**pimage***
Titel	**ptitle***
Icon	**picon***
Benutzerdefinierte Felder	**durch den Namen (Großbuchstaben)**

* die Bezeichnung kann sich noch ändern

Beispiel für Suchergebnisse:
Es soll der Link zur Kategorie des Artikels in der Artikelauflistung des Suchergebnisses angegeben werden.

```
<!--spmacro:module(kat)
sub p {
       my $index = $_[0];
       my $pagenr = ssp::get_var_article("ipagenr",$index);
       my $title = ssp::get_var_page("ptitle",$pagenr);
        main::mprint ("<a href = \"$ssp::xxpath?show$pagenr,$ssp::xxsession\">
$title  </a>");
return 0;
}
-->
```

`__kat::p__` Zeigt den Link zur Kategorie bei Artikeln im Suchergebnis an .

Ähnlich der Möglichkeiten zur Shop-Navigation über den Identifier __catalog__ können auch völlig eigene Navigationen oder Teil-Navigation erstellt werden.

<array> ssp::get_subpages (int <Seitennummer>)
Als Parameter wird eine beliebige Seitennummer aus dem Shop übergeben, die Funktion gibt ein Array mit allen Seitennummern zurück, die im Shop als Unterseite dieser Seite eingetragen sind.
Gibt es keine Unterseiten zu der angefragten Seite besitzt das erste Element das Arrays den Wert "undefined".

Copyright 2017 by IBO Internet Software

Beispiel für die Erstellung einer eigenen Navigation:
Die Auswahl soll mit Hilfe einer Listbox durchgeführt werden und alle Seiten der 1. Ebene darstellen. Die Auswahl soll mit als auch ohne aktiviertem Javascript funktionieren.

```
<!--spmacro:module(nav)
sub unterebene {
  my @subpages = ssp::get_subpages("0");

  foreach $sub (@subpages) {
    if ($sub == 'undefined') {return;} else {last;}
  }

  main::mprint ("<form Method=\"Post\" name=\"seitewechseln\"
Action=\"__xxpath__?show,__xxsession__\">\n");
  main::mprint ("<SELECT NAME=\"directPage\"
onChange='document.seitewechseln.submit()'>\n");

  foreach $sub (@subpages) {
    my  $titel = ssp::get_var_page("ptitle",$sub);
    main::mprint ("<OPTION VALUE=\"$sub\">Seite $titel  \n");
  }
  main::mprint ("</SELECT>\n");
  $noscript = qq {
     <noscript>
         <input type="submit" value="go" name="absenden">
     </noscript>
  };
  main::mprint ("$noscript \n");
  main::mprint ("</form>\n");
}
-->
```

`__nav::unterebene__` erzeugt die vollständige Listbox .

3.5.6.4 Listen

<wert> ssp::get_var_list(text <name>)

Diese Funktion gibt einen String zurück der für eine Liste hinterlegt ist. Aus diesem String kann man die einzelnen Listenelemente und Werte extrahieren. Jedes Element ist durch ein Komma getrennt. Zusätzlich zu dem Listenwert kann noch eine Artikelnummerergänzung vorhanden sein, diese ist per Doppelpunkt abgetrennt.

Beispiele
Blau:2a,Gelb:2b,Grau:2c
S,M,L,XL,XXL

```
my $liste = ssp::get_var_article('iliste1'); # Name der Liste
my $listvalues = ssp::get_var_list($liste);  # Inhalt holen
my @listelem = split(",", $listvalues);      # In Elemente teilen
```

3.5.6.5 Warenkorb

Die gleichen Funktionen sind auch für das Bestellformular, die Bestelldatenvorlage und die HTML-Mail möglich.

<wert> ssp::get_var_cart (text <name>,int <index>)

Folgende eingebauten Variablen sind möglich:

Bedeutung	**Name**
Artikelnummer	**id**
Artikelnummer des Hauptartikles (falls id Vararіante)	**hitid**
Seitennummer für Detailseite	**pgid**
Artikelpreis	**preis**

Copyright 2017 by IBO Internet Software

Anzahl	**anzahl**
Beschreibung	**produkt**
Beschreibung Hauptartikel (bei Varianten)	**hprodukt**
Pfad zum Bild (aus Artikelseite)	**image**
Zusatztext	**zusatztext**
Wert Listbox1	**zusatz1**
Wert Listbox2	**zusatz2**
Ergänzung 1	**ergaenz1**
Ergänzung 1	**ergaenz1**
Ergänzung 2	**ergaenz2**
Ergänzung 3	**ergaenz3**
Ergänzung 4	**ergaenz4**
Staffelpreis 1	**stafp1**
Staffelpreis 2	**stafp2**
Staffelpreis 3	**stafp3**
Staffelpreis 4	**stafp4**
Staffelmenge 1	**stafm1**
Staffelmenge 2	**stafm2**
Staffelmenge 3	**stafm3**
Staffelmenge 4	**stafm4**
Staffelmenge 4	**stafm4**
Benutzerdefinierte Felder des Artikles, der Subkatalogartikels oder der Variante	**durch den Namen (Großbuchstaben)**
Benutzerdefinierte Felder als Positionsvariablen im Warenkorb	**durch den Namen (Großbuchstaben)**

Der Rückgabewert ist der Wert der Variablen. Ist die angegebene Variable nicht definiert wird "undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.

num <wert> ssp::get_cart_items ()
Gibt die Anzahl **Positionen – 1** im Warenkorb zurück (höchster Index der Arrays).

num <wert> ssp::get_cart_wkprodsum ()
Gibt die Warenwertsumme aus dem Warenkorb zurück. In Zukunft besser die universeller Funktion get_wkprodsum benutzen (siehe unten).

num <wert> ssp::get_wkprodsum ()
Gibt die Warenwertsumme aus dem Warenkorb zurück, dies jedoch nicht nur in der Warenkorbseite, sondern auf allen Verzeichnis-, Artikel-, Detail- und Warenkorbseiten.

num <wert> ssp::get_wktotalsum ()
Gibt den Endbetrag inkl. Rabatte, Versandkosten, etc. an. Eine Anwendung ist z.B. für die Bestimmung des Abrechnungswertes bei Partnerprogrammen, wenn dem Kunden hohe Rabatte gewährt werden.

num <wert> ssp::get_cart_wktotalsum_netto()
Nettobetrag Endsumme des Endbetrags inkl. Versand etc. des Warenkorbs (Funktion kann nur im Warenkorb verwendet werden)

@(num <wert>, text<wert>) = ssp::get_cart_zahlart()
Gibt Informationen zur aktuell eingestellten Zahlungsart zurück.
Die Rückgabe erfolgt über ein Array, welches aus zwei Elementen besteht. Der Zahlungsart-ID und der Bezeichnung der Zahlungsart. Ist keine Zahlungsart eingestellt wird
$ssp::undefined für beide Werte zurückgegeben.

Beispiel:
my ($id,$name) = ssp::get_cart_zahlart()

Copyright 2017 by IBO Internet Software

main::mprint("Zahlungsart-ID $id Bezeichnung:$name)

@((num <wert>, text<wert>) = ssp::get_cart_zone()
Gibt Informationen zur aktuell eingestellten Lieferzone zurück.
Die Rückgabe erfolgt über ein Array, welches aus zwei Elementen besteht. Der Lieferzonen-ID und der Bezeichnung der Lieferzone. Ist keine Lieferzone eingestellt wird
$ssp::undefined für beide Werte zurückgegeben.

Beispiel:
my ($id,$name) = ssp::get_cart_zone();
main::mprint("Lieferzonen-ID $id Bezeichnung:$name");

Die Werte, die aus dem Warenkorb ermittelt, werden enthalten bei Nettoberechnung (B2B) keine Mehrwertsteuer. Bei Bruttoberechnung enthalten sie die Mehrwertsteuer des ersten Steuersatzes.

@(text<wert>) = ssp::get_all_zones($full);

Mit **$full = 0 oder fehlend** gibt die Funktion ein sortiertes Array mit den Namen aller für diesen Händler definierten Lieferzonen zurück. Dieses Array dient zum Darstellen einer Selectbox für die Adresseingabe bei der alternativen Bestellabwicklung.
Mit dem Parameter **$full = 1** gibt die Funktion ein Array von Arrays zurück. Das Datenarry enthält alle Informationen, die über diese Lieferzone hinterlegt sind.

Index	Wert
0	Lieferzonen-Id
1	Anzeigetext
2	Klassen-Zuschlagsliste
3	Zahlungskosten
4	Versandkostenfreie Artikel erlaubt
5	Steuerdaten: 1=Inland, 2=steuerfrei, 3=Ustid notwendig
6	Alternativer Text

Der Zugriff auf ein bestimmtes Element (hier Steuerdaten) erfolgt so.

```
my @zones = ssp::get_all_zones(1);
```

Achtung: Wobei der Index der Zeilen in dem Array nur mit der ID minus 1 übereinstimmt, wenn die niedrigste ID eine 1 ist und keine Lücken vorhanden sind.

```
my ($id,$text) =  ssp::get_cart_zone();
my $steuerdaten = $zones[$id -1]->[5];
```

Hashreferenz <wert> get_zones_hash()

Diese Funktion eignet sich besser als `get_all_zones` , um auf bestimmte Werte einer Zone zuzugreifen.
Jedes Hash beinhaltet hier ein Array, welches genauso aufgebaut ist, wie bei `get_all_zones`.

```
my ($id,$text) =  ssp::get_cart_zone();
my $zones = ssp::get_zones_hash();
my $steuerdaten = $zones->{$id}->[5];
```

text <wert> js_taxproplist()
Diese Funktion liefert ein Javascript Array zurück, mit dem der Client feststellen kann, wie die steuerliche Behandlung einer ausgewählten Lieferzone sein sollte.

Der zurückgegebene Code sieht ungefähr so aus (ohne Kommentare):

```
<script type="text/javascript">|;
var taxprop = new Array();
taxprop['Deutschland'] = 1; //Inland
taxprop['Schweiz'] = 2;     //steuerfrei
taxprop['Belgien'] = 3;     //Ustid
...
</script>
```

```
Ein JS Funktion welche das Eingabefeld für die UStid ein/ausblendet
```

Copyright 2017 by IBO Internet Software

sieht z.B. so aus.

```
<script type="text/javascript">
window.onload = do_ustid();
function do_ustid() {
	var tr_ustid = document.getElementById("id_tr_USTIDENTNR");
	var ustid = document.getElementById("id_USTIDENTNR");
	var lkz = document.getElementById("id_LKZ");
	if (taxprop[''+ lkz.value] != 3) { //verstecken
		tr_ustid.style.display = 'none';
		ustid.value=" "; //muss Wert haben
	} else {
		tr_ustid.style.display = 'table-row';
		ustid.value=""; //leerer Wert
	}
}
</script>
```

num <wert> ssp::get_cart_rabatt()
Gibt den absoluten Betrag des eingestellten Rabatts auf die Einkaufssumme in der Hauptwährung zurück. Wert ist in der Regel negativ.

num <wert> ssp::get_cart_myrabatt()
Gibt den absoluten Betrag des eingestellten Kundenrabatts in der Hauptwährung zurück. Wert ist in der Regel negativ.

num <wert> ssp::get_cart_minder()
Gibt den absoluten Betrag des Mindermengenzuschlags in der Hauptwährung zurück. Wert ist in der Regel positiv.

num <wert> ssp::get_cart_versand()
Gibt den absoluten Betrag der Versandkosten in der Hauptwährung zurück. Wert ist in der Regel positiv.

num <wert> ssp::get_cart_zahlartbetrag ()
Gibt den absoluten Betrag der Zahlungskosten in der Hauptwährung zurück..

num <wert> get_cart_rabatt_satz()
Gibt den normalen Summenrabattsatz zurück

num <wert> get_cart_myrabatt_satz()
Gibt den individuellen Kundenrabattsatz zurück

set_var_cart(text <name>,int <index>, text <wert>)
Mit dieser Funktion können benutzerdefinierte Variablen der Bestellposition verändert werden,
die mit <name> angegeben werden.
Die Funktion ist z.B. sinnvoll, wenn benutzerdefinierte Werte im Warenkorb noch verändert werden sollen.

3.5.6.6 Kundendaten

<wert> ssp::get_var_user (text <name>)

Diese Funktion ist auf allen Seiten möglich. Sie liefert jedoch nur definierte Werte, wenn ein Kunde auch eingeloggt ist. Neben allen benutzerdefinierten Feldern sind folgende Namen möglich:

Bedeutung	Name	Bedeutung	Name
Anrede	**anrede**	Vorname	**vorname**
Nachname	**name**	Firma	**firma**
Ort	**ort**	Postleitzahl	**PLZ**
Strasse	**strasse**	eMail-Adresse	**email**
Telefon	**tel**	Fax	**fax**
Landeskennzeichen	**LKZ**	Loginname	**login**

Copyright 2017 by IBO Internet Software

Passwort	**passwd**	Rabatt	**rabatt**
Zugeordneter Händler	**haendlerid**	Zug. Subkatalog	**subcatid**
Kundennummer	**kundennr**	Preisfaktor	**pfaktor**

Der Rückgabewert ist der Wert der Variablen. Ist die angegebene Variable nicht definiert wird "undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.

3.5.6.7 Haendlerdaten

<wert> ssp::get_var_merch (text <name>)

Diese Funktion ist auf allen Seiten möglich.
Folgende eingebauten Variablen sind möglich:

Bedeutung	**Name**
Händler ID	**hid**
Händler-Name	**hname**
Pfad zur Bilddatei	**hlogo**
Zusatztext, Adresse etc.	**hzusatz**
Homepage	**hurl**
Pfad zur Impressumdatei	**himpressum**
eMail des Händlers	**hemail**
Netto Preise	**ust_netto_preise**
Subkatalog ID	**subcatid**
Benutzerdefinierte Felder des Händlers	**durch den Namen (Großbuchstaben)**

Der Rückgabewert ist der Wert der Variablen. Ist die angegebene Variable nicht definiert wird "$ssp::undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.

3.5.6.8 Subkatalogdaten

Diese Funktion ist auf allen Seiten möglich.

<wert> ssp::get_var_subcat (text <name>)

Folgende eingebauten Variablen sind möglich:

sid oder mysubkat (Id des Subkatalogs)

Der Rückgabewert ist der Wert der Variablen. Ist die angegebene Variable nicht definiert wird "undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.

language (Index der eingestellten Sprache)

Es wird der Index der Sprache für den Subkatalog ausgegeben. Für den Hauptkatalog oder für Subkataloge ohne Spracheinstellung wird 0 zurückgegeben. Der Index ist die Nummer der Sprache in der project.ini – 1.

languagename (Name der eingestellten Sprache)

Es wird der Name der Sprache für den Subkatalog ausgegeben. Für den Hauptkatalog oder für Subkataloge ohne Spracheinstellung wird die erste Sprache in der project.ini zurückgegeben.

Beispiel: Zugriff auf Händler und Subkatalogdaten

```
<!--spmacro:module(ms)
sub katalogstatus {
  my $x = "HaendlerID " . ssp::get_var_merch("hid");
  $x .= " Haendlername " . ssp::get_var_merch("hname");
  $x .= " Haendlerzusatz " . ssp::get_var_merch("hzusatz");
```

Copyright 2017 by IBO Internet Software

```
  $x .= " SubkatalogID " . ssp::get_var_subcat("sid");
  $x .= " Sprachindex " . ssp::get_var_subcat("language");
  $x .= " Sprachname " . ssp::get_var_subcat("languagename");

  main::mprint($x);
  return 0;
}
```

Die Ausgabe erfolgt dann in der Vorlage mit

```
__ms::katalogstatus__
```

3.5.6.9 Abgeschickte Formulare

Auf der Ergebnisseite eines abgeschickten Formulars können die Werte der Formularvariablen ausgelesen werden. Es werden sowohl Standard-Formulare als
auch Multipart-Formulare unterstützt. Multipart-Formulare dienen zum Upload von Dateien.

Beispiel:

```
<form method="POST" enctype="multipart/form-data" name="send"
action="__xxpath__?showbynameUploadFormular,__xxsession__">
       <input name="datei" type="file">
       <input name="nocheinwert" value="bitte ausfüllen">
</form>
```

Achtung: Multipart-Formulare funktionieren derzeit nicht mit LiveView

<wert> ssp::get_var_form(text <name>)
Der Rückgabewert ist der Wert der Variablen mit dem Namen <name> im Formular. Ist die angegebene Variable nicht definiert wird "undefined" zurückgegeben. Wird die Variable in arithmetischen Operationen verwendet, wird dieser Wert als 0 interpretiert.
Normalerweise wird genau ein Wert pro <name> zugelassen. Das bedeutet, dass ein zweiter Wert für ein <name> den ersten Wert überschreibt. Dies kann umgangen werden, indem an den Feldnamen ein _LIST angehängt wird. Der Rückgabewert liefert dann eine Liste, deren einzelne Werte durch !_! voneinander getrennt sind (wert1!_!wert2!_!wert3!_!...).

Beispiel:
Mehrfachauswahl von eMails in einer Select-Box

```
<select name="email_LIST" multiple>
 <option value="info1@shoppilot.de">info1@shoppilot.de</option>
 <option value="info2@shoppilot.de">info2@shoppilot.de</option>
 <option value="info3@shoppilot.de">info3@shoppilot.de</option>
 <option value="info4@shoppilot.de">info4@shoppilot.de</option>
</select>
```

Werden hier nun mehrere E-Mails selektiert und das Formular abgeschickt, können sämtliche selektierte E-Mails per SSP folgendermaßen abgefragt werden:

```
...
       my $emailstring = ssp::get_var_form('email_LIST'); # Liste holen
       my (@emails) = split("!_!", $emailstring);         # Werte in Feld schreiben
main::mprint("Folgende eMails wurden ausgewählt: <br>");
       foreach my $email (@emails) {
              main::mprint("$email<br>");
}
...
```

<wert> ssp::get_var_form_filedata(text <name>)
Der Rückgabewert ist der Inhalt einer Datei, die mittels <input type="file"> hochgeladen wurde. In diesem Kontext liefert ssp::get_var_form den Dateinamen zurück.
Für beide Funktionen wird der Name des Input-Tags als <name> benutzt.

Zulässige Dateitypen:
       gif, jpg, jpeg, png, zip, rar, pdf, doc, txt, xls, csv
Sollte ein anderer Dateityp hochgeladen werden, so gibt ssp::get_var_form den String "Dateityp nicht erlaubt!" zurück und ssp::get_var_form_filedata einen Leerstring.

Copyright 2017 by IBO Internet Software

Beispiel:
Siehe Anwendungsbeispiele | Datei-Upload

<wert> ssp::set_var_form(text <name>,text<wert>)
Formularvariable <name> mit <wert> vorbelegen.

3.5.7 Anwendung von SSP-Identifiern

3.5.7.1 Identifier in SSP-Funktionen

Innerhalb von Artikelseiten und im Warenkorb ist die Benutzung von Standard-Identifiern innerhalb von SSP-Funktionen eingeschränkt möglich. Insbesondere darf jeder Identifier nur einmal innerhalb des SSP-Codes aufgeführt werden.
Die Anwendung ist deshalb möglich, weil der SSP-Code vor der Verarbeitung der übrigen Identifier ausgeführt wird. Er setzt also das Ergebnis über den SSP-Identifier in die Vorlage ein. Anschließend wird die Vorlage normal weiterverarbeitet.

Beispiel:
Abhängig vom Typ des Artikels (`ARTTYPE`) soll das Artikelbild mit Link ausgegeben werden.

```
sub image {
   my $index = $_[0];
   my $anzahl = $_[1];
   my $res = "";
   my $type = get_var_article('ARTTYPE',$index);
   if ($type == 0) {
       $res = "__image__";
   }
   main::mprint($res);
   return 0;
}
```

3.5.7.2 Mehrfachbenutzung

Soll ein SSP-Identifier für eine Artikelzeile mehrfach benutzt werden, so muss die Funktion einen return Wert von "noinc" zurückgeben.

Bei alten Funktionen wird oft kein return benutzt. In diesem Fall wird implizit der Wert in $_ (das Ergebnis der letzten Funktion) zurückgegeben, ist diese ungleich "$ssp::noinc", ist Kompatibilität gegeben. Dies spielt auch nur dann eine Rolle falls der SSP-Identifier, der erste Identifier für einen Artikel ist.

Beispiel zur Benutzung der Rückgabewerte:

```
<!--spmacro:module(test)
sub artnr_ohne_wechsel {
       my $p = ssp::get_var_article('itemid', $index);
       main::mprint "$p";
       return "$ssp::noinc";
}
-->

<!--spmacro:module(test)
sub artnr_mit_wechsel {
       my $p = ssp::get_var_article('itemid', $index);
       main::mprint "$p";
       return 0;
}
-->
```

Wenn die erste Funktion in einer HTML Vorlage nun folgendermaßen aufgerufen wird,
würde dreimal die Artikelnummer des selben Artikels ausgegeben.

```
...
__test::artnr_ohne_wechsel__<br>
__test::artnr_ohne_wechsel__<br>
__test::artnr_ohne_wechsel__<br>
...
```

Copyright 2017 by IBO Internet Software

ergibt

4711
4711
4711

Im Gegensatz dazu würde folgendes die Artikelnummern von 3 aufeinander folgenden Artikeln ausgeben.
...
__test::artnr_mit_wechsel__
__test::artnr_mit_wechsel__
__test::artnr_mit_wechsel__
...

ergibt

```
4711
4232A
6767C
```

Copyright 2017 by IBO Internet Software

Callbacks

Ist innerhalb einer Vorlage ein Modul mit dem Namen "Callback" definiert, können innerhalb des Moduls sog. Callback-Funktionen verwandt werden. Diese Funktionen werden, wenn sie innerhalb des Moduls definiert sind, an bestimmten Stellen des Shop-Kernels ausgeführt.

3.5.7.3 invoice_addpos

invoice_addpos(ref num <gesamtsumme>,
ref num <gesamtsumme_netto>,
ref num <gesamtsumme_steuer>,
num <Umsatzsteuer>)

Diese Funktion dient dazu innerhalb der Summenberechnung der Rechnung eine zusätzliche Rechnungsposition zu berücksichtigen. Dazu müssen die als Referenz übergebenen Variablen der Summenberechnung modifiziert werden. Dies sind die Gesamtsumme, die Gesamtsumme des Nettowertes und die Gesamtsumme der Steuer. Als zusätzliche Informationen wird der Umsatzsteuersatz 1 und ein Flag übergeben, welches anzeigt, ob eine Nettorechnung vorliegt. Die Gesamtsumme des Nettowerts hat derzeit keine Bedeutung und kann ignoriert werden.

3.5.7.4 invoice_addpos_final

invoice_addpos_final(ref num <gesamtsumme>,
ref num <gesamtsumme_netto>,
ref num <gsamtsumme_steuer>,
num <Umsatzsteuer>,
flag <nettoberechnung>,
ref num < versandbetrag>,
ref num < minderbetrag>,
ref num < zahlkosten>
)

Die Parameter entsprechen der Funktion invoice_addpos() und zusätzlich Versandkosten, Mindermengenzuschlag und Zahlungskosten. Im Unterschied zu invoice_addpos() wird jedoch die Veränderung der Rechnungssummen **nach** der Berechnung aller übrigen Kosten (Versand, Zahlung) und Gutschriften (z.B. Rabatte) berechnet.

Beispiel für die Anwendung bei Berechnung der MwSt. Die MwSt soll für bestimmte Zielländer (Lieferzonen) nicht berechnet werden. Dazu ist der folgende Code in die Datei stdcart.ipl einzutragen. Die Datei muss als zusätzliche Datei im WC eingetragen werden, damit Sie auf den Server übertragen wird.

Beispiel 1: keine MWST für Zone 10

```
<!--spmacro:module(Callback)

sub invoice_addpos_final {
  my $ref_gesamt = $_[0];
  my $ref_gesamt_steuer = $_[2];
  my $ust = $_[3]/100;
  my $netto = $_[4];

  my ($wert,$text) =  ssp::get_cart_zone();

  # Wenn 10 die Zahlungszonen-ID für ein Land ohne Mwst. Berechnung ist
  if ($wert == 10) {
        $$ref_gesamt -= $$ref_gesamt_steuer;
        $$ref_gesamt_steuer = 0;
  }
  return 0;
}
-->
```

Beispiel 2: keine Versandkosten ab 250 EUR für Zone 7

```
<!--spmacro:module(Callback)
```

Copyright 2017 by IBO Internet Software

```
sub invoice_addpos_final {
  my $ref_gesamt = $_[0];
  my $ref_gesamt_steuer = $_[2];
  my $ust = $_[3]/100;
  my $netto = $_[4];
  my $versandkosten = $_[5];

  my ($wert,$text) =  ssp::get_cart_zone();

  # Wenn 7 die Zahlungszonen-ID für das Land
  if ($wert == 7) {
       if ($$ref_gesamt - $$versandkosten > 250) {
         $$ref_gesamt -= $$versandkosten;
         $$ref_gesamt_steuer -= (1 - 1/(1+$ust))* $$versandkosten;
         $$versandkosten = 0;
       }
  }
  return 0;
}
-->
```

3.5.7.5 @text mailtext = invoice_addpos_mail()

Fügt in der Text-Email eine Rechnungszeile an, in der Rabatte oder Zuschläge ausgewiesen werden können

Es muss ein Feld mit zwei Elementen zurückgegeben werden.

```
$mailtext[0] =  Betrag (num)
$mailtext[1] =  Beschreibung (text)
```

Diese Werte werden als letzte Position in die Text-Email eingefügt.

3.5.7.6 num <anzahl> begin_read_template()

Callback für Ausführung von Aktionen vor dem vollständigen Einlesen der Vorlage (siehe Datenbankzugriff).

3.5.7.7 after_read_template(num <key>, text <file>, num <count>)

Diese Funktion liefert einen definierten Callback für weitere Bearbeitungen nach dem Einlesen der Template Datei. Sie ist somit ein frühes Callback, da das Template ja auch den SSP Code beinhaltet. Hier kann z.B. die Funktion cart_put_formvars ausgeführt werden, um übergebene Variablen permanent zu speichern.

Beispiel:

```
<!--spmacro:module(Callback)
sub after_read_template {
       main::cart_put_formvars("par");
}
-->
```

3.5.7.8 @templates_new modify_search_templates(<text> templ_notfound, <text> templ_found, <text> templ_found_asn)

Erlaubt die Überschreibung der eingetragenen Suchseitenvorlagen bevor nach einer Suche die entsprechende Suchergebnisseite ausgegeben wird. Somit ist es beispielsweise möglich, je nach Suchparameter (Suchbegriff, Suchort, Suchkategorie) unterschiedliche Vorlagen für die Suchergebnisse zu verwenden.

Übergeben werden die im Shop eingetragenen Vorlagen für "Vorlage für 'nichts gefunden'", "Vorlage für Suchergebnisse" und "ASN Vorlage".

Zum Überschreiben aller (oder einzelner Vorlagen) kann die Funktion ein Feld mit den einzutragenden Vorlagen zurückgeben. Das Feld muss 3 Elemente enthalten und folgendermassen aufgebaut sein:

```
(
       <text> neue_vorlage_nichts_gefunden,
       <text> neue_vorlage_suchergebnisse,
```

Copyright 2017 by IBO Internet Software

<text> neue_vorlage_suchergebnisse_asn
)

Um nur einzelne Vorlagen zu überschreiben, können für die übrigen Vorlagen Leertexte ('') übergeben werden.

Beachten Sie, dass die Funktion ausschließlich in der stdall.ipl verwendet werden kann!

Beispiel:

```
<!--spmacro:module(Callback)
sub modify_search_templates {
    # Alte Vorlagen werden übergeben
    my $templ_nothing_found = shift;
    my $templ_found = shift;
    my $templ_found_asn = shift;

    # Je nach "suchort" werden unterschiedliche Vorlagen für die
    # Suchergebisseite verwendet. Die Vorlagen für die ASN-Suche und
    # für die Seite "nichts gefunden" werden NICHT überschrieben.
    my $suchort = int(ssp::get_var_form('suchort'));
    if ($suchort == 10) { # Es wurde in der Kategorie "Drucker" gesucht
        return ('', 'html_de/p_suche_found_drucker.html', '');
    } elsif ($suchort == 40) { # Es wurde in der Kategorie "Kabel" gesucht
        return ('', 'html_de/p_suche_found_kabel.html', '');
    } elsif ($suchort == 60) { # Es wurde in der Kategorie "Laufwerke" gesucht
        return ('', 'html_de/p_suche_found_laufwerke.html', '');
    }

    # Für alle anderen Fälle wird nichts zurückgegeben,
    # wodurch dann die originalen Vorlagen verwendet werden.
}
-->
```

3.5.7.9 text <fuereinenartikel> set_simpleradio_text(text <artikelnr>, text <beschreibung>, text <preis>, num <index>)

Erlaubt die Ausgabe bei Variantenartikel, die mit dem Identifier __radiovar__ dargestellt werden, zu modifizieren. Dabei werden die Werte für Artikelnummer, Artikelbeschreibung und Preis an die Funktion übergeben. Als Rückgabewert wird ein Textstring erwartet, der als Beschreibung eines Radiobuttons dargestellt wird.

Beispiel:

```
<!--spmacro:module(Callback)
sub set_simpleradio_text {
   my $art    = $_[0];
   my $beschr = $_[1];
   my $preis  = $_[2];

   return "$art <b>$beschr</b> $preis";
}
-->
```

4711 **Rote Grütze 200g** EUR 2,39

4712 **Rote Grütze 500g** EUR 4,39

Copyright 2017 by IBO Internet Software

3.5.7.10 text <jscode> js_placecart_insertion()

Fügt in die Funktion placecart (inwk) Code ein, um zusätzliche Funktionen innerhalb des JavaScript Codes zu realisieren. Der Code wird ausgeführt, wenn ein Artikel in den Warenkorb gelegt wird (jedoch nicht bei Pagetype 1 oder 2).

Beispiel:
Der Funktions-Indentifier lieferbar::test wird in jede Artikelzeile eingefügt und definiert ein nicht sichtbares Formular, welches den Lieferstatus je Artikel in einer Variablen "A$artikel" ausweist.

```
<!--spmacro:module(Callback)
# 1. Die benutzerdef. Var. LSTATUS zeigt an, dass ein Artikel nicht lieferbar ist.
# Der Wert der Var. wird bei jedem Artikel durch die Funktion lieferbar::test
# als hidden input festgehalten.
# 2. Es wird ein alert Fenster zur Bestätigung des "In den Warenkorb legen"
# angezeigt
sub js_placecart_insertion {
my $js = qq[
   status = eval('document.LieferForm' + xarticle + '.A' + xarticle).value
       if (status==1) {alert('Artikel ist zur Zeit nicht lieferbar'); return;}
       alert('Der Artikel wird in den Warenkorb gelegt');
       ];
   return $js;
}
-->

<!--spmacro:module(lieferbar)
sub test {
       my $index = $_[0];
       my $anzahl =  $_[1];
       if ($index <= $anzahl) {
              my $artikel = ssp::get_var_article('artnr',$index);
              main::mprint("<form name=\"LieferForm$artikel\" ><input
name=\"A$artikel\" type=hidden val
ue=\"ssp::get_var_article('LSTATUS',$index)"></form>");
       }
       return 0;
}
-->

__lieferbar::test__
```

Copyright 2017 by IBO Internet Software

Die Funktion placecart_insertion() kann auch direkt in den Quelltext der Vorlage geschrieben werden.
Der Aufruf erfolgt, sobald etwas in den Warenkorb gelegt wird:
(Ab ShopPilot Version 2.80.03)

```
<script type="text/javascript">
	function placecart_insertion(pgid,id,article,start,pack,min){

		alert('Artikel wurde in den Warenkorb gelegt');
	}
</script>
```

Die Funktion erhält als Parameter alle relevanten Variablen übergeben.

3.5.7.11 num <price> modify_int_price(num <index>, num <anazhl>)

Diese Funktion ermöglicht es den Preis im Warenkorb zu verändern.
Übergeben wird der Index der Warenkorbposition und die Anzahl-1.
Der Preis ist mit ssp::get_var_cart zu lesen. Zückgegeben wird der veränderte Preis.

3.5.7.12 num <price> modify_db_price(num <price>, text <artnr>)

Diese Funktion ermöglicht es den Preis beim Einlesen der Daten aus der Datenbank zu verändern. Die Preisänderung ist also für alle Preisangaben im Shop wirksam.
Übergeben wird der Originalpreis und die Artikelnummer. Zurückgegeben wird der veränderte Preis.

3.5.7.13 modify_price_quantity_scale

```
(P,M1,M2,M3,M4,P1,P2,P3,P4)=
Callback::modify_price_quantity_scale(P,M1,M2,M3,M4,P1,P2,P3,P4,ItemID)
```

P Einzelpreis
M1-M4 Staffelmenge
P1-P4 Staffelpreis
ITEMID Artikelnummer

Abhängig von anderen Bedingungen (Z.B. angemeldeter Benutzer) können alle Preise und Staffelmengen verändert werden. Sie werden der Funktion übergeben und die Rückgabe liefert die neuen Werte.

Beispiel für die Verschiebung von Staffelpreisen, abhängig von der dem Kunden zugewiesenen Preisgruppe:

```
<!--spmacro:module(Callback)
sub modify_price_quantity_scale {
	my $p = $_[0];
	my $m1 = $_[1];
	my $m2 = $_[2];
	my $m3 = $_[3];
	my $m4 = $_[4];
	my $p1 = $_[5];
	my $p2 = $_[6];
	my $p3 = $_[7];
	my $p4 = $_[8];
	my $itid = $_[9];

	my $pg = ssp::get_var_user('PREISGRUPPE');
	if (($pg eq $ssp::undefined) || ($pg == 0))
	{		$pg = 1; # nicht angemeldete Benutzer
	}
#	ssp::trace("begin ($pg) $p,$m1,$m2,$m3,$m4,$p1,$p2,$p3,$p4\n");
	if ($itid eq 'A4711')
	{		$pg = 1; #  kein Rabatt für Artikel Nr. A4711
	}
	##  0..1 nothing to do
	if ($pg==2) {
	  $p  = $p1;
	  $p1 = $p2;
	  $m1 = $m2;
```

Copyright 2017 by IBO Internet Software

```
          $p2 = $p3;
          $m2 = $m3;
          $p3 = $p4;
          $m3 = $m4;
          $m4 = 0;
          $p4 = 0;
        }
        if ($pg==3) {
          $p  = $p2;
          $p1 = $p3;
          $m1 = $m3;
          $p2 = $p4;
          $m2 = $m4;
          $p3 =  0;
          $m3 =  0;
          $m4 =  0;
          $m4 =  0;
        }
         ….
        # else nothing to do
#       ssp::trace("begin ($pg) $p,$m1,$m2,$m3,$m4,$p1,$p2,$p3,$p4\n");
        return ($p,$m1,$m2,$m3,$m4,$p1,$p2,$p3,$p4);
}
```

3.5.7.14 Shopcontol Callback

ShopControl::printateob()

Dieser Callback wird für die Einbettung von zusätzlichem Code vor dem End of Body-Tag genutzt. Er dient dazu Anbindungen für Analyse und Webcontrol Systeme zu schaffen.
Um die Funktion zu nutzen, muss ein Modul ShopControl mit der Funktion printateob() existieren. Üblicherweise wird dieses Modul in stdall.ipl abgelegt, damit automatisch alle Seiten erfasst werden.

Die Funktion printateob gibt mit main::mprint ein Quellcodeschnipsel aus, das dann vor dem Body-Tag eingefügt wird. Man hat damit die Möglichkeit, die Ausgabe zu steuern, ohne die Vorlagen zu verändern.

z.B. für Google Analytics

```
<!--spmacro:module(ShopControl)
sub printateob {
       my $out;
       $out = qq{
<script src="https://ssl.google-analytics.com/urchin.js" type="text/javascript">
</script>
<script type="text/javascript">
_uacct = "UA-1234567-1";
urchinTracker();
</script>
};
main::mprint($out);
}
-->
```

Für das Shopcontrol-System Econda (www.econda.de) ist ein Modul verfügbar, welches genauere Analysen als Google ermöglicht.

3.5.7.15 get_catalog_markup

```
Callback::get_catalog_markup(level,selected,fist,last,link,title)
```

Die Funktion kann die Standardausgabe des Catalog-Identifiers ersetzen. Dazu muss die Funktion einen HTML-Schnipsel als Rückgabe bereitstellen.

level: Level der Einrückung von 1.. n (die Hauptkategorie ist 1)
selected: eine ausgewählte Kategorie hat 1 sonst 0

Copyright 2017 by IBO Internet Software

first: der erste Eintrag einer Ebene hat hier 1 stehen.
last: last ist immer 0. Nur nach dem letzten Eintrag einer Ebene, wird diese Funktion erneut aufgerufen. Dann ist last =1, alle anderen Parameter haben dann keine Bedeutung.
link:: Die URL zum Aufruf der Kategorie
title: Die Beschreibung der Kategorie

```
<!--spmacro:module(Callback)

sub get_catalog_markup {
	$level = $_[0];
	$selecetd = $_[1];
	$first = $_[2];
	$last = $_[3];
	$link = $_[4];
	$title = $_[5];
	return "<a href=\"$link\">$title</a>L:$level S:$selecetd F:$first
L:$last<br>\n";
}
-->
```

3.5.7.16 exclude_zahlart

Mit dieser Callback-Funktion ist es möglich Zahlarten auszuschließen.
Die Funktion gibt eine kommagetrennte Liste der ID's der auszuschließenden Zahlarten zurück oder auch nur eine einzige ID.

Beispiel es soll eine Zahlart abhängig vom Warenwert nicht angeboten werden. (Hier Zahlungsart Rechnung erst ab 30 EUR anbieten.)

```
<!--spmacro:module(Callback)

sub exclude_zahlart {

	my $sum = ssp::get_wkprodsum();
	if ($sum < 30.00) {
		return 1; # ID für Rechnung ist 1
	} else {
		return "";
	}
}
-->
```

3.5.7.17 AddSQLCondition

Hiermit können die SQL Statements, die für das Auslesen der Artikel verantwortlich sind verändert werden. Es kann eine zusätzliche Bedingung eingebaut werden. Damit können die sichtbaren Artikel im Shop eingeschränkt werden. Zu beachten ist, dass die Namen der Spalten qualifiziert sein müssen d.h. aus TABELLE.SPALTE bestehen sollen.

Beispiel in stdall.ipl:

```
<!--spmacro:module(Callback)
sub AddSQLCondition {
	return " AND (ITEM.HERSTELLER = 'Bosch')"
}
-->
```

Hier werden z.B. nur Artikel eines Herstellers angezeigt.

Das AddSQLCondition wird bei der Vorbereitung von verschiedenen SQL-Statements aufgerufen. Zur Unterscheidung dient ein übergebener Parameter.

Wert	Funktion	
1	Artikel suchen (Shopsuche)	
2	Varianten suchen (Shopsuche)	
3	Detailseite und Warenkorb	

Copyright 2017 by IBO Internet Software

4	Artikel einer Seite einlesen	ORDER BY erlaubt

Bei der Funktion 4 kann satt der Bedingung oder zusätzlich zu der Bedingung ein Sortierung mitgegeben werden.

3.5.7.18 modify_search_templates

Diese Callback-Funktion dient dazu im Bedarfsfall alternative Templates für die Suchergebnisse bereitzustellen. Die Standardtemplates werden als Parameter an die Funktion übergeben. Als Rückgabe wird eine Liste der alternativen Templates erwartet. Da diese Funktion vor dem Einlesen eines Templates ausgeführt wird, macht die Funktion nur in stdall.ipl Sinn. Weiterhin braucht man noch ein Unterscheidungsmerkmal, welches erkennbar macht, wann die alternativen Templates zum Einsatz kommen soll. Eine Möglichkeit ist z.B. eine hidden Formularfeld zu benutzen.

<input type="hidden" name="alternativesuche" value="ja"> wird in das alternative Suchformular eingebaut.

```
<!--spmacro:module(Callback)
sub modify_search_templates {
       my ($nichts_gefunden,$suchergebnisse,$asn_suche) = @_;
       if (!length(ssp::get_var_form('alternativesuche')) {
              return; ## keine Änderung der Templates
       }
       # nur die Suchergebnisseite wird verändert
       $suchergebnisse = "html_de/meinesuchergebnisse.html";
       return ($nichts_gefunden,$suchergebnisse,$asn_suche)
}
-->
```

3.5.7.19 searchdb_limit_results

```
num <maxresults> Callback::searchdb_limit_results (
        num             <maxresultsDefault>,
        arrayref        <searchterms>
)
```

Standardmässig werden bei einer Suche nur 500 Suchergebnisse zurückgegeben.
Über diese Callback-Funktion kann diese Begrenzung angepasst werden.
Dazu muss die Funktion den gewünschten Wert lediglich zurückgeben.

3.5.7.20 check_order

```
hash <errors> Callback::check_order(
        text    <form_template>,
        flag    <do_order_silent>,
        num     <atype>,
        text    <actiontemplate>,
        flag    <no_mails>
)
```

Das Callback check_order wird beim Absenden des Bestellformulares aufgerufen. Wenn der alternative Bestellablauf verwendet wird, wird das Callback auf allen Seiten ab dem Bestellformular aufgerufen.

Parameter:

- `form_template`: HTML-Vorlage Bestellformular
- `do_order_silent`: keine interaktive Bestellung, falls gesetzt
- `atype`: Aktionstyp des Formulares (1 = Nachricht, 0 = Bestellung)
- `actiontemplate`: Aktionsvorlage
- `no_mails`: Kein Mailversand, falls gesetzt

Mithilfe des Identifiers check_order ist es möglich eigene Fehlerprüfungen für die einzelnen Formulare im Bestellprozess durchzuführen. Im Fehlerfall können die entsprechenden Fehler in dieser Funktion an den Shop zurückgegeben werden. Im weiteren Verlauf werden diese Fehler wie intern aufgetretene Fehler behandelt. Das heisst, dass das Absenden des Formulares abgebrochen wird und die Meldungen im Browser ausgegeben werden.
Die Rückgabe muss hierfür pro Fehler den Namen des Feldes definieren, sowie den auszugebenden Fehlertext.

Im nachfolgenden Beispiel wird im Fehlerfall und bei aktivierten Inline-Fehlermeldung das Formular neu aufgerufen und __errorMYTESTFIELD__ durch die entsprechende Fehlermeldung ersetzt.

Copyright 2017 by IBO Internet Software

Beispiel HTML-Vorlage (myvorlage.html):

```
<form method="post" action="...">
      <!-- Hier die Felder des Formulares -->

      Bitte geben Sie Ihr Geburtsdatum im Format TT.MM.JJJJ ein:
      <input type="text" name="MYTESTFIELD" value="">
      <div class="error">__errorMYTESTFIELD__</div>

    <input type="submit" value="abschicken">
</form>
```

Beispiel stdall.txt:

```
<!--spmacro:module(Callback)
sub check_order {
    my $template = shift;

    # Nur im entsprechenden Formular prüfen
    return if ($template !~ /myvorlage\.html/);

    my $gebdatum = ssp::get_var_form('MYTESTFIELD');
    if (!$gebdatum) {
        return (MYTESTFIELD => 'Kein Geburtsdatum angegeben');
    }

    my ($tag, $monat, $jahr) = split(/\./, $gebdatum);
    if (int($tag) < 1 || int($tag) > 31) {
        return (MYTESTFIELD => 'Ungültiger Tag');
    }

    if (int($monat) < 1 || int($monat) > 12) {
        return (MYTESTFIELD => 'Ungültiger Monat');
    }

    if (int($jahr) < 1900 || int($jahr) > 2000) {
        return (MYTESTFIELD => 'Ungültiges Jahr');
    }

    return (); # Keine Fehler setzen
}
-->
```

3.5.7.21 check_udc_action

```
hash <errors> Callback::check_udc_action(
        num     <pgid>
)
```

Das Callback check_udc_action wird beim Absenden von Formularen automatisch aufgerufen.

Parameter:
- pgid: Seitennummer des abgeschickten Formulares

Mithilfe des Identifiers check_udc_action ist es möglich eigene Fehlerprüfungen für die einzelnen Formulare. Im Fehlerfall können die entsprechenden Fehler in dieser Funktion an den Shop zurückgegeben werden. Im weiteren Verlauf werden diese Fehler wie intern aufgetretene Fehler behandelt.
Das heisst, dass das Absenden des Formulares abgebrochen wird und die Meldungen im Browser ausgegeben werden.
Die Rückgabe muss hierfür pro Fehler den Namen des Feldes definieren, sowie den auszugebenden Fehlertext.

Im nachfogenden Beispiel wird im Fehlerfall und bei aktivierten Inline-Fehlermeldung das Formular neu aufgerufen und __errorMYTESTFIELD__ durch die entsprechende Fehlermeldung ersetzt.

Copyright 2017 by IBO Internet Software

Beispiel HTML-Vorlage (myvorlage.html):

```
<form method="post" action="__UDC__">
      <!-- Hier die Felder des Formulares -->

      Bitte geben Sie Ihr Geburtsdatum im Format TT.MM.JJJJ ein:
      <input type="text" name="MYTESTFIELD" value="">
      <div class="error">__errorMYTESTFIELD__</div>

    <input type="submit" value="abschicken">
</form>
```

Beispiel stdall.txt:

```
<!--spmacro:module(Callback)
sub check_udc_action {
    my $pgid = shift;

    # Nur im entsprechenden Formular prüfen
    return if ($pgid != 1300001970);

    my $gebdatum = ssp::get_var_form('MYTESTFIELD');
    if (!$gebdatum) {
        return (MYTESTFIELD => 'Kein Geburtsdatum angegeben');
    }

    my ($tag, $monat, $jahr) = split(/\./, $gebdatum);
    if (int($tag) < 1 || int($tag) > 31) {
        return (MYTESTFIELD => 'Ungültiger Tag');
    }

    if (int($monat) < 1 || int($monat) > 12) {
        return (MYTESTFIELD => 'Ungültiger Monat');
    }

    if (int($jahr) < 1900 || int($jahr) > 2000) {
        return (MYTESTFIELD => 'Ungültiges Jahr');
    }

    return (); # Keine Fehler setzen
}
-->
```

Copyright 2017 by IBO Internet Software

3.5.7.22 modify_mailtext

Diese Funktion wird bei jedem Senden einer Email aufgerufen. Es wird neben dem Empfänger und dem Betreff zusätzlich der Text der Mail als Referenz übergeben. Der Text kann also verändert werden. Dies ist z.B. dann sinnvoll, wenn man einen Anhang anfügen möchte. Ist die Länge des referezierten Mail-Textes nach dem Aufruf 0, wird keine Mail versendet.

```
Callback::modify_mailtext(<empfänger>,<betreff>, ref <mailtext>)
```

```
Beispiel:

<!—spmacro:module(Callback)
sub modify_mailtext {
       my $textref = $_[2];
       ## Disclaimer anhängen
       $$textref .= "\n\nDisclaimer Copyright by us.";
}
-->

Beispiel2: Mailversand verhindern.

<!—spmacro:module(Callback)
sub modify_mailtext {
       my $to = $_[0];
       my $textref = $_[2];
       if ($to =~ /evil@/) {
              $$textref = ""; ## nichs senden
       }
}
-->

Beispiel3: Eine PDF-Datei anhängen

<!--spmacro:module(Callback)
sub modify_mailtext {
       my $to = $_[0];
       my $subject = $_[1];
       my $ref_mailtext = $_[2];
       my $type = "application/pdf";
       use MIME::Base64;
       my $pdffile = "agb.pdf"; #liegt im Scriptordner (da wo auch iboshop.cgi liegt)
       open(PDF, "<$pdffile");
       my @lines = <PDF>; close PDF;
       my $attachment = join("",@lines);

       # Die Bestellmail wird anhand des Subject erkannt
       if ($attachment && $subject eq "Bestätigung Konfiguratorshop (EP)") {
              my $mailtext = $$ref_mailtext;
              my $bound = "----------" . $subject . $to;

              my $newmail = "MIME-Version: 1.0"."\n";
              $newmail .= "Content-type: multipart/mixed; boundary=\"$bound\""."\n\n";
              $newmail .= "--$bound"."\n";
              $newmail .= "Content-type: text/html"."\n";
              $newmail .= $mailtext."\n\n";
              $newmail .= "--$bound"."\n";
              $newmail .= "Content-type: $type;  name=\"agb.pdf\"\n";
              $newmail .= "Content-Transfer-Encoding: base64\n";
              $newmail .= "Content-Disposition: attachment; filename=\"agb.pdf\"\n\n";

              $newmail .= MIME::Base64::encode($attachment);

              $newmail .= "\n--$bound--\n";

              $$ref_mailtext = $newmail;
       }
}
-->
```

3.5.7.23 change_mydelwk

Beim Löschen des Warenkorbes werden alle Zeilen entfernt, die Produkte enthalten. Man kann diese Operation jetzt abändern. Indem man einen 1 zurück gibt, wird die betreffend Zeile nicht gelöscht. Diese Funktion wird

Copyright 2017 by IBO Internet Software

aufgerufen, wenn der Befehl "delwk" ausgeführt wird. Sie wird für jede Zeile des Warenkorbs aufgerufen. Anhand des Arrayelements kann man den Typ der Zeile erkennen. "ord0" ist z.B. ein Artikel im Warenkorb.
Das erste Element ist der Typ, das zweite die Artikelnummer (bei "ord0") das 3. die Anzahl (bei "ord0").

```
num <nichtlöschen> Callback::change_mydelwk(arrayref <warenkorbzeile>)

Beispiel:

<!—spmacro:module(Callback)
sub change_mydelwk {
      my $arrayref = $_[0];
      ## ist es eine Artikelzeile ?
      if ($arrayref->[0] eq "ord0") {
            ## ist es Artikel 4711 ?
            if ($arrayref->[1] eq "4711") {
                  return 1; ## nicht löschen
      }
      return 0; ## nichts ändern
}
-->
```

3.5.7.24 on_syserr

Das Callback on_syserr wird automatisch aufgerufen, wenn im Shop ein Fehler per main::syserr() ausgegeben wird. Hierbei erfolgt zunächst die normale Fehlerbehandlung durch main::syserr (zb Log-Eintrag schreiben). Erst danach wird das Callback ausgeführt. Im Callback on_syserr kann somit ohne Nebenwirkungen der Scriptablauf per `die` oder `exit` beendet werden.

Parameter:
- fehlermeldung: Fehlermeldung mit der main::syserr aufgerufen wurde.

Über dieses Callback können zusätzliche eigene Fehlerbehandlungen realisiert werden.
Beispielsweise können eigene error-Logs angelegt werden, oder eine Email mit der Fehlermeldung versendet werden.

Beispiel stdall.ipl:

```
<!--spmacro:module(Callback)
sub on_syserr {
    my $errormsg = $_[0];

    ## Problematische Zeichen escapen
    $errormsg =~ s/\0/\\0/g;
    $errormsg =~ s/\r/\\r/g;
    $errormsg =~ s/\n/\\n/g;

    main::plog("D, schwerwiegender Fehler: $errormsg\n");
}
-->
```

3.5.7.25 check_write_user

```
<num> Callback::check_write_user()
```

Die Funktion dient als Schalter und steuert, ob ein neuer Benutzer in die Datenbank eingetragen wird oder nicht.

```
Rückgabe 1  Benutzer wird in die Datenbank eingetragen.
Rückgabe 0  Benutzer wird nicht eingetragen (Zurückgegeben Benutzernummer
ist "-1").
```

Copyright 2017 by IBO Internet Software

3.5.7.26 modify_detail_template

```
string <template> Callback::modify_detail_template(string
<template>,string <itemid>, int <subcatalog>)
```

Diese Funktion kann das standardmäßig für Detailseiten aufgerufene Template verändern, indem sie einen anderen Dateinamen zurückgibt. Dies kann möglicherweise sogar abhängig vom Dateinamen, der Artikelnummer oder dem aktuellen Subkatalog erfolgen. Oder wie im nachfolgenden Beispiel von einer Formularvariablen.

```
sub modify_detail_template {
  my $template = $_[0];
  my $itemid   = $_[1];
  my $subcat   = $_[2];
  if (ssp::get_var_form(pdf) ==  1) {
    $template = "html_de/p_detail-pdf.html";
  }
  return $template;
}
```

3.5.7.27 override_template

```
string <template> Callback::override_template(string <template>)
```

Diese Funktion ersetzt Template-Dateien aufgrund des Dateinamens. Hierdurch kann man z.B. leicht unterschiedliche Sets von Templates für verschiedene Anwendungen benutzen.

```
# Beispiele andere Templates für mobile Geräte benutzen
sub override_template {
  my $template = $_[0];
  my ($path,$file) = split(/\//,$template);
  if ($mobile_mode && $path ~= /^html_de/) {
    $path = "mobile_de";
  }
  return "$path/$file";
}
```

3.5.7.28 short_stock_warning

```
short_stock_warning (<produktid>,<product_cnt>,<index>)
```

produktid: Artikelnummer
product_cnt: Anzahl in dieser Bestellung
index: Index im Warenkorb

Diese Funktion wird aufgerufen, wenn sich der Lagerbestand eines Artikels durch einen Kauf vermindert hat und das Feld LBSAUTO definiert ist. Es kann dann überprüft werden, ob der vorhandene Lagerbestand noch ausreichend ist. Falls nicht könnte eine Mail versendet werden. Führt man z.B. ein weiteres Feld "LBSLIMIT" ein, kann man für jeden Artikel die Warngrenze indivduell festelegen.
Mit dem Parameter index können weitere Artikeldaten mit ssp::get_var_cart ausgelesen werden.

3.5.7.29 social_login

```
<num> social_login()
```

Mit dieser Callback-Funktion kann man ein alternatives Login steuern. Der Benutzer wird eingeloggt, wenn die Funktion einen Wert >= 0 zurückgibt.

Copyright 2017 by IBO Internet Software

3.5.7.30 on_placecart

`string <cartline> Callback::on_placecart(string <cartline>)`

Erlaubt die Anpassung der Warenkorbzeile beim Hereinlegen eines Artikels.
Kann beispielsweise benutzt werden um bestimmte maximale Abgabemengen zu erzwingen.
Ein Beispiel dazu finden Sie beim Callback on_changecart.

3.5.7.31 on_changecart

`string <cartline> Callback::on_changecart(string <cartline>)`

Erlaubt die Anpassung der Warenkorbzeile beim Ändern einer Warenkorbposition.
Kann beispielsweise benutzt werden um bestimmte maximale Abgabemengen zu erzwingen.

Beispiel: Eine maximale Abgabemenge von 1 erzwingen
Berücksichtigt das „in den Warenkorb legen“ und auch die Änderung einer Warenkorbposition.

```
sub on_placecart {
    my $cartline = $_[0];
    return modifycartitem($cartline);
}

sub on_changecart {
    my $cartline = $_[0];
    return modifycartitem($cartline);
}

sub modifycartitem {
    my $cartline = $_[0];

    my @token = split(/$main::FD/, $cartline, -1);

    ## ord0 = Artikel
    if ($token[0] eq 'ord0') {
        my $itemid = $token[1];

        ## Artikeldaten aus Datenbank lesen
        my $sth = $main::dbh->prepare_cached("SELECT * FROM ITEM WHERE ITEMID=?");
        $sth->execute($itemid);

        ## Abgabemenge 1 falls benutzerdef. Feld ISLIMITED eine 1 enthält
        my $bIsLimited = 0;
        while (my $pRS = $sth->fetchrow_hashref()) {
            $bIsLimited = $pRS->{ISLIMITED};
        }

        if ($bIsLimited) {
            ## Prüfe Menge und setzte ggfls auf 1 zurück
            if ($token[2] > 1) { ## $token[2] enthält die neue Menge
                $token[2] = 1;

                ## Die hier eingetragenen Fehler können per SSP
                ## in den jew. Vorlagen ausgegeben werden
                $main::FORM{myerrors} = [] if (!$main::FORM{myerrors});
                push(@{$main::FORM{myerrors}},
                    "Die maximale Abgabemenge für diesen Artikel beträgt
                    1 Stück.\nWir haben die Bestellmenge deshalb angepasst
                    und danken für Ihr Verständnis.");
            }
        }
    }

    $cartline = join($main::FD, @token);

    return $cartline;
}
```

Copyright 2017 by IBO Internet Software

3.5.7.32 can_display_user_item

<bool> can_display_user_item(<produktid>,@token)

produktid: Artikelnummer
@token alle Werte aus der Tabelle ITEM für einen Artikel.
$token->[1] ist z.B. auch die Artikelnummer
$token->[8] ist z.B. das Feld Ergänzung2
$token->[28] ist das erste benutzerdef. Feld

Die Funktion sollte 0 oder 1 zurückgeben. Wird eine 0 zurückgegeben,
wird der enstprechende Artikel nicht dargestellt, als wäre er in der Datenbank überhaupt nicht vorhanden.

Im Kapitel „Filtern mit can_display“ gibt es ein Anwendungsbeispiel.

3.5.7.33 can_display_page(@token)

<bool> can_display_page(@token)
@token alle Werte aus der Tabelle PAGE für eine Seite.
$token[0] ist z.B. die Seitennumer
$token[10] ist das erste benutzerdef. Feld

Die Funktion sollte 0 oder 1 zurückgeben. Wird eine 0 zurückgegeben,
wird die enstprechende Seite nicht dargestellt, als wäre sie in der Datenbank überhaupt nicht vorhanden.

Im Kapitel „Filtern mit can_display“ gibt es ein Anwendungsbeispiel.

3.5.7.34 override_zahlartradio_item

```
$outItem = &Callback::override_zahlartradio_item(
      html => $outItem,
      pZahlart => $pZahlart,
      bIsChecked => ($gzahlungsart_selected == $pZahlart->{id}),
   );
```

Hier kann ein eigener HTML-Text pro Option zurückgegeben werden,
um die Darstellung des Radiofeldes zu verändern.

Copyright 2017 by IBO Internet Software

3.6 Automatisch eingebundene Standard-ipls

Es gibt einige ipl-Dateien die SSP-Code enthalten können und automatisch eingebunden werden, sofern Sie im Wurzelverzeichnis des Projekts vorhanden sind. Es ist also hierfür kein include notwendig.

- stdall.ipl – Globale IPL Datei, wir in jede Seite eingebunden.
- stdcart.ipl – Nur für Warenkorb und Bestellung.
- stdplacecart.ipl – Immer wenn etwas in den Warenkorb gelegt wird.

3.6.1.1 stdall.ipl

Hier kann Code stehen, der auf allen Seiten des Shops verwendet werden soll.
Z.B. um globale SSP-Identifier zur Verfügung zu stellen. Oder um globale Callbacks einzurichten.

3.6.1.2 stdcart.ipl

Wird die Bestellung ausgelöst, sollen natürlich die gleichen Berechnungsvorschriften gelten, wie bei der Anzeige im Shop. Es ist hierfür notwendig, dass der SSP Code im ersten erzeugten Dokument ausgeführt wird,. in dem die Identifier genutzt werden sollen. Hierdurch werden alle Funktionen definiert und benötigten Variablen berechnet und stehen dann in allen nachfolgenden Vorlagen zur Verfügung. Eine Ausführung des gleichen Codes in mehreren Vorlagen kann zu Fehlern führen.

Bearbeitungsreihenfolge:

1. Email an Kunden
2. Email an Händler
3. Bestelldatenvorlage
4. Bestätigungsseite (Meldung positiv)

Wenn die definierten Identifier für alle Ausgabeseiten zur Verfügung stehen soll, muss der Code also in der Vorlage für die Email an den Kunden (falls es sich um eine HTML-MAIL handelt) unterzubringen.

Ab der Version 2.20 ist eine Standard-Ipl Datei definiert, die den SSP-code für Warenkorb, Bestellformular und Mails aufnehmen kann. Der Name der Datei ist "**stdcart.ipl**" diese ipl-datei wird, sofern vorhanden, automatisch in alle Vorlagen eingebunden, die mit dem Bestellvorgang zu tun haben. Also Warenkorb, Bestellformulare, Bestellvorlage
und die Emails. Es wird empfohlen, SSP-code der z.B. für die Berechnung von Rabatten vorgesehen ist, in dieser Datei abzulegen. Insbesondere beim versenden von Mails
braucht man nicht mehr darauf zu achten, dass der Code ausschließlich in alle HTML-Mail an den Kunden eingebunden ist.

3.6.1.3 stdplacecart.ipl

In dieser Datei kann Code hinterlegt werden, der ausgeführt wird, wenn ein Artikel in den Warenkorb gelegt wird.

Sie möchten z.B. ein Datum mit drei Listboxen (fürTag, Monat, Jahr) eingegeben wird.
Hierdurch sollen Eingabefehler der Nutzer zu vermieden werden. In der Datenbank soll das Datum allerdings einer benutzerdefinierten Positionsvariablen "DATUM" als Sekunden der Epoche gespeichert werden.

```
<!--spmacro:module(form)      # beliebiger Modulname
  use Time::Local;
  my $artikel = $main::FORM{'article'}; # verstecktes Feld Artikelnummer
  my $t = timelocal(0,0,12,$main::FORM{'day'},
                    $main::FORM{'month'}-1,$main::FORM{'year'} + 100 );
  $main::FORM{"DATUM__$artikel"} = $t; # Positionsvariable vorbelegen
-->
```

Der Code ist hier nicht als Funktion hinterlegt, sondern steht im Modulrumpf. Der Code wird in der Funktion placecart vor jeder anderen Bearbeitung ausgeführt. Damit ist es möglich Eingabewerte aus einem Formular entsprechend umzuformen oder zu korrigieren.

Copyright 2017 by IBO Internet Software

3.7 Anwendungsbeispiele

3.7.1 Artikelspezifischer Rabatt

Aufgabe: Durch einen benutzerdefiniertes Feld pro Artikel "RABKLASSE" wird je Artikel eine Zahl als Rabattklasse definiert. Bleibt das Feld leer, soll der Wert 0 angenommen werden.

Aufgrund der Rabattklasse wird der tatsächliche Wert des Rabatts berechnet.

Beim Artikel soll der Originalpreis und der rabattierte Preis ausgewiesen werden. Dies sowohl bei der Artikeldarstellung als auch im Warenkorb. Die Rabattsumme soll ebenfalls im Warenkorb ausgewiesen werden.

Zusätzlich soll ein Benutzerrabatt in die Berechnung eingehen. Dazu wird beim Benutzer das Feld "KATEGORIE" angelegt. Durch einen Zahlenwert, der mit dem ermittelten Rabatt multipliziert wird, kommt der Gesamtrabatt zustande.

Folgende Vorlagen sind betroffen:

adv_produkt.html (Artikelseitenvorlage)
Hier ist folgenden Zeile einfügen:

```
<!--spmacro:include(rabatt_produkt.ipl)-->
```

Außerdem werden die neuen Identifier in die Artikelzeile eingetragen.

__image__ __produkt__ __favhinzu__ __preis__ __rabatt::rwert__ __rabatt::rpreis__ __detail__ __id__

Script **rabatt_produkt.ipl**. Dieses Script realisiert die Identifier **rabatt::rwert** (% Wert) und **rabatt::rpreis** (Rabatt Preis). Zu Beginn wird das Script **defs.ipl** eingebunden. Hier wird über eine Tabelle die Zuordnung zwischen RABKLASSE und Prozentwert des Rabatts definiert. Dies hat den Vorteil, dass der Prozentwert für eine Klasse einer Stelle global änderbar ist.

```
<!--spmacro:include(defs.ipl)-->

<!--spmacro:module(rabatt)
sub rpreis {
       my $index = $_[0];
       my $anzahl =  $_[1];
       if ($index <= $anzahl) {
              my $wert = @defs::rabatt[ssp::get_var_article('RABKLASSE',$index)] *
                   ssp::get_var_user('KATEGORIE');
              if ($wert > 0 ) {
                     main::mprint( &main::fp((1-$wert) *
           ssp::get_var_article('preis',$index))) ;
              }
       }
       return 0;
}

sub rwert() {
       my $index = $_[0];
       my $anzahl =  $_[1];
       if ($index <= $anzahl) {
              my $wert = @defs::rabatt[ssp::get_var_article('RABKLASSE',$index)] *
                   ssp::get_var_user('KATEGORIE');
              if ($wert > 0 ) {
                     main::mprint( $wert * 100 . "%") ;
              }
       }
       return 0;
}
-->
```

Weiterhin sind alle Vorlagen, die die Bestellung und den Warenkorb betreffen zu modifizieren.

adv_cart.html (Warenkorb)

Copyright 2017 by IBO Internet Software

adv_order.html (Bestellformular)
adv_oderdata.html (Bestelldatenvorlage oder alternativ in **einer** HTML-Bestellmail Vorlage)

Hier ist folgenden Zeile einfügen:

```
<!--spmacro:include(rabatt_callback.ipl)-->
```

Das Modul Callback mit der Funktion add_invoice_position sorgt dafür, dass die ermittelte Rabattsumme tatsächlich vom Gesamtbetrag abgezogen wird.

Das Modul rabatt ist hier etwas anders als bei den Artikelseiten realisiert, da es auf die Variablen des Warenkorbs zugreift.

Ihr bisheriger Einkauf

Produkt-Nr.	***Bezeichnung***	***Stückzahl***	***Einzelpreis***	***Gesamtpreis***
__id__ __zusatz1__ __zusatz2__	__produkt__	__anzaendern__	__preis__ __rabatt::rpreis__	__gesamteinzel__ __rabatt::rgesamt__
			__versandtext__	__versand__
			__rabatttext__	__rabatt__
			Kundenrabatt	__rabatt::rsumme__
Netto:__gesamtnetto__		__gesamtanzahl__		__**gesamtpreis**__

__logo__ __wkloeschen__ __inhalt__ __zurueck__ __bestellen__

```
<!--spmacro:include(defs.ipl)-->

<!--spmacro:module(Callback)

$rabattsumme = 0;

sub invoice_addpos {
       my $ref_gesamt = $_[0];
       my $ref_gesamt_steuer = $_[2];

       my $anzahl = ssp::get_cart_items();

       my $ust = $_[3]/100;
       my $netto = $_[4];

       for (my $i = 0;$i <= $anzahl;++$i) {
              my $wert = @defs::rabatt[ssp::get_var_cart('RABKLASSE',$i)] *
                     ssp::get_var_user('KATEGORIE');
              $rabattsumme +=  $wert * ssp::get_var_cart('preis',$i) *
                     ssp::get_var_cart('anzahl',$i);
       }
       if ($netto > 0) {
              $$ref_gesamt -=  $rabattsumme;
              $$ref_gesamt_steuer -=  $rabattsumme*$ust;
       } else {
              $$ref_gesamt -= $rabattsumme;
              $$ref_gesamt_steuer -=  $rabattsumme*(1-1/(1+$ust));
       }
       return 0;
}
-->


<!--spmacro:module(rabatt)

sub rpreis {
       my $index = $_[0];
       my $anzahl =  $_[1];

       if ($index <= $anzahl) {
              my $wert = @defs::rabatt[ssp::get_var_cart('RABKLASSE',$index)] *
```

Copyright 2017 by IBO Internet Software

```
                  ssp::get_var_user('KATEGORIE');
              if ($wert > 0 ) {
                    main::mprint( &main::fp((1-$wert) *
ssp::get_var_cart('preis',$index))) ;
              }
       }
       return 0;
}

sub rgesamt {
       my $index = $_[0];
       my $anzahl =  $_[1];

       if ($index <= $anzahl) {
              my $wert = @defs::rabatt[ssp::get_var_cart('RABKLASSE',$index)] *
                   ssp::get_var_user('KATEGORIE');

              if ($wert > 0 ) {
                    main::mprint( &main::fp((1-$wert) *
ssp::get_var_cart('preis',$index) *
                         ssp::get_var_cart('anzahl',$index))) ;
              }
       }
       return 0;
}

sub rwert() {
       my $index = $_[0];
       my $anzahl =  $_[1];
       if ($index <= $anzahl) {
              my $wert = @defs::rabatt[ssp::get_var_cart('RABKLASSE',$index)] *
                   ssp::get_var_user('KATEGORIE');
              if ($wert > 0 ) {
                    main::mprint( $wert * 100 . "%") ;
              }
       }
       return 0;
}

sub rsumme  {
       main::mprint(main::fp($Callback::rabattsumme));
       return 0;
}

-->
```

3.7.2 Staffelrabatt nach Warenwert

Aufgabe: Durch einen benutzerdefiniertes Feld ("KATEGORIE") pro Kunde kann dem Kunden eine Staffelrabattklasse zugewiesen werden.
Der Rabatt auf den Warenwert wird dann durch eine Rabattliste berechnet. In der Rabattliste sind die verschiedenen Preisstufen mit den entsprechenden Rabatten hinterlegt. Durch das Feld Kategorie kann aus verschiedenen definierten Listen gewählt werden.

Es sind folgende Vorlagen betroffen:
adv_cart.html (Warenkorb)
adv_order.html (Bestellformular)
adv_oderdata.html (Bestelldatenvorlage oder alternativ in **einer** HTML-Bestellmail Vorlage)
Natürlich können Sie die Auflistung des Rabatts zusätzlich in beliebige andere Seiten übernehmen.

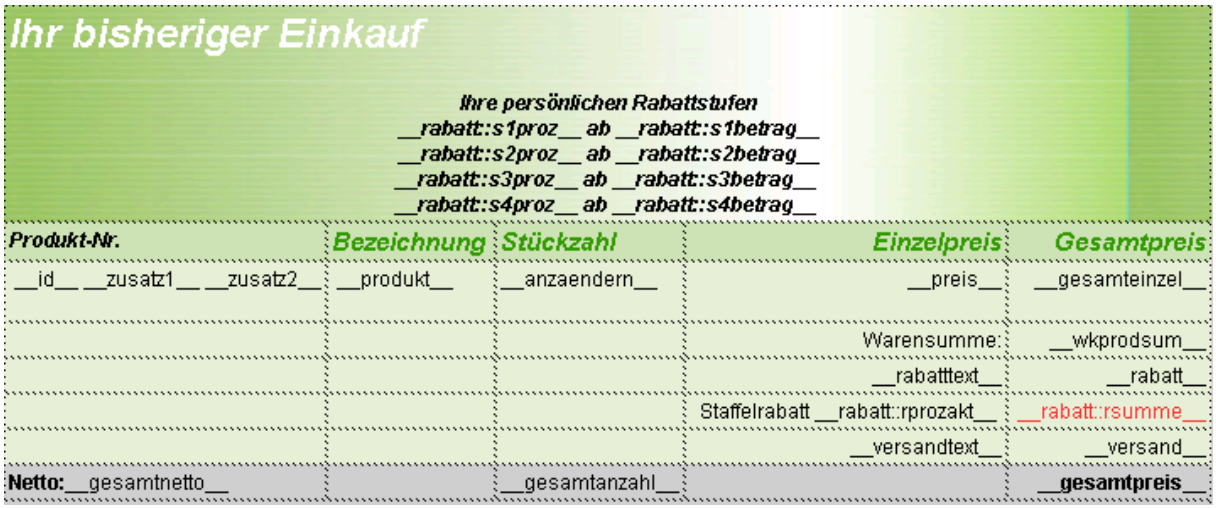
Hier jeweils folgende Zeile einfügen:

```
<!--spmacro:include(rabatt.ipl)-->
```

Die Zeile ist am besten vor dem BODY-Tag einzufügen.

Die neuen Identifier für den Warenkorb. Bestellformular, HTML-Mail und Bestelldatenvorlage analog. Die Anzeige der Rabattwerte können Sie natürlich analog auch auf anderen Seiten verwenden.

Copyright 2017 by IBO Internet Software



Ihr bisheriger Einkauf

Ihre persönlichen Rabattstufen
__rabatt::s1proz__ ab __rabatt::s1betrag__
__rabatt::s2proz__ ab __rabatt::s2betrag__
__rabatt::s3proz__ ab __rabatt::s3betrag__
__rabatt::s4proz__ ab __rabatt::s4betrag__

Produkt-Nr.	Bezeichnung	Stückzahl	Einzelpreis	Gesamtpreis
__id__ __zusatz1__ __zusatz2__	__produkt__	__anzaendern__	__preis__	__gesamteinzel__
			Warensumme:	__wkprodsum__
			__rabatttext__	__rabatt__
			Staffelrabatt __rabatt::rprozakt__	__rabatt::rsumme__
			__versandtext__	__versand__
Netto:__gesamtnetto__		__gesamtanzahl__		__gesamtpreis__

Die Rabattlisten werden in dem Modul defs.ipl bereitgestellt. Hier sind vier Staffeln für 3 Kundenkategorien definiert. Jede Staffel besitzt eine Untergrenze, eine Obergrenze und den Rabattsatz in Prozent.

```
<!--spmacro:module(defs)
#verschiedene Staffeln abhängig von der Kunden-Rabattkasse
$staffelrab = [
              [[100,200,1.0],[200,300,2.0],[300,600,3.0],[600,99999999,4.0]],
              [[100,200,1.5],[200,300,2.5],[300,500,4.0],[500,99999999,5.0]],
              [[100,250,0.5],[250,350,3.5],[350,750,4.5],[750,99999999,5.5]]
              ];
-->
```

Der Wert Kategorie beim Kunden, kann hier Werte zwischen 0 und 2 haben. Der Wert 0 entspricht übrigens auch einem leeren Feld beim Kunden. Es ist auch möglich eine andere Anzahl von Staffelstufen für jede Zeile zu verwenden.

```
Soll z.B. der Wert 0 für Kategorie keinen  Staffelrabatt bewirken, wäre dies z.B.
mit folgender Tabelle möglich:

$staffelrab = [
              [[99999999,99999999,0.0]],
              [[100,200,1.5],[200,300,2.5],[300,500,4.0],[500,99999999,5.0]],
              [[100,250,0.5],[250,350,3.5],[350,750,4.5],[750,99999999,5.5]]
              ];
```

Das Script rabatt.ipl stellt schließlich die benötigten Identifier zur Verfügung. Die Berechnung des Rabatts erfolgt im Callback-Modul mit der Funktion invoice_addpos().
Diese Funktion wird aufgerufen, wenn der Warenkorb neu berechnet wird.

```
<!--spmacro:include(defs.ipl)-->
<!--spmacro:module(rabatt)

sub s1proz {show_proz(0);}
sub s2proz {show_proz(1);}
sub s3proz {show_proz(2);}
sub s4proz {show_proz(3);}
sub s1betrag {show_betrag(0);}
sub s2betrag {show_betrag(1);}
sub s3betrag {show_betrag(2);}
sub s4betrag {show_betrag(3);}

sub show_proz {
       my $staf = $_[0];
       my $kat = ssp::get_var_user('KATEGORIE');
       main::mprint($defs::staffelrab->[$kat][$staf][2] . " %") ;
       return 0;
}
```

Copyright 2017 by IBO Internet Software

```
sub show_betrag {
       my $staf = $_[0];
       my $kat = ssp::get_var_user('KATEGORIE');
       main::mprint(main::fp($defs::staffelrab->[$kat][$staf][0])) ;
       return 0;
}

sub rsumme  {
       main::mprint(main::fp($Callback::rabattsumme));
       return 0;
}
```

Copyright 2017 by IBO Internet Software

```
sub rprozakt  {
       main::mprint($Callback::prozentaktuell . " %");
       return 0;
}
-->

<!--spmacro:module(Callback)

$rabattsumme = 0;
$prozentaktuell = 0;

sub invoice_addpos {
       my $ref_gesamt = $_[0];
       my $ref_gesamt_steuer = $_[2];
       my $ust = $_[3]/100;
       my $netto = $_[4];

       my $kat = ssp::get_var_user('KATEGORIE');
       my $wkprodsum = ssp::get_cart_wkprodsum();

       foreach $i (@{$defs::staffelrab->[$kat]}) {
              if (($i->[0] <= $wkprodsum) && ($wkprodsum < $i->[1])) {
                $rabattsumme = $i->[2]/100 *  $wkprodsum;
                $prozentaktuell = $i->[2];
              }
       }
       if ($netto > 0) {
              $$ref_gesamt -=  $rabattsumme;
              $$ref_gesamt_steuer -=  $rabattsumme*$ust;
       } else {
              $$ref_gesamt -= $rabattsumme;
              $$ref_gesamt_steuer -=  $rabattsumme*(1-1/(1-$ust));
       }
}
-->
```

3.7.3 Artikelabhängige Versandkosten

Aufgabe: Die Versandkosten sollen abhängig von einer Artikelgröße berechnet werden, dies kann z.B. das Gewicht oder Volumen sein. Auch Kombinationen aus mehreren Größen lassen sich leicht realisieren.

Für das Gewicht wird das benutzerdefinierte Feld 'GEWICHT' angelegt. Hier kann für jeden Artikel das wirksame Gewicht angegeben werden.

In der Artikelseite und im Warenkorb kann das Gewicht mit dem Identifier __GEWICHT__ ausgewiesen werden.

Im Warenkorb schließlich muss aus dem Gewicht die Versandkosten berechnet werden. Dies geschieht auf Basis einer Tabelle verskosttab in defs.ipl.

```
<!--spmacro:module(defs)
#Versandkosten abhängig vom Gewicht
$verskosttab = [
              [0,2,3.50],[2,8,5],[8,18,8.2],[18,40,12],[40,999999,25]
                ];
#Beispiel: [0,2,3] = von 0 bis 2 KG 3,50 EUR Versandkosten
-->
```

Die Funktionen zur Berechnung der Werte und zum Hinzufügen in die Rechnung befindet sich in der Datei "versk_callback.ipl".

```
<!--spmacro:include(defs.ipl)-->
<!--spmacro:module(Callback)

$vsumme = 0;
$gewichtsumme =0;

sub invoice_addpos {
```

Copyright 2017 by IBO Internet Software

```
	my $ref_gesamt = $_[0];
	my $ref_gesamt_steuer = $_[2];

	my $anzahl = ssp::get_cart_items();

	my $ust = $_[3]/100;
	my $netto = $_[4];

	for (my $i = 0;$i <= $anzahl;++$i) {
		my $wert =
ssp::get_var_cart('anzahl',$i)*ssp::get_var_cart('GEWICHT',$i);
		$gewichtsumme +=  $wert;
	}
	foreach my $i (@{$defs::verskosttab}) {
		ssp::trace("$i->[0] $i->[1] $i->[2]\n");

		if (($i->[0] <= $gewichtsumme) && ($gewichtsumme < $i->[1])) {
		  $vsumme = $i->[2];
		  break;
		}
	}

	if ($netto > 0) {
		$$ref_gesamt +=  $vsumme;
		$$ref_gesamt_steuer +=  $vsumme*$ust;
	} else {
		$$ref_gesamt += $vsumme;
		$$ref_gesamt_steuer +=  $vsumme*(1-1/(1-$ust));
	}
	ssp::trace("+++++ invoice_addpos $$ref_gesamt $$ref_gesamt_netto
$$ref_gesamt_steuer\n");
}
-->

<!--spmacro:module(versk)

# Gesamtsumme des Gewichts
sub gesgew {
main::mprint($Callback::gewichtsumme);
return 0;
}

# Gesamtgewicht einer Position
sub gewicht {
	my $index = $_[0];
	my $anzahl =  $_[1];

	if ($index <= $anzahl) {

	main::mprint(ssp::get_var_cart('anzahl',$index)*ssp::get_var_cart('GEWICHT',
$index));
	return 0;
}

# Gesamtsumme der Versandkosten
sub versk  {
	main::mprint(main::fp($Callback::vsumme));
}

-->
```

Ihr bisheriger Einkauf

Produkt-Nr.	Bezeichnung	Stückzahl	Einzelpreis	Gesamtpreis
__id__ __zusatz1__ __zusatz2__	__produkt__	__anzaendern__	__preis__ __GEWICHT__ KG __rabatttext__	__gesamteinzel__ __versk::gewicht__ KG __rabatt__

Copyright 2017 by IBO Internet Software

		Versandkosten	__versk::gesgew__ KG	__versk::versk__
Netto:__gesamtnetto__		__gesamtanzahl__		**__gesamtpreis__**
__logo__	__wkloeschen__	__inhalt__	__zurueck__	__bestellen__

3.7.4 Zusätzliche Artikelbilder

Möchte man für einige Artikel neben dem Detailbild weitere Artikelbilder darstellen und diese mit ShopPilot verwalten, bieten sich hierfür benutzerdefinierte Datenfelder vom Typ "Bilddatei" an. In diesen Datenfeldern wird lediglich der Pfad der Bilddatei festgehalten.
Gewünscht ist eine Darstellung, wo jedoch der komplette HTML-Bereich für ein zusätzliches Bild hinzugefügt wird, aber nur wenn dieses auch vorhanden ist.. Dies kann mit einer kleinen SSP-Funktion erreicht werden.

Folgender Code ist z.B. in die Detailseite oder Artikelseite einzubinden, wenn das Datenfeld für die Bilddatei BILD3 heißt:

```
<!--spmacro:module(bild)
 sub b3 {
   my $index = $_[0];
   my $anzahl =  $_[1];
   if ($index <= $anzahl) {
      my $b = ssp::get_var_article('BILD3',$index);
      if (length($b) > 1) {
         $b = qq{<p><img src="$b"></p>};
         main::mprint($b);
      }
   }
   return 0;
}
-->
```

In der Vorlage trägt man dann an die Stelle, wo der Absatz mit dem Bild stehen soll einfach __bild::b3__ ein.

3.7.5 Seitenhierarchie als Pfad

Mit diesem Modul ist es möglich die Seitenhierarchie als Pfad auf Artikelseiten anzugeben. Wobei die einzelnen Elemente zum direkten Aufruf der übergeordneten Seiten dienen können.

Z.B. **Verbrauchsmaterial::Drucker::HP::Laser** wäre ein solcher Pfad der angezeigt werden kann.

```
<!--spmacro:module(kat)
sub ap {
  my $index = $_[0];
  my $pagenr = ssp::get_var_article("ipagenr",0);
  my $text;
  while( int($pagenr) > 0) {
        my $title = ssp::get_var_page("ptitle",$pagenr);
        $text = "::<a href =
\"$ssp::xxpath?show$pagenr,$ssp::xxsession\">$title</a>" . $text;
        $pagenr = ssp::get_var_page("parent_pagenr",$pagenr);
  }
  main::mprint($text);
  return 0;
}
-->

__kat::ap__
```

Hier wird aufgrund des ersten Artikels in der Seite die Seitennummer bestimmt
`my $pagenr = ssp::get_var_article("ipagenr",0);`

In der while-Schleife wird der Link und der Text gebildet und die neue Seitennummer ermittelt. Diese ergibt sich aus der Vaterseite der aktuellen Seite.
`$pagenr = ssp::get_var_page("parent_pagenr",$pagenr);`

Copyright 2017 by IBO Internet Software

Die Schleife wird so lange durchlaufen, bis die Vaterseite den Wert 0 hat, dann ist man auf der ersten Ebene angelangt.

3.7.6 Verwandte Artikel aus Suchergebnis

Im Suchergebnis kann es wünschenswert sein, alle Artikel der zugehörigen Produktgruppe zu einem Artikel direkt aufrufbar zu machen. Dazu dient die folgende Funktion., sie erzeugt einen Link auf die zugehörige Seite (Produktgruppe)

```
<!--spmacro:module(kat)
sub p {
  my $index = $_[0];
  my $pagenr = ssp::get_var_article("ipagenr",$index);
  my $title = ssp::get_var_page("ptitle",$pagenr);
  main::mprint ("<a href = \"$ssp::xxpath?show$pagenr,$ssp::xxsession\">
$title  </a>");
 return 0;
}
-->
```

__kat::p__ Zeigt den Link zur Kategorie bei Artikeln im Suchergebnis an .

3.7.7 Angabe des Rechnungsdatums

Soll in der Bestätigungsmail auch ein Rechnungsdatum ausgewiesen werden, kann man dies mit einer kleinen SSP-Routine erreichen.
Die Definition der SSP Routine wird im Head-Bereich der HTML-Seite als HTML-Kommentar (s.u.) eingebaut. Es ist eigentlich nur wichtig, dass sie vor dem ersten Aufruf definiert wird. Wenn man sie also vor dem <body> Tag schreibt ist diese Bedingung in jedem Fall erfüllt.

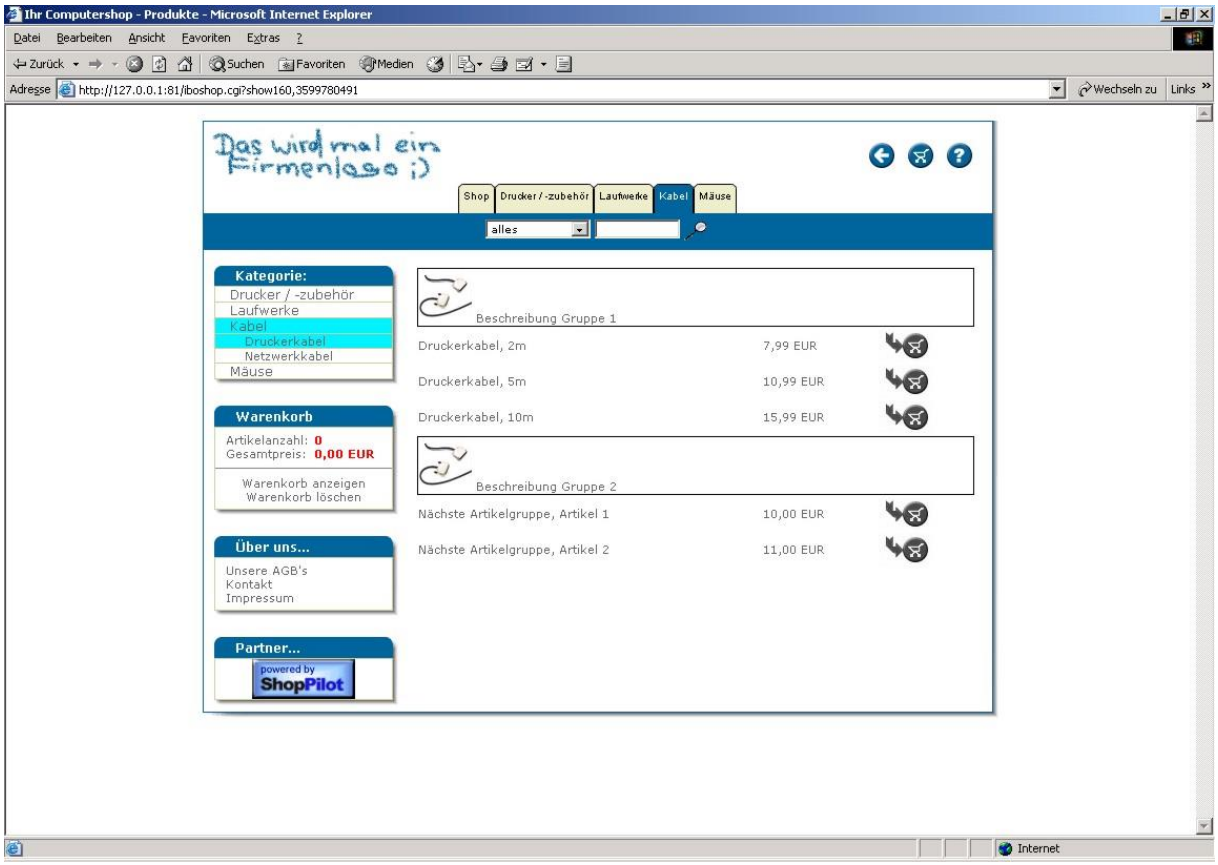
```
<!--spmacro:module(date)
sub today {
  ($sec,$min,$hour,$mday,$mon,$year) = localtime(time);
  $mon += 1; $year += 1900; # Y2K
  main::mprint sprintf("%2d.%2d.%4d",$mday,$mon,$year);
  return 0;
}
-->
```

Die Anzeige des Datums erfolgt dann einmal an beliebiger Stelle in der HTML-Seite indem man __date::today__ schreibt.

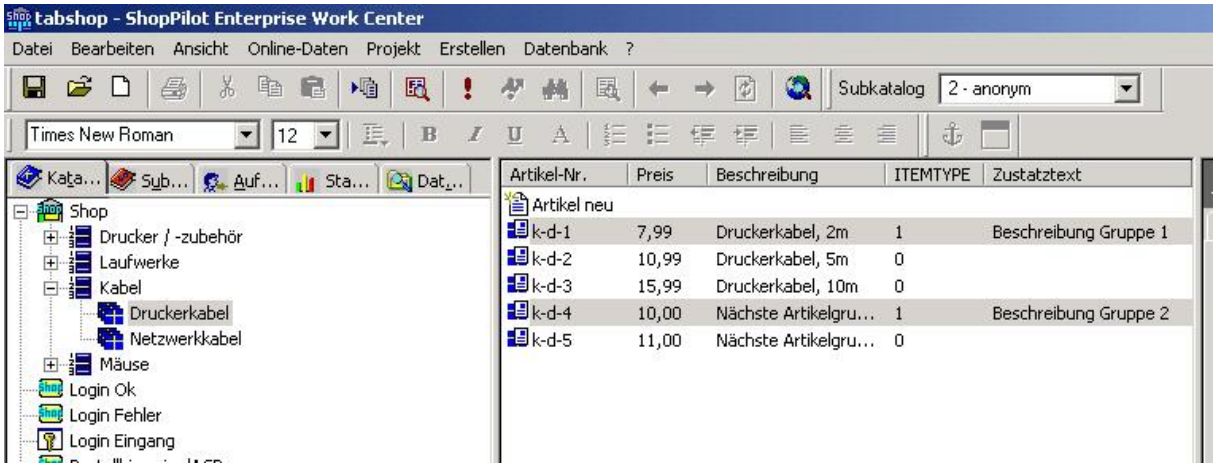
3.7.8 Artikel gruppieren

Die Artikel auf Artikelseite sollen gruppiert werden. Dazu wir der erste Artikel einer Gruppe anders als die folgenden Artikel der Gruppe dargestellt. Zur Unterscheidung wird in ShopPilot ein benutzerdefiniertes Feld angelegt (hier ITEMTYPE), mit dem der erste Artikel einer Gruppe gekennzeichnet wird.

Copyright 2017 by IBO Internet Software



Nötige Schritte:
Als erstes wird ein benutzerdefiniertes Feld für Artikel (Typ: Text, Zahl) angelegt. Dann muss man die Artikel innerhalb von ShopPilot gruppieren. In diesem Beispiel werden 2 Gruppen von Druckerkabeln angelegt. Derjenige Artikel, der der erste einer Gruppe sein soll, erhält als Wert für ITEMTYPE eine 1. Alle weiteren zur Gruppe gehörenden Artikel erhalten eine 0. Es ist also zwingend erforderlich die Artikel genauso zu sortieren, wie sie später ausgegeben werden sollen.
Um die Überschriften der Gruppen festzulegen, wird das Feld Zusatztext benutzt. Hier kann aber auch jedes andere Feld benutzt werden.



Nun wird die SSP Funktion erstellt. Dazu wird eine Datei namens "**itemgroup.ipl**" angelegt, in die folgender Code geschrieben wird:

```
<!--spmacro:module(itemgroup)
sub prod {
	my $index = $_[0];
	my $anzahl = $_[1];

	my $type = ssp::get_var_article('ITEMTYPE', $index);
```

Copyright 2017 by IBO Internet Software

```
	my $html = "";

	if ($type == 0) {
		# hier der HTML-Code für Folgeartikel
		$html = qq{
			<tr>
				<td>__produkt__</td>
				<td>__preis__</td>
				<td>__id__</td>
			</tr>
		};

	} elsif ($type == 1) {
		# hier der HTML-Code für den Kopfartikel
		$html = qq{
			<tr>
				<td colspan="3" style="border: #000000 solid 1px;">
					__image__ __zusatztext__
				</td>
			</tr>
			<tr>
				<td>__produkt__</td>
				<td>__preis__</td>
				<td>__id__</td>
			</tr>
		};

	}
	main::mprint($html);
}
-->
```

Innerhalb des Bereiches der mit **qq{** geöffnet und mit **};** geschlossen wird steht der HTML-Code einschließlich der Identifier. Dieser ist vom Aufbau identisch mit dem Code der sonst direkt im Body stehen würde.

Diese Funktion erzeugt die Darstellung einer Produkt-Zeile einer Tabelle.
Wenn nötig wird im Falle Typ = 1 eine Überschriftzeile vor dem eigentlichen Artikel platziert.

Um diese Funktion zu nutzen, muss die ipl-Datei per
<!--spmacro:include(itemgroup.ipl)-->
in die Vorlagendatei für Produktseiten eingebunden werden.

Der letztlich Aufruf erfolgt in dieser Art:

```
<table>
	<!--spmacro:loopitem-->
		__itemgroup::prod__
	<!--spmacro:enditem-->
</table>
```

3.7.9 Datei-Upload

Es soll eine beliebige Datei hochgeladen werden.
Dazu kann folgendes Formular benutzt werden:

```
<form method="POST" enctype="multipart/form-data" name="send"
action="__xxpath__?showbynameUploadFormular,__xxsession__">
	<input name="datei" type="file">
</form>
```

Nachdem dieses Formular abgeschickt wurde, können mittels SSP alle notwendigen Informationen abgerufen werden. Per ssp::get_var_form(*"Name des Input-Feldes"*) kann der Name der hochgeladenen Datei abgerufen werden und per ssp::get_var_form_filedata(*"Name des Input-Feldes"*) kann der zugehörige Dateiinhalts ermittelt werden.

Beispiel:
Die empfangene Datei wird auf dem Server gespeichert.

Copyright 2017 by IBO Internet Software

```
sub SaveUploadedFile {
       my $filename = ssp::get_var_form('datei');
       my $filecontents = ssp::get_var_form_filedata('datei');
       # Lokaler Pfad auf dem Server, unter dem gespeichert werden soll
       # kann auch relativ zu iboshop.cgi sein
       my $saveto = "../shop/uploads/$filename";
       open( FILEHANDLE, ">$saveto" );  # Datei anlegen
       binmode FILEHANDLE;              # Im Binämodus schreiben
       print FILEHANDLE $filecontents;  # Inhalt in Datei schreiben
       close( FILEHANDLE );             # Datei schliessen
}
```

3.7.10 XTabs - Erweiterte Tabs

Die XTabs stellen eine Möglichkeit dar, individuellere Reiter darzustellen wie die ursprünglichen Tabs (__tab__).
Es ist nun möglich beliebige HTML-Konstrukte als einzelnen Tab anzugeben.
Ein enormer Vorteil dieser Methode besteht darin, dass CSS-Styles verwendet werden können und die Tabs jede beliebige Form annehmen können! Die Tabs müssen nicht mal mehr aussehen wie Reiter.
Ausserdem können die Tabs auf diese Weise nicht nur horizontal ausgegeben werden, sondern auch vertikal.

Die prinzipielle Funktionsweise ist dabei die gleiche wie bei den alten Tabs.
Die Tabs bilden die Kategorien der Hauptebene in Form von Navigationselementen ab deren Aussehen Sie vorgeben.

3.7.10.1 Definieren der Tabs

Zur Ausgabe der Tabs benötigen Sie zwei Textdateien.
In der einen Datei wird der aktive Tab definiert und in der anderen werden die inaktiven Tabs definiert.
Die Dateien können frei benannt werden, müssen im Shop aber als zusätzliche Dateien angegeben werden!

Der Aufbau der Dateien ähnelt dem der embedded Templates. Sie können beliebige HTML-Konstrukte verwenden, sollten jedoch dabei beachten, dass nur Ausschnitte einer HTML-Datei zulässig sind, die für die Darstellung notwendig sind. Also keine Elemente wie, <html> <head> <body> etc.
Die folgenden Platzhalter können innerhalb dieser Vorlagendateien benutzt werden, um Seiten-Informationen auszugeben:

`__xtab_link__`	Link auf die Seite
`__xtab_descr__`	Beschreibung
`__xtab_title__`	Seitentitel
`__xtab_img__`	Bild der Seite
`__xtab_icon__`	Icon der Seite
`__xtab_count__`	Nummer des Tabs (um eine eigne ID für CSS zu vergeben)

Hinweis:
Beachten Sie, dass diese Platzhalter ausschließlich in den Vorlagendatei für die XTabs benutzt werden können und nicht an anderen Stellen im Shop.

Beispiel:

Ein Template um einen solchen Tab zu erzeugen, könnte folgendermaßen aussehen.
Das legt im Beispiel den aktiven Tab fest und wird als **active.txt** im Unterordner **xtabs** des Projektverzeichnisses gespeichert.

```
<div style=" background-image:url(design/hintergrund.jpeg);
background-repeat:repeat-x;
background-color:#1369B1;
font-weight:bold;
```

Copyright 2017 by IBO Internet Software

```
padding:7px;">
<a href="__xtab_link__">__xtab_title__</a>
</div>
```

Das Template für die inaktiven Tabs wird als **inactive.txt** im Unterordner **xtabs** des Projektverzeichnisses gespeichert und enthält folgenden Code:

```
<div style=" background-image:url(design/hintergrund_grau.jpeg);
background-repeat:repeat-x;
background-color:# AAAAAA;
padding:6px;">
<a href="__xtab_link__">__xtab_title__</a>
</div>
```

Tip:
Die Angabe background-image legt ein Hintergrundbild fest.
Geben Sie dort den Pfad zur Datei (relativ zum Projektverzeichnis) an, das als Hintergrundbild für den jeweiligen Reiter verwendet werden soll.

Ausgeben der Tabs

Sie können die Tabs mittels SSP in jeder beliebigen HTML-Vorlage ausgeben!
Zu Ausgabe eines Tabs kann die SSP-Funktion ssp::xtab benutzt werden.

```
ssp::xtabs(String aktiv, String inaktiv);
```

Dieser Funktion wird als erster Parameter der Pfad zur Vorlagendatei für den aktiven Tab übergeben und als zweiter Parameter der Pfad Vorlagendatei für die inaktiven Tabs.
Beide Pfade sind relative Pfade zum Projektverzeichnis.

Beispiel:
Im folgenden Beispiel wurden die Vorlagendateien als **active.txt** und **inactive.txt** bezeichnet.
Sie befinden sich im Unterordner **xtabs** des Projektverzeichnisses.

```
<!--spmacro:module(xtabs)
ssp::init_xtabs();
sub print {
      my $templActive     = "xtabs/active.txt";
      my $templInactive   = "xtabs/inactive.txt";
      ssp::xtab($templActive, $templInactive);}
-->
```

Der Aufruf dieser Funktion erfolgt dann per

```
      __xtabs::print__
```

Beachten Sie, dass Sie die Funktion auf einer Seite beliebig oft aufrufen können.
Es werden nur soviele Tabs ausgegeben, wie Hauptkategorien existieren.
Alle weiteren Identifier werden durch einen leeren Text ersetzt.
Es bietet sich also an, immer einige Identifier zuviel einzutragen!
Wichtig: Die Funktion *ssp::init_xtabs()* muss einmalig aufgerufen werden bevor der erste XTab ausgegeben wird.

Beispiel:

```
<table cellspacing="0">
<tr>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
      <td>__xtabs::print__</td>
</tr>
```

Copyright 2017 by IBO Internet Software

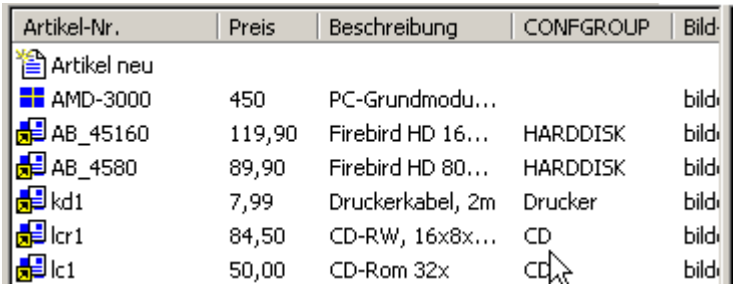
```
</table>
```

3.7.11 Produktkonfigurator

Wenn man dem Kunden ermöglichen will, eine individuelle Konfiguration für ein Produkt zu erstellen, wird ein Produktkonfigurator benötigt. Dieser ermöglicht das Produkt aus verschiedenen Teilprodukten zusammenzustellen. Der Kunde erhält damit die Freiheit seine Wunschkonfiguration zusammenzustellen und hat außerdem die Garantie, dass alles zusammen passt.

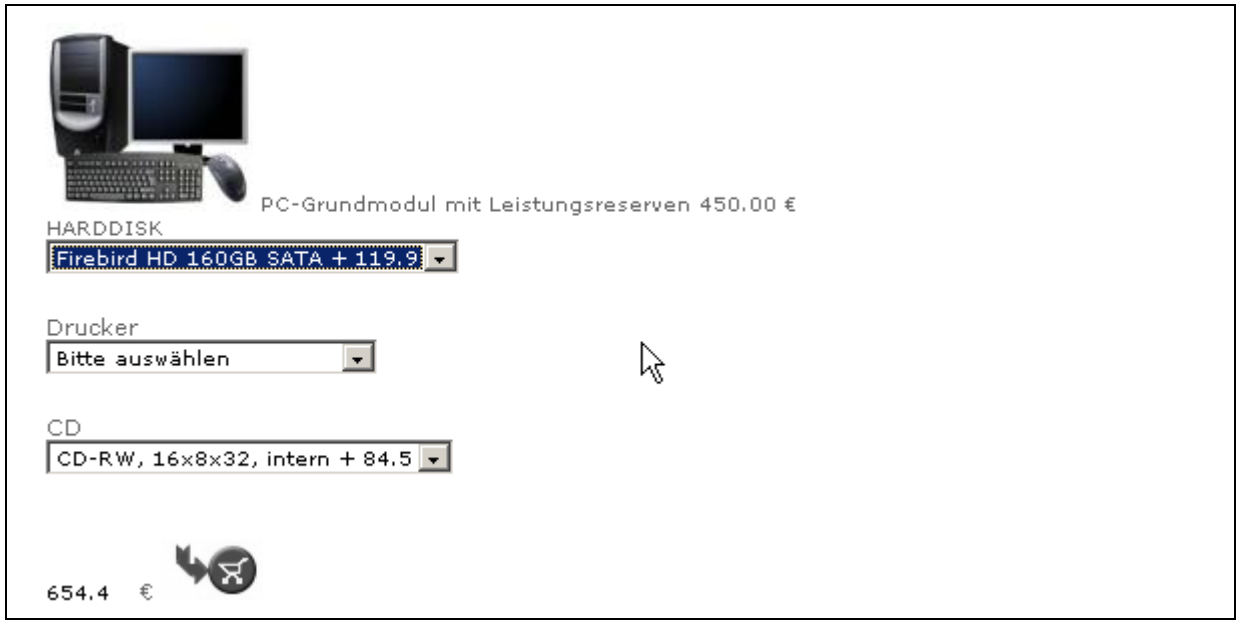
3.7.11.1 Prinzip eines einfachen Produktkonfigurators

Die Artikel, die bei der Konfiguration verwendet werden, werden auf einer Artikelseite hinterlegt. Es wird ein benutzerdefiniertes Feld "CONFGROUP" eingerichtet. Hier wird jeweils die Gruppe eingetragen. Falls es einen Grundartikel gibt, erhält dieser keinen Eintrag.
Artikel mit gleicher Gruppe werden als Liste dargestellt, aus der der Kunden einen Artikel auswählen kann.



Artikel-Nr.	Preis	Beschreibung	CONFGROUP	Bild
Artikel neu				
AMD-3000	450	PC-Grundmodu...		bild
AB_45160	119,90	Firebird HD 16...	HARDDISK	bild
AB_4580	89,90	Firebird HD 80...	HARDDISK	bild
kd1	7,99	Druckerkabel, 2m	Drucker	bild
lcr1	84,50	CD-RW, 16x8x...	CD	bild
lc1	50,00	CD-Rom 32x	CD	bild

Die Shopseite erscheint dann z.B. so:



Der SSP Code kann in der Seite hinterlegt werden:

```
<!--spmacro:module(conf)
sub list {
       $index = $_[0];
       $anz =  $_[1];
       my $preis = ssp::get_var_article("preis",0);

       my $out = qq{<script>
       var arraypreis = new Array();
       var arrayid = new Array();
       var arraygroup = new Array();

       function selchange(me) {
              var txt;
              var group;
```

Copyright 2017 by IBO Internet Software

```
		var preis = $preis;

		for (var i = 1; i < arraypreis.length ; i++) {
			actgroup = arraygroup[i];
			var aanz = eval('document.conf.a__' + arrayid[i]);
			aanz.value = 0;
			if (group != actgroup) { // neues select
				// ist etwas selektiert
				var list = eval('document.conf.' + actgroup);
				if (list.selectedIndex) {
					index = list.options[list.selectedIndex].value;
					if (index > 0 && (index == i)) {
						preis += arraypreis[index];
						group = actgroup;
						aanz.value = 1;
					}
				}
			}
		}
		document.conf.preis.value = Math.round(preis*100)/100;
	}
	</script>\n};

	$out .= qq{<form method="post" name="conf" action="__xformcart__">\n
	__image__ __produkt__  __preis__<br>
	};
	my $i;
	my $group;
	my $first = 1;
	for($i=0; $i <= $anz; ++$i) {
		my $group = ssp::get_var_article("CONFGROUP",$i);
		my $artnr = ssp::get_var_article("itemid",$i);
		if (length($group)) {
			$out .= qq{<input type=hidden name="a__$artnr" value="0">\n};

		} else {
			$out .= qq{<input type=hidden name="a__$artnr" value="1">\n};
		}
	}
	$out .= "<script> \n";
	for($i = 1; $i <= $anz; ++$i) {
		my $actgroup = ssp::get_var_article("CONFGROUP",$i);
		my $artnr = ssp::get_var_article("itemid",$i);
		my $preis = ssp::get_var_article("preis",$i);
			$out .= qq{arraypreis[$i] = $preis; arrayid[$i] = '$artnr';
arraygroup[$i] = '$actgroup';\n};
	}
	$out .= "</script>\n";
	for($i = 1; $i <= $anz; ++$i) {
		my $actgroup = ssp::get_var_article("CONFGROUP",$i);
		my $artnr = ssp::get_var_article("itemid",$i);
		my $preis = ssp::get_var_article("preis",$i);
		my $descr = ssp::get_var_article("idescription",$i);

		if ($actgroup ne $group) {
			if (!$first) {
				$out .= qq{</select><p>\n};
			}
			$out .= qq{$actgroup<br>\n<select name="$actgroup"
onChange="selchange(this)">
<option value="">Bitte auswählen</option>
<option value="$i">$descr + $preis</option>\n};
			$group = $actgroup;
			$first = 0;
		} else {
			$out .= qq{<option value="$i">$descr + $preis</option>\n};
		}
	}
	$out .= qq{</select><p>\n<br>
```

Copyright 2017 by IBO Internet Software

```
      <input name="preis" style="size=20px;border=0;padding=0" size="7"
value="$preis">€ __id__\n</form>\n};
      main::mprint($out);
}
-->
<!--spmacro:pagetype(1)-->
                                                <table width="100%">
                                                      <tr>
                                                            <td class="f_normal">
                                                            __conf::list__
                                                       </td>
                                                      </tr>
                                                </table>
```

3.7.12 Anzahl der auszugebenden Artikeln pro Seite

Damit der Besucher im Shop die Anzahl der ihm angezeigten Artikeln pro Seite ändern kann, muss er entweder einen Link aufrufen oder ein Formular absenden. Dabei muss dem Shop dann der Parameter "itemsperpage" mit einem gültigen Wert übergeben werden. Die so übergebene Anzahl wird in den Warenkorb des Benutzers eingetragen und wird danach bei allen Artikelauflistungen per loopitem berücksichtigt.

Lediglich in 2 Fällen wird die Auswahl des Benutzers ignoriert:

1. Alle Auflistungen der im Warenkorb befindlichen Artikel
2. Auflistungen, bei denen loopitem fester Anzahl der Wiederholungen angegeben ist (zb <!--spmacro:loopitem5-->)

Beispiele für die Ausgabe einer Auswahlmöglichkeit für den Benutzer

Einfache Links

```
<a href="http://__xxpath__?show__xxpgid__,__xxsession__,,,itemsperpage=10">10</a>
<a href="http://__xxpath__?show__xxpgid__,__xxsession__,,,itemsperpage=50">50</a>
<a href="http://__xxpath__?show__xxpgid__,__xxsession__,,,itemsperpage=100">100</a>
```

Dynamisch erzeugte Links mit Hervorhebung der bereits gewählten Anzahl

Artikel pro Seite (per Link):1 **10** 50 100 1000

```
<!--spmacro:module(artikelproseite)
sub itemsperpage_links {
    my $bisher = ssp::getItemsPerPage();
    my $out = "Artikel pro Seite (per Link):";
    my $modus = "items"; ## Artikel pro Seite angeben, nicht Zeilen
                         ## "rows" würde den Zeilenmodus aktivieren.

    foreach my $auswahl (ssp::getItemsPerPageOptions()) {
        if ($auswahl != $bisher) {
            $out .= qq|<a
href="http://__xxpath__?show__xxpgid__,__xxsession__,,,itemsperpage=$auswahl&amp;mo
de=$modus">$auswahl</a> |;
        } else {
            $out .= qq|<b>$auswahl</b> |;
        }
    }

    main::mprint($out);
}
-->
```

Aufruf der Funktion per __artikelproseite::itemsperpage_links__.

Dynamisch erzeugte Auswahlliste mit Vorselektierung der bereits gewählten Anzahl.

Artikel pro Seite (per Auswahlbox): 10

1
10
50
100
1000

```
<!--spmacro:module(artikelproseite)
sub itemsperpage_select {
    my $bisher = ssp::getItemsPerPage();
    my $modus = "items"; ## Artikel pro Seite angeben, nicht Zeilen
                         ## "rows" würde den Zeilenmodus aktivieren.

    my $out = qq|
```

Copyright 2017 by IBO Internet Software

```
        <form method="post" id="itemsperpage"
action="http://__xxpath__?show__xxpgid__,__xxsession__">
            <input type="hidden" name="mode" value="items">
            Artikel pro Seite (per Auswahlbox):
            <select name="itemsperpage"
onchange="document.getElementById('itemsperpage').submit();">|;

    foreach my $auswahl (ssp::getItemsPerPageOptions()) {
        $out .= qq|<option value="$auswahl" |.($auswahl == $bisher ? 'selected' :
'').qq|>$auswahl</option>|;
    }

    $out .= qq|</select><noscript><input type="submit"
value="ok"></noscript></form>|;

    main::mprint($out);
}
-->
```

Aufruf der Funktion per __artikelproseite::itemsperpage_select__.

3.7.13 Artikel sortieren

Sollen Artikel bei der Ausgabe auf einer Artikelseite nach einem bestimmten Feld sortiert werden, kann man hierzu die Callback-Funktion AddSQLCondition benutzen. Man fügt mit Hilfe dieser Funktion einen zusätzlichen ORDER BY Befehl in das SQL-Statement zum Auslesen der Artikel ein. Wichtig ist, dass AddSQLCondition im Modul Callback in stdall.ipl hinterlegt wird, damit der Callback vor dem Einlesen der Artikel wirksam werden kann.

```
<!—spmacro:module(Callback)
sub AddSQLCondition {
      my $select = $_[0];
      my $pre  = ssp::db_get_prefix();
      my $sqlstr;
      my ($sfield,$sdirection);

      if (defined $main::FORM{'sfield'}) {
            $sfield = $main::FORM{'sfield'};
            if (defined $main::FORM{'sdirection'}) {
                  $sdirection = $main::FORM{'sdirection'};
            }
      }

      if ($select == 4) { ## nur für Artikelseiten
            if ($sfield == 1) { # Preis
                  if ($sdirection == 1) {
                        $sqlstr = " ORDER BY ($pre\ITEM.PRICE + 0.0) ASC ";
                  } else {
                        $sqlstr = " ORDER BY ($pre\ITEM.PRICE + 0.0) DESC ";
                  }
            }
            if ($sfield == 2) { # Artikelnummer
                  if ($sdirection == 1) {
                              $sqlstr = " ORDER BY ($pre\ITEM.ITEMID) ASC ";
                  } else {
                              $sqlstr = " ORDER BY ($pre\ITEM.ITEMID) DESC ";
                  }
            }
      }
      return $sqlstr;
}
-->
```

Die Umschaltung der Sortierrichtung und die Auswahl des Sortierfeldes erfolgt über die Formularvariablen (sdirection und sfield). Diese werden per GET als Parameter (nach dem 4. Komma) übergeben, damit wird ein komplettes Form-Element überflüssig.

Copyright 2017 by IBO Internet Software

Für die optische Darstellung der Sortierrichtung und für die Bereitstellung der Links, wird noch ein Modul sort in die Produktseite eingebunden. Mit Hilfe der CSS-styles sort_asc, sort_desc, sort_inactive werden die verschiedenen Zustände der Sortierelemente dargestellt.

```
<!--spmacro:module(sort)

# sfield = 1 Price
sub price {
	my $s = ssp::get_var_form("sdirection");
	my $f = ssp::get_var_form("sfield");
	if ($f == 1) {
		if ($s == 1) {
			$out = _renderlink("Preis",1,2,"sort_asc");
		}
		if ($s == 2) {
			$out = _renderlink("Preis",1,1,"sort_desc");
		}
	} else {
			$out = _renderlink("Preis",1,2,"sort_inactive");
	}
	main::mprint($out);
}

# sfield = 2 Itemid (Artikelnummer)
sub itemid {
	my $s = ssp::get_var_form("sdirection");
	my $f = ssp::get_var_form("sfield");
	if ($f == 2) {
		if ($s == 1) {
			$out = _renderlink("Art.Nr",2,2,"sort_asc");
		}
		if ($s == 2) {
			$out = _renderlink("Art.Nr",2,1,"sort_desc");
		}
	} else {
			$out = _renderlink("Art.Nr",2,2,"sort_inactive");
	}
	main::mprint($out); }


# sfield = 3 Description (Artikelbeschreibung)
sub descr {
[...]
}

sub _renderlink  {
		$title = $_[0];
		$field = $_[1];
		$sdir  = $_[2];
		$style = $_[3];
		return qq{<div class="sorter"><a id="$style"
href="__xxpath__?show$ssp::pgid,__xxsession__,,,sfield=$field&sdirection=$sdir">$ti
tle</a></div>};
}
-->
```

Die Funktion wird dann per __sort::price__ oder __sort::itemid__ eingebunden. Im Prinzip kann nach jedem Feld sortiert werden, dass sich in der Tabelle ITEM befindet. Man muss nur den richtigen Feldnamen.

3.8 Datenbankzugriff

Durch Datenbankzugriff über SSP ist es möglich, beliebige Informationen aus der Shop Datenbank auszulesen und flexibel darzustellen. Auch die Veränderung von Daten in Tabellen ist möglich. Damit die Veränderung auch Offline in ShopPilot Work Center registriert wird, wird die Veränderung über die Funktion "Online Daten einlesen" auch lokal abgebildet.

Copyright 2017 by IBO Internet Software

3.8.1 Grundsätzliche Vorgehensweise

Eine Seite mit freiem Inhalt beruht wie alle anderen Seiten auf einer HTML-Vorlage.
Normalerweise wird der Dateninhalt der Seite aus dem Katalog gelesen. Der Katalog enthält die Informationen über Artikel und Seiten. Beim Aufruf einer Seite ist durch das Kommando, welches iboshop als Parameter übergeben wird klar, welche Daten dargestellt werden sollen.

Anders beim direkten Datenbankzugriff, statt der Ermittelung der Daten über das ausgeführte Kommando wird hier der Dateninhalt über eine Funktion im Template bereitgestellt. Diese Funktion ist eine sog. Callback-Funktion, da sie aus dem Kern des Scripts aufgerufen wird. Dies Funktion hat den Namen **begin_read_template()** im und muss im Modul **Callback** definiert sein.

Falls Callback::begin_read_template() in einer HTML-Vorlage definiert ist, wird es vor der Verarbeitung des BODY –Abschnitt des Templates ausgeführt. Dazu muss begin_read_template() im HEAD-Abschnitt des Templates definiert sein. Die Ausführung von begin_read_template() und Rückgabe der Anzahl eingelesener Objekte, ist vor der Verarbeitung des Body-Anschnitts notwendig, um festzustellen, wie viele Objekte eingelesen wurden, was bestimmt wie oft loopitem ausgeführt werden soll.

Administrative Funktionen können nur von Benutzern mit dem Benutzertyp "Admin"
verwendet werden. Ansonsten sorgen Sicherheitsvorkehrungen dafür, dass Veränderungen nicht vollständig ausgeführt werden.

Für die Nutzung ist außerdem das Modul ShopPilot Visual Forms notwendig.

3.8.1.1 Anwendung von begin_read_template

```
<!--spmacro:module(Callback)

sub begin_read_template {
    my $kdnr = ssp::get_var_user("kundennr");

    my $pre  = ssp::db_get_prefix();
    my $col = $pre . "USR.KUNDENNR";
    my $k = ssp::db_query_orders("$col = \'$kdnr\'");
    return $k;
}
-->
```

In diesem Beispiel werden die Bestellungen eines Kunden ausgelesen, wie es z.B. für eine Seite notwendig ist, die die Bestellungen des eingeloggten Kunden auflistet. Wichtig ist, dass die Anzahl ausgelesener Datensätze von `begin_read_template` zurückgegeben werden (hier $k).

Das Argument von `db_query_orders` ist hier z.B. `"A01_USR.KUNDENNR = '4711'".` Der String A01_ ist der Tabellenprefix, der mit der Funktion `db_get_prefix()` ermittelt wird. Ein Tabellenprefix ist nur dann vorhanden, wenn mehrere Shops in einer DB installiert sind.

3.8.1.2 Daten einlesen

Zum Einlesen der Daten gibt es einige vorbereitete Abfragen. Diese Abfragen beziehen sich auf eine oder mehrere Tabellen, die entsprechend verknüpft sind.

Vorbereitete Abfragen sind:

Tabellenname	Objektyp	Funktion
ITEM	Artikel	**db_query_item(int <subcatid>, text <wherebedingung>)**
ORDERS	Bestellungen	**db_query_orders (text <wherebedingung>)**
OITEMS	Bestellpositionen	**db_query_oitems (text <wherebedingung>)**
PAGE	Seiten	**db_query_page (text <wherebedingung>)**
USR	Kunden	**db_query_usr (text <wherebedingung>)**

Die Abfragen erhalten als optionales Argument eine Where-Bedingung. Die Where-Bedingung wird mit der in der Abfrage intern schon vorhandenen Where-Bedingung per AND verknüpft.
Sollen mehr als eine Bedingung notwendig sein kann man diese im SQL Dialekt der verwendeten Datenbank formulieren.
Um die Eindeutigkeit von Spaltennamen in der Where-Bedingungen zu erreichen, muss das Datenfeld mit dem Tabellennamen qualifiziert werden. Der Tabellenname ist identisch mit dem für diesen Objekttyp zuständigen Tabelle. Um sicherzustellen, dass die Tabellennamen auch richtig sind, wenn ein Tabellenprefix benutzt wird, ermittelt die Funktion ssp::db_get_prefix() den evtl. benutzten Prefix.

Copyright 2017 by IBO Internet Software

Beispiel 1:
```
my $pre  = ssp::db_get_prefix();
my $tab = $pre . "OITEMS";
db_query_oitems("$tab.ORDERNR = 4711")
Liest die Bestellpositionen der Bestellung mit der Nummer 4711 ein.
```

Beispiel 2:
```
my $pre  = ssp::db_get_prefix();
my $tab = $pre . "OITEMS";
db_query_oitems("$tab.ORDERNR = 4711 AND $tab.ITEMID = \'artnr_1\'")
Liest die Bestellpositionen der Bestellung mit der Nummer 4711 ein, für welche die
Artikelnummer "artnr_1" ist.
```

3.8.1.3 Daten anzeigen

Um die Daten einer ausgeführten Abfrage zu erhalten, gibt es die Funktion

<wert> ssp::get_var_db (text <name>,int <index>)

Diese funktioniert genauso, wie z.b. get_var_cart(), name ist also der Name des Datenfeldes und index die entsprechende Position in der Aufzählung.

Die Namen der Datenfelder sind hier durch die Abfragen definiert. Sie entsprechen den Spaltennamen in den jeweiligen Tabellen. Zusätzliche Felder erhalten eigene Namen.

Die Darstellung im Template erfolgt dann analog zu anderen Variablen über kleine SSP-Funktionen in einem Modul.

```
<!—spmacro:module(u)
sub firma {
       my $index = $_[0];
       my $firma = ssp::get_var_db("FIRMA",$index);
       main::mprint("$firma");
}
-->
```

3.8.1.4 Daten ändern

Sollen Daten verändert, werden dies per Formular mit der Methode POST übergeben. Durch ein Formularvariable mit dem Namen "mode" wird die durchzuführende Aktion gekennzeichnet. In der Funktion begin_read_template wird dann die entsprechende Update-Funktion aufgerufen. Zusätzlich ist notwendig, dass die Primärschlüssel der entsprechenden Tabelle in einem Formular-Übergabefeld definiert sind. Die Konvention ist, dass beim Primärschlüssel der Feldname mit "_PK" erweitert wird, also z.b. ITEMID_PK für die Artikelnummer.

Vorbereitete Abfragen sind:

db_update_item(int <subcatid> text, <artikelnr>, int <seitennr>,
text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)
db_update_orders(text <bestellnr>,
text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)
db_update_page(int <seitennr>,
text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)
db_update_usr(text <kundennr>,
text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)

Die Veränderung der Daten wird bei Bestellungen mit der Funktion db_update_orders durchgeführt.

Beispiel:
```
db_update_orders($ordernr,"STATUS",$status,"SHIPPINGINFO",$shippinginfo)
```

Das erste Argument gibt den Wert für den Primärschlüssel an. Dann folgen die zu verändernden Werte als Wertepaar mit Spaltennamen und Wert.

Copyright 2017 by IBO Internet Software

3.8.1.5 Daten einfügen

Zum Einfügen neuer Artikel (und Subkatalog-Artikel) werden folgende Funktionen bereitgestellt:

db_insert_item(int <subcatid>,text <artikelnr>, int <seitennr>,
text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)

Mittels db_insert_item wird ein Artikel eingefügt und für den mit subcatid angegebenen Subkatalog sichtbar geschaltet. Wird für subcatid 0 angegeben, wird nur der Hauptkatalog berücksichtigt. Die Primärschlüssel artikelnr, seitennr, subcatid müssen angegeben werden.

Beispiel:

```
my $seite           = 10;
my $artikelnummer   = "artikelnr_1";
my $beschreibung    = "Ich bin ein Artikel";

# Einfügen des Artikels und vorbelegen der Beschreibung
db_insert_item(0,$artikelnr, $seite, "DESCR", $beschreibung);
```

3.8.1.6 Daten löschen

int status = db_delete_item(int <subcatid>,text <artikelnr>, int <seitennr>)
int status = db_delete_iscat (int <subcatid>, text <artikelnr>, int <seitennr>)
int status = db_delete_orders(int <bestellnr>)

Beispiel:

```
my $seite           = 10;
my $artikelnummer   = "artikelnr_1";
# Löschen des Artikels
db_delete_item(0,$artikelnr, $seite);
```

3.8.2 Funktionen für Shopobjekte

3.8.2.1 Artikel

3.8.2.1.1 db_query_item

int anzahl = db_query_item(int <subcatid>, text <wherebedingung>)

Durch den Wert **subcatid** kann ein Subkatalog ausgewählt werden. Hat **subcatid** den Wert 0, wird der Hauptkatalog gewählt.

Diese Funktion liest aus der Datenbank alle Artikel aus. die der Where-Bedingung genügen.
Die Rückgabewert ist die Anzahl der ausgelesenen Artikel. Die Where-Bedingung ist Optional. Falls kein Subkatalog existiert, sind die Datenfelder für den Subkatalog leer.
Falls ein Subkatalog existiert und MODI = 1 werden die Werte für Preise automatisch durch die entsprechenden Angaben im Subkatalog überschrieben.

Folgende Felder können nach dieser Abfrage mithilfe von **get_var_db** ausgelesen werden.

Typ	***Name***	***Bedeutung***
int	**SID**	Positionsnummer auf der Seite
int	**PGID**	Seitennummer
text	**ITEMID**	Artikelnummer
text	**DESCR**	Beschreibung
text	**PRICE****	Preis
text	**IMG**	Artikelbild
text	**ZTEXT**	Zusatztext
text	**UST**	Umsatzsteuer-ID
text	**ERG1 – ERG4**	Ergänzungsfelder 1-4
text	**LIST1 – LIST2**	Listen 1 und 2
text	**TEMPL**	Vorlagendatei
text	**DIMG**	Detail-Bild
text	**DETAIL**	Detail-Beschreibung

Copyright 2017 by IBO Internet Software

text	**STAFM1 – STAFM4****	Staffelmengen 1-4
text	**STAFP1 – STAFP4****	Staffelpreise 1-4
text	**„Bezeichnung benutzerdef. Feld“**	Benutzerdef. Felder Hauptartikel
int	**SCID ***	ID des zugeh. Subkatalogs
int	**MODI ***	Wenn Originalartikel benutzt wird 0, sonst 1
text	**BESTNR ***	Bestellnr
text	**„Bezeichnung benutzerdef. Feld“ ***	Benutzerdefinierte Felder Subkatalog

*nur bei Subkatalog
** bei MODI=1 durch Subkatalogwerte überschrieben

Eine Besonderheit weisen die Benutzerdefinierten Felder für die Sprache auf. Diese überschreiben entsprechende Hauptkatalogfelder immer (unabhängig von MODI) , wenn sie nicht leer sind.

3.8.2.1.2 db_update_item

`int status = db_update_item`(int <subcatid>, text <artikelnr>, int <seitennr>,, text <name1>, text/int <wert1>, text <name2>, text/int <wert2>, ...)

Der Parameter **subcatid,** ist derzeit ohne Bedeutung. Aus Gründen der Aufwärtskompatibilität sollte der Wert 0 übergeben werden.

Diese Funktion überschreibt angegebene Werte aller Artikel mit der Artikelnummer **artikelnr** und zugehöriger Seite **seitennr**. Ist seitennr leer, werden auch evtl. vorhandene Verknüpfungen berücksichtigt.

Die nachfolgenden Parameter name*X* und wert*X* geben die zu ändernden Felder und die neuen Werte an. Die Felder können alle diejenigen sein, die mithilfe von db_query_item ausgelesen werden können.

Der Rückgabestatus zeigt bei einem Wert ungleich Null einen Fehler an.

3.8.2.1.3 db_insert_item

int status = db_insert_item(int <subcatid>, text <artikelnr>, int <seitennr>, text <name1>, text/int <wert1>, text <name2>, text/int <wert2>, ...)

Diese Funktion erstellt einen neuen Haupt-Artikel.

Der Parameter **subcatid,** ist derzeit ohne Bedeutung. Aus Gründen Aufwärtskompatibilität sollte der Wert 0 übergeben werden.

Die Parameter artikelnr und seitennr müssen angegeben werden, da sonst der Artikel nicht eindeutig zugeordnet werden kann!
Die Subcatid unterscheidet die verschiedenen Subkataloge und ist ebenfalls ein Mussparameter.
Es wird ein Artikel eingefügt und für den mit subcatid angegebenen Subkatalog sichtbar geschaltet. Wird für subcatid -1 angegeben, wird nur der Hauptkatalog berücksichtigt.

Die nachfolgenden Parameter name*X* und wert*X* geben die zu ändernden Felder und die neuen Werte an. Die Felder können alle diejenigen sein, die mithilfe von db_query_item ausgelesen werden können.

Der Rückgabestatus zeigt bei einem Wert ungleich Null einen Fehler an. Ein häufige Fehlerursache ist bei Insert eine Schlüsselverletzung, die anzeigt, dass ein Datensatz mit identischen Schlüsselwerten schon in der Datenbank vorhanden ist und deshalb die Operation fehlgeschlagen ist. Ein entsprechender Fehlerhinweis sollte sich in der Logdatei finden.

3.8.2.1.4 db_delete_item

int status = db_delete_item(text <artikelnr>, int <seitennr>)

Diese Funktion löscht einen oder mehrere durch die Primärschlüssel festgelegte Artikel.
Wenn die Seitennummer leer ist, werden auch Verknüpfungen gelöscht.
Das Löschen eines Artikels bewirkt implizit auch, dass der Artikel in allen Subkatalogen gelöscht wird.

Beispielaufruf:

```
int status = db_delete_item("artikelnr_1", 10);      # im Hauptkat. und allen
# Subkatalogen löschen
```

3.8.2.2 Subkatalog Artikel

Die Bedeutung der Parameter und Rückgabewerte entspricht den Angaben bei den entsprechenden Funktionen für Artikel. Der Parameter subcatid muss hier einem gültigen Subkatalog entsprechen. Sollen Einträge in mehrere Subkataloge durchgeführt werden, so sind die Funktionen entsprechend mehrfach mit unterschiedlichen subcatid anzuwenden.

Copyright 2017 by IBO Internet Software

3.8.2.2.1 db_query_iscat
int anzahl = db_query_iscat (text <wherebedingung>)

3.8.2.2.2 db_update_iscat
int status = db_update_iscat (int <subcatid>, text <artikelnr>, int <seitennr>,
text <name1>, text/int <wert1>, text <name2>, text/int <wert2>, ...)

3.8.2.2.3 db_insert_iscat
int status = db_insert_iscat (int <subcatid>, text <artikelnr>, int <seitennr>,
text <name1>, text/int <wert1>, text <name2>, text/int <wert2>, ...)

3.8.2.2.4 db_delete_iscat
int status = db_delete_iscat (int <subcatid>, text <artikelnr>, int <seitennr>)

Diese Funktion löscht den Eintrag für den entsprechenden Subkatalog. Die bedeutet, der Artikel ist in diesem Subkatalog nicht mehr sichtbar. Der Hauptkatalog und andere Subkataloge bleiben hiervon unberührt.

3.8.2.3 Seiten

3.8.2.3.1 db_query_page

int anzahl = db_query_page(text <wherebedingung>)

Diese Funktion liest aus der Datenbank alle Seiten aus.
Die Rückgabewert ist die Anzahl der ausgelesenen Seiten.
Die Where-Bedingung ist Optional.

Folgende Felder können nach dieser Abfrage mithilfe von **get_var_db** ausgelesen werden.

Typ	*Name*	*Bedeutung*
int	**SID**	Positionsnummer
int	**PGID**	Seitennummer
int	**PPGID**	Seitennr. der übergeordneten Seite
text	**DESCR**	Beschreibung
text	**TITLE**	Seitentitel
text	**ICON**	Icon
text	**IMG**	Bild
text	**TEMPL**	Vorlagendatei

3.8.2.3.2 db_update_page

int status = db_update_page(int <seitennr>, text <feld1>, text/int <wert1>, text <feld2>,
text/int <wert2>, ...)

Diese Funktion überschreibt angegebene Werte der Seite mit der Seitennummer seitennr.
Der Parameter seitennr muss übergegeben werden.
Die nachfolgenden Parameter feld*X* und wert*X* geben die zu ändernden Felder und die neuen Werte an.
Die Felder können alle diejenigen sein, die mithilfe von db_query_page ausgelesen werden können.

3.8.2.4 Bestellungen

3.8.2.4.1 db_query_orders

`int anzahl = db_query_orders`(text <wherebedingung>)

Diese Funktion liest aus der Datenbank alle Bestellungen aus, hinzu kommen einige Zusatzinformationen zu den Bestellpositionen und den Kunden.
Die Rückgabewert ist die Anzahl der ausgelesenen Bestellungen.
Die Where-Bedingung ist Optional.

Folgende Felder können nach dieser Abfrage mithilfe von **get_var_db** ausgelesen werden.

Typ	*Name*	*Bedeutung*
int	**ORDERNR**	Bestellnr
int	**PRICE**	Gesamtpreis (o. Versand)
int	**POSITIONS**	Anzahl Bestellpositionen
int	**OTIME**	Bestellzeitpunkt

Copyright 2017 by IBO Internet Software

int	**STATUS**	Status der Bestellung
text	**KUNDENNR**	Kundennr.
text	**NAME**	Kundenname
text	**FIRMA**	Kundenfirma
text	**SHIPPINGINFO**	Versandinformation
text	**SESSIONID**	SessionID
int	**SCID**	Subkatalog-ID
text	**MERCHID**	Händler-ID

3.8.2.4.2 db_update_orders

int status = db_update_orders(int <bestellnr>, text <feld1>, text/int <wert1>, text <feld2>, text/int <wert2>, ...)

Diese Funktion überschreibt angegebene Werte der Bestellung mit der Bestellnummer bestellnr.
Der Parameter bestellnr muss übergegeben werden.
Die nachfolgenden Parameter feld*X* und wert*X* geben die zu ändernden Felder und die neuen Werte an.
Die Felder können folgende sein:

Typ	***Name***	***Bedeutung***
text	**ORDERNR**	Bestellnummer
int	**OTIME**	Bestellzeitpunkt
int	**STATUS**	Status der Bestellung
text	**KUNDENNR**	Kundennummer
text	**SHIPPINGINFO**	Versandinformation

3.8.2.4.3 db_delete_orders

int status = db_delete_orders(int <bestellnr>)

Löscht Bestellung und Bestellpositionen zu einer angegebenen Bestellnummer.

3.8.2.5 Bestellpositionen

3.8.2.5.1 db_query_oitems

int anzahl = db_query_oitems(text <wherebedingung>)

Diese Funktion liest aus der Datenbank alle Bestellpositionen aus. Hinzu kommen einige Zusatzinformationen zu der Bestellung, des Kunden und den Artikeln.
Die Rückgabewert ist die Anzahl der ausgelesenen Bestellpositionen
Die Where-Bedingung ist optional.

Folgende Felder können nach dieser Abfrage mithilfe von **get_var_db** ausgelesen werden.

Typ	***Name***	***Bedeutung***
int	**ORDERNR**	Bestellnr
int	**OTIME**	Bestellzeitpunkt
int	**ORDERSTATUS**	Status der Bestellung
text	**SESSIONID**	Sessionid der Bestellung
text	**ITEMID**	Artikelnr
text	**DESCR**	Beschreibung
int	**COUNT**	Anzahl Bestellpositionen
int	**PRICE**	Einzelpreis
int	**STATUS**	Status der Bestellposition
text	**KUNDENNR**	Kundennr.
text	**NAME**	Kundenname
int	**ITEMLIST1**	Liste 1
int	**ITEMLIST2**	Liste 2

3.8.2.6 Kunden/Benutzer

3.8.2.6.1 db_query_usr

int anzahl = db_query_usr(text <wherebedingung>)

Diese Funktion liest aus der Datenbank alle Kunden aus.
Die Rückgabewert ist die Anzahl der ausgelesenen Kunden.

Copyright 2017 by IBO Internet Software

Die Where-Bedingung ist Optional.
Folgende Felder können nach dieser Abfrage mithilfe von **get_var_db** ausgelesen werden.

Typ	*Name*	*Bedeutung*
Text	**HAENDLERID**	HändlerID
text	**KUNDENNR**	Kundennummer
text	**LOGIN**	Login-Name
text	**PASSWORD**	Login-Passwort
text	**ANMERK**	Anmerkung
text	**ANREDE**	Anrede
text	**VORNAME**	Vorname
text	**NAME**	Nachname
text	**FIRMA**	Firma
text	**ORT**	Ort
text	**PLZ**	Postleitzahl
text	**STRASSE**	Strasse
text	**EMAIL**	eMail-Adresse
text	**TEL**	Telefon
text	**FAX**	Fax
text	**LKZ**	Landeskennziffer
text	**RABATT**	Rabatt
text	**PFAKTOR**	Preisfaktor
int	**MODITIME**	Zeitstempel der letzten Kundenänderung
int	**SUBC**	Subkatalog des Kunden
int	**CTYPE**	Kundentyp
text	**„Bezeichnung benutzerdef. Feld“**	Benutzerdefinierte Felder

3.8.2.6.2 db_update_usr

**db_update_usr(text <kundennr>, text <feld1>, text/int <wert1>, text <feld2>,
text/int <wert2>, ...)**

Diese Funktion überschreibt angegebene Werte des Kunden mit der Kundennummer kundennr.
Der Parameter kundennr muss übergegeben werden. Die nachfolgenden Parameter feld*X* und wert*X* geben die zu ändernden Felder und die neuen Werte an. Die Felder können alle diejenigen sein, die mithilfe von db_query_usr ausgelesen werden können.

3.8.2.6.3 db_delete_usr

int status = db_delete_usr(text <kundennr>)

Löscht einen Kunden mit der angegebenen Kundennummer..

3.8.3 Freie Datenbankabfragen mit SQL

Neben dem Zugriff auf die bekannten Objekte mit den vordefinierten Funktionen ist auch der komfortable Zugriff auf beliebige Tabellen mit Hilfe der Funktion readSQLData() möglich. Das Ergebnis der Abfrage kann man mit der Funktion get_var_db() ausgelesen werden.

3.8.3.1 readSQLData

int anzahl readSQLData(text <SQL-Select-Statement>,@text <Variablenname>)
Diese Funktion führt das angegebene SQL-Select-Statement aus.
Die Liste der Variablen korrespondiert über die Reihenfolge mit der Liste der Spaltenbezeichnungen im Select-Statement. Die erste angegebene Spalte wird also in der ersten Variablen gespeichert. Die zweite Spalte in der zweiten Variablen usw..
Die Namen der Variablen können von den Namen der Spalten abweichen, obwohl es sicherlich sinnvoll ist, diese in den meisten Fällen gleich zu halten.
Die Rückgabewert ist die Anzahl der ausgelesenen Zeilen. Wird -1 zurückgegeben deutet dies auf einen Fehler. Der Fehlertext kann mit ssp::get_last_sqlerr() ermittelt werden.

Die Daten werden mit der Funktion get_var_db() und dem Namen der jeweiligen Variablen ausgelesen.

Beispiel:

Copyright 2017 by IBO Internet Software

```
my $count = ssp::readSQLData("SELECT VORNAME, NAME FROM USR WHERE
NAME='Meyer'","VORNAME","NAME");
if ($count >= 0) {
       my $i=0;
       while ($count > $i) {
          $name = ssp::get_var_db("NAME",$i);
          $vorname = ssp::get_var_db("VORNAME",$i);
          main::mprint("Familie Meyer: $vorname $name<br>\n");
          ++$i;
       }
} else {
my $errtext = ssp::get_last_sqlerr();
       main::mprint("Fehler beim DB Zugriff: $errtext<br>\n");
}
```

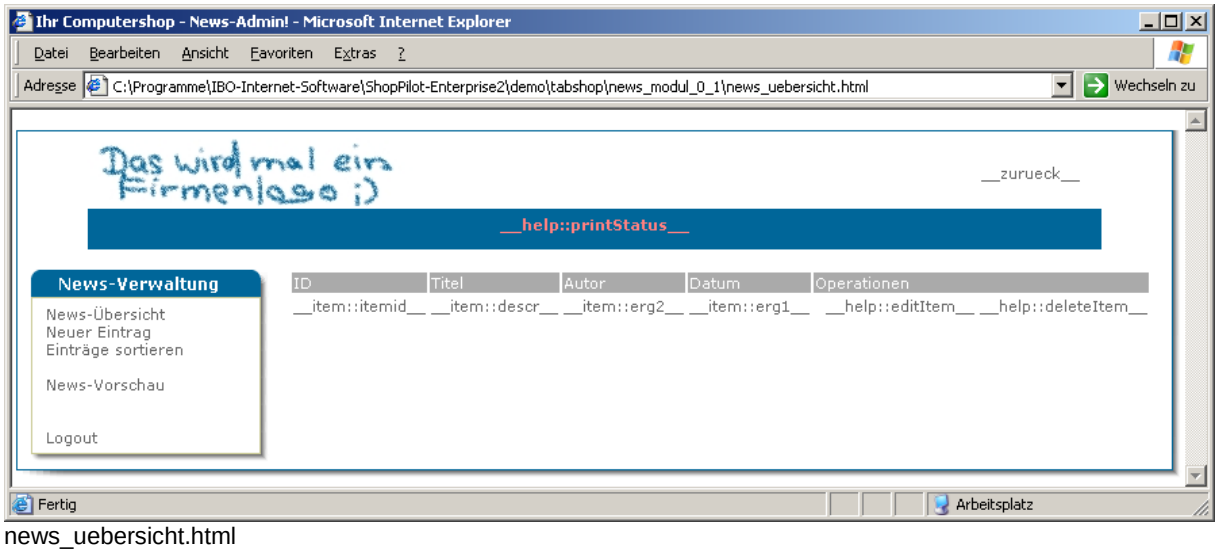
3.8.3.2 get_last_sqlerr

text get_last_sqlerr()
Gibt den Fehlertext der letzten SQL-Abfrage zurück. Die Anwendung ist für alle SQL-Abfragen möglich (also nicht nur für readSQLData)

3.8.4 Anwendungsregeln anhand eines Beispiels

Anhand eines Beispiel wird erläutert, wie man eine Anwendung in einen Shop integriert die es ermöglicht Eigenschaften eines Artikels Online zu ändern, neue Artikel anzulegen, die Reihenfolge von Artikeln zu ändern und Artikel zu löschen. Diese Anwendung soll den Namen "online_editor" erhalten, da sie z.B. dazu verwendet werden kann, externe Redakteure per Webbrowser mit der Pflege von Inhalten zu betrauen. Gleichzeitig sollen hierbei auch einige Konventionen festgehalten werden, die den Umgang mit dem Framework erleichtern sollen.

3.8.4.1 Übersicht

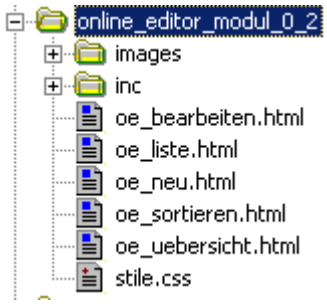


news_uebersicht.html

Für dieses Projekt werden 5 Vorlagen benötigt für die aktiven Funktionen gibt es entsprechende ipl-Dateien.

oe_uebersicht.html	oe_index.ipl	Auflistung der News
oe_bearbeiten.html	oe_edit.ipl	Editieren
oe_neu.html	oe_new.ipl	Neuer Eintrag
oe_sortieren.html	oe_sort.ipl	Sortieren
oe_liste.html		Vorschau

Copyright 2017 by IBO Internet Software



Weiterhin: oe_settings.ipl für Standardeinstellungen und oe_idenitfier.ipl zur Realisierung der notwendigen Platzhalter.

Voraussetzung für die zugangsgeschützte Funktion ist ein eigener Subkatalog, in dem die Verwaltungsseiten aktiviert werden. Der Shop sollte darüber hinaus nicht über den Hauptkatalog aufrufbar sein (Subkatalog für Standardhändler einstellen!). In allen anderen Subkatalogen sollten diese Seiten deaktiviert sein. Zugang erhalten ausschließlich Benutzer vom Typ "Administrator".
Seitenaufruf mit showbyname: Statt eine Seite über eine Seitennummer aufzurufen, wird hier ein anderes Verfahren benutzt, der Aufruf per Name. Das Kommando showbyname ermittelte eine Seite anhand des Titels, statt wie sonst üblich anhand der Seitennummer. Dies ermöglicht eine bessere Modularität.

Beispielaufruf:
`__xxpath__?showbynameNews_Uebersicht,__xxsession__`

3.8.4.2 News-Übersicht

Datei: news_uebersicht.html

```
<html>
<head>
<title>Ihr Computershop - News-Admin!</title>
<!--spmacro:include(online_editor_modul_0_2/inc/news.ipl)-->
</head>
<body> .... </body>
</html>
```

Die Datei unterscheidet sich lediglich durch das Include von news.ipl von anderen ShopPilot Vorlagen. Die benutzten Identifier werden durch das Modul help und item bereitgestellt.

Datei: news.ipl

```
<!--spmacro:include(online_editor_modul_0_2/inc/oe_settings.ipl)-->
<!--spmacro:module(Callback)
sub begin_read_template {
       $errmsg =  "";

       # erwartet werden die folgenden Felder: ITEMID, mode
       my $mode            = ssp::get_var_form("mode");
       my $itemid          = ssp::get_var_form("ITEMID");
       ## Voreinstellung
       my $pgid = $oe_settings::newspgid;

       if ($mode eq "delete") {
              if (length($itemid) && length($pgid)) { ## Sicherheitsabfrage
                     my $ok = ssp::db_delete_item(0, $itemid, $pgid);
                     if (!$ok) {
$errmsg = "Artikel $itemid wurde gelöscht";
} else {
$errmsg = "Artikel $itemid konnte nicht gelöscht werden";
}
}
       }

       ## Liste auslesen
       my $query;
       $where .= "PGID = $pgid";
       my $count = ssp::db_query_item(0, $where);
       return $count;
}
-->

<!--spmacro:module(help)
sub printStatus { ## __help::printStatus__
```

Copyright 2017 by IBO Internet Software

```
	main::mprint($Callback::errmsg);
}

sub editItem { ## __help::editItem__
	my $index = $_[0];
	my $itemid = ssp::get_var_db("ITEMID", $index);
	my $pgid = ssp::get_var_db("PGID", $index);
	my $url =
"__xxpath__?showbynameNews_Bearbeiten,__xxsession__,,,ITEMID=$itemid";
	my $str = qq{
		<a href='$url'><img src="images/news_bearbeiten.gif" border="0"></a>
	};
	main::mprint($str);
}

sub deleteItem { ## __help::deleteItem__
	my $index = $_[0];
	my $itemid = ssp::get_var_db("ITEMID", $index);
	my $pgid = ssp::get_var_db("PGID", $index);
	my $url =
"__xxpath__?showbynameNews_Uebersicht,__xxsession__,,,ITEMID=$itemid&
PGID=$pgid&mode=delete";
	my $script = qq|
if (confirm("Artikel $itemid wirklich löschen?")) {
location.replace("$url");
}
|;
	my $str = qq{
		<a href='javascript:void(0);' title="Artikel löschen"
onClick='$script; return false;'>
<img src="images/news_loeschen.gif" border="0"></a>
	};
	main::mprint($str);
}
-->
<!--spmacro:include(online_editor_modul_0_2/inc/news_identifier.ipl)-->
```

News.ipl enthält zwei Module. Das Callback-Modul führt in begin_read_template die aktiven Funktionen aus. Dies sind hier das Löschen eines Items und das Auslesen der Liste der verfügbaren Items. Das Löschen wird nur durchgeführt, wenn der Parameter "mode" den Werte "delete" enthält. Das Modul help definiert die für diese Vorlage speziellen Identifier zur Statusanzeige, zum Aufruf der Edit-Seite und zum löschen eines Eintrags.
Die Parameterübergabe beim Löschen durch einen GET-Aufruf wird durch Anhängen der Parameter hinter den ShopPilot Standard-Parametern (nach dem 4. Komma) realisiert. Dies hat den Vorteil, dass in der aufgerufenen Seite keine Unterscheidung zwischen Parameterübergabe per POST oder GET notwendig ist und das Auslesen immer per get_var_form erfolgen kann.

3.8.4.3 Einen Eintrag editieren

Diese Seite ermöglicht es einen Eintrag zu editieren. Deshalb muss hier ein Update der Tabelle ITEM durchgeführt werden. Die hierfür verwendete Funktion ist unabhängig von den Feldern, die im Formular benutzt werden. Hierzu werden die möglichen Felder mit get_table_cols ermittelt und Formularfelder deren Name mit einem Spaltennamen übereinstimmt in die Update-Funktion db_update_item aufgenommen.

Datei: news_edit.ipl

```
<!--spmacro:include(online_editor_modul_0_2/inc/oe_settings.ipl)-->
<!--spmacro:module(Callback)

sub begin_read_template {
	my $prefix         = ssp::db_get_prefix();
	my $tablename      = "ITEM";
	my $table          = $prefix . $tablename;

	# erwartet werden die folgenden Felder: ITEMID, mode
	my $mode           = ssp::get_var_form("mode");
	my $itemid         = ssp::get_var_form("ITEMID");

	## Voreinstellung
	my $pgid = $oe_settings::newspgid;
```

Copyright 2017 by IBO Internet Software

```
	$errmsg = "Bereit";
	my $count = 0;
	if ($mode eq "edit") {

		# Spalten der ITEM-Tabelle
		my @cols = ssp::get_table_cols($tablename);

		## Felder finden die einem Spaltennamen entsprechen
		foreach my $col (@cols) {
			my $val = ssp::get_var_form($col);
			if ($val ne $ssp::undefined) {
				$COLS{$col} = $val;
			}
		}

		my @params = ();
		my @colvals = values %COLS;
		my @colkeys = keys %COLS;
		my $i = 0;
		for ($i = 0; $i <= $#colkeys; $i++) {
			# Parameter müssen folgendermassen übergeben werden:
			#  COL1, WERT1, COL2, WERT2, ...
			push(@params, $colkeys[$i]);
			push(@params, $colvals[$i]);
		}
		my $ok = ssp::db_update_item(0, $itemid, $pgid ,@params);
		if (!$ok) {
$errmsg = "Aktualisierung durchgeführt";
} else {
$errmsg = "Fehler bei Aktualisierung";
}
	} else {
		#Datensatz einlesen...
		my $query = "$table.ITEMID = \'$itemid\' AND $table.PGID = $pgid";
		$count = ssp::db_query_item(0, $query);
	}
	return $count;
}
-->

<!--spmacro:include(online_editor_modul_0_2/inc/news_identifier.ipl)-->
```

3.8.4.4 Einen neuen Eintrag anlegen

Um einen neuen sichtbaren Eintrag anzulegen, muss dieser in die Tabelle ITEM eingefügt werden. Der Primärschlüssel ITEMID wird besonders behandelt und erhält im Formular einen Namen der keine Spalte in der Tabelle ITEM ist (ITEMID_PK), da er als Parameter in der Funktion db_insert_item gebraucht wird. Als weiterer Parameter ist die vordefinierte Seitennummer anzugeben. Die übrigen Parameter werden wiederum durch Vergleich der Formularfelder mit den Spaltennamen der Tabelle ermittelt. Zusätzlich ist noch ein Eintrag in den Subkatalogen durchzuführen, in denen der neue Artikel sichtbar sein soll. Diese werden hier durch die benutzerdefinierte Variable PUBSUBCAT vorgegeben. Der Fall, dass die Artikelnummer schon vergeben ist, wird als Fehler ausgewiesen.

Datei: news_new.ipl

```
<!--spmacro:include(online_editor_modul_0_2/inc/oe_settings.ipl)-->
<!--spmacro:module(Callback)

$errmsg ="Bereit";

sub begin_read_template {
	my $prefix		= ssp::db_get_prefix();
	my $tablename		= "ITEM";
	my $table		= $prefix . $tablename;


	# erwartet werden die folgenden Felder: ITEMID, mode
	my $mode		= ssp::get_var_form("mode");
	my $itemid		= ssp::get_var_form("ITEMID_PK");
	## Voreinstellung
```

Copyright 2017 by IBO Internet Software

```
	my $pgid = $oe_settings::newspgid;

	my $count = 0;
	if ($mode eq "new") {
		# Spalten der ITEM-Tabelle
		my @cols = ssp::get_table_cols($tablename);
		## Felder finden die einem Spaltennamen entsprechen
		foreach my $col (@cols) {
			my $val = ssp::get_var_form($col);
			if ($val ne $ssp::undefined) {
				$COLS{$col} = $val;
			}
		}

		my @params = ();
		my @colvals = values %COLS;
		my @colkeys = keys %COLS;
		my $i = 0;
		for ($i = 0; $i <= $#colkeys; $i++) {
			# Parameter müssen folgendermassen übergeben werden:
			#  COL1, WERT1, COL2, WERT2, ...
			push(@params, $colkeys[$i]);
			push(@params, $colvals[$i]);
		}

		my $ok = ssp::db_insert_item(0, $itemid, $pgid, @params);
		if ($ok ) { ## Artikel existiert
			$errmsg = "Operation nicht erfolgreich: Artikelnr. existiert
bereits. Wählen Sie eine andere Artikel-Nr.";
		} else {
			$errmsg = "Erfolgreich: Artikel eingefügt.";

			## Gültige Subcats für den Nutzer auslesen
			my @scids = split(',',ssp::get_var_user("PUBSUBCAT"));

			## Eigenen Subkatalaog hinzufügen
			push(scids,ssp::get_var_subcat('mysubkat'));

			foreach my $cursc (@scids) {
				ssp::db_insert_iscat($cursc,$itemid,$pgid);
			}
		}
	}
	return $count;
}
```

3.8.4.5 Die Reihenfolge der Einträge bestimmen

Die Reihenfolge wird durch die Spalte SID in der Tabelle ITEM bestimmt.
Um die Reihenfolge zu verändern, muss ein Update für diesen Wert durchgeführt werden.
Das Verfahren beruht darauf, dass die Einträge in einer Liste mit dem über eine Listbox änderbaren Wert von SID dargestellt werden. Beim Abschicken des Formulars werden die neuen Werte übernommen und damit eine Änderung der Reihenfolge erreicht.

Datei: news_sort.ipl

```
<!--spmacro:include(online_editor_modul_0_2/inc/oe_settings.ipl)-->
<!--spmacro:module(Callback)

sub begin_read_template {

	my $prefix          = ssp::db_get_prefix();
	my $tablename       = "ITEM";
	my $table           = $prefix . $tablename;
	my $mode            =  ssp::get_var_form("mode");

	########### Verschieben per SID ######################
	if ($mode eq "sort") {
		my $itemcount = ssp::get_var_form("ITEMCOUNT");
		my $i = 0;
```

Copyright 2017 by IBO Internet Software

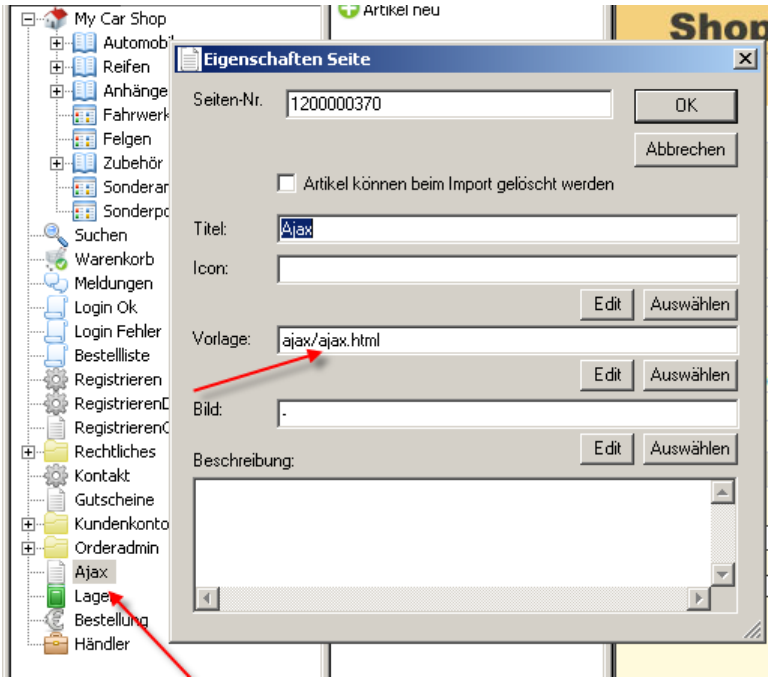
```
              for ($i = 0; $i < $itemcount; $i++) {
                     my $val      = ssp::get_var_form("SID_SELECT_$i");
                     (my $sid_new, my $itemid, my $pgid) = split("!_!", $val);
                     ### Neue SID schreiben
                     ssp::db_update_item(0, $itemid, $pgid, "SID", $sid_new);
              }
       }

       my $query;
       $query = qq{SELECT MAX($prefix\ITEM.SID) FROM $prefix\ITEM
                  WHERE $prefix\ITEM.PGID =  $news_settings::newspgid};

       ssp::readSQLData($query,"SIDMAX");

       my $count = ssp::db_query_item(0, "$prefix\ITEM.PGID =
$oe_settings::newspgid");
       return $count;
}
-->
<!--spmacro:include(online_editor_modul_0_2/inc/news_identifier.ipl)-->
```

Copyright 2017 by IBO Internet Software

3.9 Nutzung von Ajax/Json

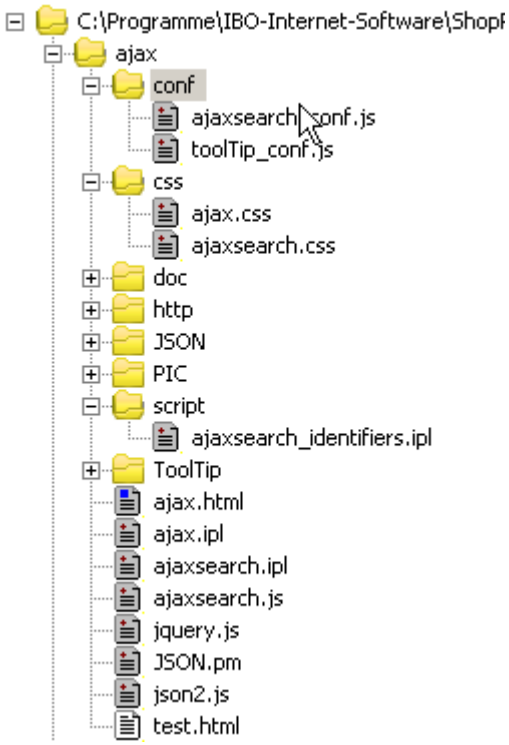
Dies ist eine Technik, die es erlaubt in einer Website Elemente zu verändern , ohne dass die komplett Seite neu geladen werden muss. Dies ermöglicht neue Effekte, die bislang nicht mit HTML-Seiten realisierbar waren. Ajax ist eine wesentliche Technologie des sog. Web2.0. Ähnlich ist JSON, der Unterschied zu Ajax ist nur, dass Objekte nicht als XML Struktur sondern als Javascript Objektdefinition übertragen werden. Json ist im allgemeinen schlanker und schneller als Ajax. In der folgenden Beschreibung wird Ajax und Json synonym benutzt. Die Übertragung der Daten erfolgt im Hintergrund durch asynchrone Serverabfragen. Meist werden die Ergebnisse durch Abfragen von auf dem Server vorhandenen Datenbanken ermittelt.

3.9.1 Aufbau



Die Interface Seite. Eine Informationsseite mit dem Template ajax.html.

Copyright 2017 by IBO Internet Software



Die Datei und Ordnerstruktur für Ajax mit der Anwendung Ajaxsearch. Alle Dateien, die zu ajaxsearch gehören haben einen entsprechenden Namen.

3.9.2 Der Kern

ajax.html	Nimmt für den Server den Request entgegen und ruft die gewünscht Funktion anhand des Methodennamen auf.
css/ajax.css	CSS Definitionen für ajax. Z. Zt. leer.
ajax.ipl	Hiermit wir der Identifier __ajax::printscript__ angeboten, mit dem die notwendigen Dateien in eine HTML Seite eingebunden werden können. Die Verwendung ist fakultativ.
json2.js	Javascript für Kommunikation mit Server (Request).
JSON.pm	Perl für Kommunikation mit Client, Bereitstellung der Rückgabe im JSON Format.

Die Bibliothek

jquery.js	das jQuery Javascript Framework, falls benutzt.

Die Anwendungsdateien

ajaxsearch.js	Leitet den Ajaxcall und bearbeitet die Rückgabe.
scripts/ajaxsearch_identifier.ipl	Hiermit werden versch. Identifier z.B. __ajaxsearch::printscript__ angeboten, mit dem die notwendigen Dateien in eine HTML Seite eingebunden werden können. Die Verwendung ist fakultativ. Sie können die Einbindung der notwendigen Dateien auch direkt in das Template schreiben.
css/ajaxsearch.css	CSS Definitionen für ajaxsearch.
ajaxserach_conf.ipl	Hier werden Variablen zur Konfiguration definiert, falls ein Modul über Konfigurationsmöglichkeiten verfügt.
ajaxserach.ipl	Wird von ajax.html aufgerufen. Die angebotenen Methode heißt getPatterns und liefert Suchergebnisse anhand des übergeben Suchstrings .

Copyright 2017 by IBO Internet Software

3.9.3 Hinzufügen einer neuen Ajaxanwendung

Der Name der neuen Anwendung sei z.B. "selector". Die Anwendung soll auf Basis von Einstellungen, die der Anwender (z.B. mit Slidern) vornimmt. Artikel suchen, die in diese Eigenschaften haben. Bei jeder Veränderung durch den Anwender, ändert sich sofort, das dargestellte Ergebnis in Form der selektierten Artikel (deshalb Ajax).

1. Die Methode soll den Namen selectItems haben. Dieser Name wir in der Datei ajax.html eingetragen. Das zugehörige Modul soll ajaxselector.ipl heißen. Es wird ebenfalls eingetragen, damit es zur Laufzeit geladen werden kann.

```
my %methods =
 ('ajaxsearch::getPatterns' => 'ajax/ajaxsearch.ipl',  # Ajaxsuche
  'ajaxselector::selectItems' => 'ajax/ajaxselector.ipl'); # eine andere Methode
```

2 . Weiterhin benötigen wir die Implementierung der Methode die in ajaxselector.ipl erfolgt:

```
<!--spmacro:module(ajaxselector)

sub selectItems {
    my $pCall= $_[0];

    [...]

    return;

}
-->
```

3. Wir benötigen die Javascript Datei (ajaxselector.js) , welche die Abfrage durchführt.
Als Modell hierfür können wir ajaxsearch.js hernehmen. Autocomplete wird aufgerufen, wenn der Benutzer eine gewisse Anzahl Buchstaben getippt hat. Dann ein Request abgesetzt und eine onSuccess Funktion definiert. Hier zunächst eine anonyme Funktion, die nach der Umwandlung des Json Strings in eine Javascript-Datenstruktur schließlich getPatterns_handleSuccess mit den Ergebnisdaten als Parameter aufruft. Die Ergebnisdaten fließen dann in das DOM der Seite ein und werden hiermit für den Anwender sichtbar.

```
function autocomplete_js() {

  var postdata = {
       method : 'getPatterns',
       // Alternativ kann auch der Funktionsname inkl. Modul benutzt werden.
       // method: 'ajaxsearch::getPatterns',
       // Damit können identische Funktionsnamen in unterschiedlichen
       // SSP-Modulen benutzt werden.
       id : 0,
       params : [pattern, weitere Parameter ....]
       };

 try {
       req = new HTTP.Request(
       {
          uri: myinterface,
          postbody: JSON.stringify(postdata),  //für JSON2 geändert
          onSuccess: function (replydata) {
              var data;
              try {
                 data = eval('('+ replydata.responseText + ')'); // JSON "parsen"
              } catch(e) {
                 return;
              }
              selectItems_handleSuccess(data);
        }
       });

   }
   catch(e)
```

Copyright 2017 by IBO Internet Software

```
    }
      return;
    }
}

// Hier wir das Ergebnis (data) sichtbar gemacht
function selectItmes_handleSuccess(data){
       [...]
}
```

Alle Dateien außer der Datei „ajax.html“ aus dem Ordner „ajax“ und dessen Unterordner müssen dem Shop-Projekt als zusätzliche Dateien hinzugefügt werden.
Es ist klar, dass wir die angelegten Dateien in irgendeiner Form einbinden müssen.

Hierzu binden wir zunächst die SSP-Scripte ein.

<!--spmacro:include(ajax/ajax.ipl)-->
<!--spmacro:include(ajax/script/ajaxselect_identifiers.ipl)-->

Diese geben dann mit Hilfe der Identifier

__ajax::print_scripts__
__ajaxsearch::print_scripts__

die HTML Scripte aus, die notwendig sind, um die entsprechenden JavaScript und CSS-Dateien zu laden. Natürlich kann man stattdessen den Code auch direkt in die HTML-Seiten schreiben.

3.9.4 Module mit Ajax

Um Module mit Ajax zu installieren, muss folgender Codeabschnitt in ajax.html vorhanden sein.

```
if (defined %main::module_methods) {
       %methods = (%main::methods,%module_methods);
}
```

Es gibt folgende Konvention für die Erstellung von Modulen: mit Ajax.
Innerhalb der Modulverzeichnisse gibt es eine Datei mit dem Namen `module_ajax_method.ipl`. Diese Datei enthält die serverseitige Ajax-Funktion des Moduls. Der Code beginnt mit der Deklaration des SSP Moduls und enthält die Ajax-Funktion mit gleichem Namen. Zum Beispiel für ein Modul CeckAddress (Adressprüfung) sieht die Datei `module_ajax_method.ipl` so aus.

```
<!—spmacro:module(checkAddress)

sub checkAddress {
    my $pCall= $_[0];

    [...]

    return;
}
-->
```

Modulname und Funktionsname (Methodenname) müssen also identisch sein. Dann wird die Methode automatsich in den Hash %module_methods aufgenommen und kann aufgerufen werden, ohne dass die Methode explizit in das Hash Methods eingetragen wurde.

3.10 Filtern mit „can_display“

Es kann machmal sinnvoll sein eine alternative Sicht auf die Artikel im Shop zu ermöglichen. Man hat einen Kategoriebaum angelegt, der die Artikel im Shop strukturiert. In der Regel nach der Warengruppe z.B. Textilien / Hemd oder Textilien/ Hose. Nun könnten die Artikel aber auch bestimmten Zwecken zugeordnet sein z.B. festlich,

Copyright 2017 by IBO Internet Software

freizeit oder outdoor. Dann wünscht man sich eine Kategorie Festlich/Hemd und Festlich/Hose. Dieses Verhalten kann man mit den „can_display“ Callbackfunktionen erreichen.

Es wird dazu ein benutzerdef. Feld beim Artikel und der Seite angelegt, das wir FILTER nennen. Der Name ist aber eigentlich nicht relevant, da wir das Feld später über seine Position adressieren. Hier wird davon ausgegangen, dass es jeweils das erste benutzerdefinierte Feld ist. Falls es nicht das erste Feld ist muss die Position entsprechend erhöht werden. In disem Feld Filter legen wir beim Artikel die enstsprechenden Filterbgriffe ab. Um beim Besipiel zu bleiben eignet sich ein Hemd sowohl für Freizeit als auch für Outdoor. Dann legen steht im Feld Filter „Freizeit, Outdoor“. Im Filter-Feld der Seite legen wir die gleichen Begriffe ab, wenn sich Artikel eines Verwendungszewecks in der Seite oder auf Unterseiten befinden. Keine Angst, das brauchen wir nicht händisch zu machen, sondern das kann durch ein Script beim Veröffentlichen des Shops automatisch eingetragen werden.

Die URL zum Aufruf einer bestimmten Seite mit Filter soll so aussehen
http://www.....iboshop.cgi?show200,<sessionid>,,,filter=outdoor
Hier wird also die Shopseite 200 mit dem Filter outdoor aufgerufen.
Bei Aufruf einer Seite mit dem Identifier __catalog__ wird automatisch der Filterbegriff an die erzeugten Links angehängt. So dass man eine kompletten Bau erhält, der nur noch Artikel und Seiten eines Zwecks enthält.

Hier der Code im Modul Callback in der stdall.ipl. Der Filterbegriff wird jeweils der Aufrufurl entnommen und mit den Werten in der Seite oder im Artikel verglichen. Falls der Begriff enthalten ist soll der Artikel oder die Seite dargestellt werden, also wird ein 1 zurückgegeben. Im andern Fall eine 0 und der Artikel/Seite wird ausgeblendet.

```
sub can_display_page {
     my $token = $_[0];
     my $f = ssp::get_cmdln_arg(4);
     if (length($f) < 3) {return 1;}
     if ($token->[0] <= 9) {return 1;} # feste Seiten unter 10
     my ($d,$filter) = split(/=/,$f);
     if ($d ne 'filter'){return 1;}
     if ($token->[10] =~/$filter/) {
        return 1;
     }
     return 0;
}

sub can_display_user_item {
     my $token = $_[1];
     my $f = ssp::get_cmdln_arg(4);
     if (length($f) < 3) {return 1;}
     my ($d,$filter) = split(/=/,$f);
     if ($d ne 'filter'){return 1;}
     if ($token->[28] =~ /$filter/) {
        return 1;
     }
     return 0;
}
```

Copyright 2017 by IBO Internet Software