Dokumentation „Themes im yalst-LiveSupportTool“

Einleitung

Ein Theme für das yalst-LiveSupportTool ist eine Textdatei im XML-Format, in der das Aussehen der Fenster auf der Besucherseite beliebig angepasst und verändert werden kann.

Dieses erfolgt dabei für jeden Zugang/Lizenz separat, so dass bei einer Mehrlizenzinstallation bzw. auch bei der von Visisoft gehosteten Lösung eine Änderung des Designs, der graphischen Elemente, der Funktionalitäten, der Textausgaben möglich wird.

Dabei überschreiben die Einstellungen in den Themes alle anderen Einstellungen. Das betrifft sowohl die Installationseinstellungen, die der webbasierten Konfiguration als auch die im yalst-LiveSupportTool fest integrierten Eigenschaften.

Insgesamt sollte man beachten, dass der Erstellung und Bearbeitung die Datei in der Codierung gespeichert wird, die im XML-Header angegeben ist (hier: iso-8859-1). Wenn die Installation des yalst-LiveSupportTool in Unicode erfolgt ist, muss auch das entsprechende Theme im Unicode vorliegen und der Header entsprechende Informationen enthalten.

Hochladen kann man das Theme unter **Einstellungen - CSS und Themes** im webbasierten Frontend zur Konfiguration des einzelnen Zugangs/Lizenz. Dabei wird das Theme komplett hochgeladen als auch durch den Themeparser beim Upload in die einzelnen Teile zerlegt (Bilder, Sprachdateien, Javascript-Dateien usw.). Die einzelnen Dateien werden im Data-Verzeichnis der jeweiligen Installation abgelegt. Im untenstehenden Bespiel sind die entstandenen Dateien für eine Site mit dem Sitenamen 4-1 aufgelistet:

Beispiel – Dateiliste nach Parsing eines Themes im data-Verzeichnis für eine Site 4-1

```
theme_4-1.ytf

theme_4-1_images
theme_4-1_scripts.js
theme_4-1_styles.css
theme_4-1_vlang.de.ini
theme_4-1_vlang.en.ini
theme_4-1_vlang.es.ini
theme_4-1_vlang.fr.ini
theme_4-1_vlang.it.ini
```

Unter Umständen kann es nützlich sein, nach dem Upload des Themes oder bei Fehlern in der Darstellung sich die entstandenen Dateien nachträglich nochmals per FTP herunterzuladen, um mit einem entsprechenden Editor mit CSS- oder Javascript-Highlighting die Dateien einzeln auf Fehler zu durchsuchen. Durch das Gemisch verschiedener Syntaxregeln in den einzelnen Abschnitten kann es durchaus zu kleineren Fehlern kommen, die grosse Auswirkungen auf die Funktionsfähigkeit des Themes haben können. Zwar prüft auch der Parser das Theme auf einige grundsätzliche Probleme, wird aber nicht alle Fehler in der komplexen Datei finden.

Einige Fenster im yalst-LiveSupportTool, wie z. B. das eigentliche Chatfenster, bestehen aus historisch bedingt aus sichtbaren und z. T. unsichtbaren Frames. Das kann unter Umständen, insbesondere in Bezug auf Hintergrundbilder, dazu führen, dass Fotos nicht über das ganze Fenster durchgängig sichtbar sind, sondern in jedem Frame entsprechend von oben neu gerendert werden. Deshalb ist es empfehlenswert, Hintergrundbilder mit sich wiederholender Struktur zu verwenden.

Insgesamt erschwert das die Erstellung von Themes im allgemeinen, da man an verschiedenen Stellen im Theme Einstellungen ändern muss. In den mitgelieferten Beispielthemes würde Wert darauf gelegt, die Struktur der Fenster in der Reihenfolge der IDs nachzubilden.

Zur Erstellung eines neuen Themes kann man das Standard-Theme verwenden. In diesem sind allerdings keine CSS-Eigenschaften (siehe 4) für irgendein Element festgelegt und alle Einträge auskommentiert. Außerdem sind auch im Javascript-Abschnitt (siehe 5) nur sehr wenig beispielhafte Inhalte hinterlegt.

Der einfachere Weg ist die Nutzung eines bereits erstellen Theme, da dort bereits die wesentlichen Elemente mit CSS-Eigenschaften belegt sind. Weiterhin werden in den meisten Themes auch schon Icons getauscht bzw. neue Hintergrundbilder für die Seiten hinterlegt, so dass der Aufwand bei einer Änderung deutlich kleiner ist.

Die Struktur der Themes

1. Der XML-Header mit der Angabe der Codepage

Beispiel - XML-Header für eine XML-Datei im iso-8859-1-Format

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

2. Der Abschnitt für DOCTYPE-Einstellungen für Sonderzeichen

Beispiel - Behandlung von Sonderzeichen für eine XML-Datei im iso-8859-1-Format

```
<!DOCTYPE theme [
   <!ENTITY auml "&amp;auml;">
   <!ENTITY ouml "&amp;ouml;">
   <!ENTITY uuml "&amp;uuml;">
   <!ENTITY Auml "&amp;Auml;">
   <!ENTITY Ouml "&amp;Ouml;">
   <!ENTITY Uuml "&amp;Uuml;">
   <!ENTITY szlig "&amp;szlig;">
]>
<theme>
```

3. Der Abschnitt für Metadaten

Der Abschnitt für die Metadaten des Themes unter Angabe der Version des Themes und der Softwareversion des yalst-LiveSupportTools für die das Theme erstellt wurde

Beispiel – Metadaten einer Theme-Datei in der Version 1.0 für eine yalst-Version 9.0

```
<metadata>
        <version>1.0</version>
        <software>9.0</software>
        <author>Visisoft, Germany, www.visisoft.de</author>
</metadata>
```

4. Der CSS-Abschnitt

4.1 Allgemeine Struktur

Themes ermöglichen es, alle Elemente in den verschiedenen Besucherfenstern einzeln anzusprechen und mit neuen CSS-Eigenschaften zu versehen. Diese überschreiben sowohl die Einstellungen aus der webbasierten Konfiguration als auch die im yalst-LiveSupportTool fest integrierten Eigenschaften.

Der Bereich für die css-Definitionen ist durch

```
<css>
<content>
```

und

```
</content>
</css>
```

gekennzeichnet:

Beispiel – CSS-Definition für den <a>-Tag und Textfarbe eines Tabellenelement

```
<css>
        <content>

                a {outline: none;}
                /* a {color: #4545A7} */

                ……

                /* BEGIN: CHAT START WINDOW (NORMAL VISITORS) */
                /* START: CHATSTARTFENSTER (NORMALER BESUCHER) */

                 /* Text and Department Selector (if activated) */
                 /* Text und Abteilungsauswahl (falls eingeschaltet) */
                /* #indexphp_dept_select {} */
                #indexphp_dept_td {color:#000000;}

                        …..

        </content>
</css>
```

In den mitgelieferten Themes stehen dabei alle IDs sämtlicher Elemente aller Seiten. Diese können dann entsprechend mit beliebigen CSS-Eigenschaften versehen werden.

4.2 Nutzung von !important

Wenn unter Umständen die Vererbung nicht richtig zu funktionieren scheint bzw. es entsprechende inline-CSS-Definitionen gibt, hilft ein

```
!important
```

hinter der der Eigenschaft:

Beispiel – CSS-Definition für den Text- und Randfarbe einer Tabelle mit !important

```
#indexphp_button_buttonbox_table {background: #FFFFFF !important; border-color:#000000 !important;}
```

Dabei gibt der Teil der ID vor dem ersten Unterstrich jeweils den Namen der Seite/des Skriptes an. Damit hat man die Möglichkeit, neben allgemeinen Einstellungen für bestimmte Standardtags, jedes einzelne Element auf verschiedenen Seiten auch unterschiedliche Eigenschaften zuzuweisen.

4.3 Integration von Bildern

Um Bilder, z. B. für Hintergründe, Icons usw. verwenden zu können, müssen die Bilder im Bilderteil des Themes (siehe unten) base64-encoded eingebettet werden. Danach können Sie einfach unter Nutzung des PHP-Hilfsskriptes `timg.php` verwendet werden:

Beispiel – CSS-Definition für ein Hintergrundbild einer Seite

```
#indexphp_body {background-image: url(timg.php?name=bg.jpg&amp;site=%site%);}
```

Angepasst werden muss der Name des Bildes. Dabei ersetzt man

```
bg.jpg
```

durch den Namen des Bildes, unter dem es im Bilderabschnitt des Themes ablegt ist. Die Variable

```
%site%
```

ist ein Kürzel für die Nummer des Zugangs/Lizenz, der vom Themeparser beim Upload automatisch eingesetzt wird. Dort muss also nichts ersetzt werden.

Wie aus den Beispielen schon hervorgeht, sind die meisten CSS-Definitionen, wie auch die beschreibenden Textkommentare mit `/*` Ausdruck `*/` auskommentiert. So ist der Ausdruck:

```
/* #indexphp_dept_select {} */
```

nicht aktiv und wird durch den Themeparser beim Upload entfernt. Anders beim Ausdruck:

```
 #indexphp_dept_td {color:#000000;}
```

Dieser wird für das Design für das Element mit der ID „indexphp_dept_td“ berücksichtigt.

4.4 Rundungen und Buttons

Da insbesondere ältere Internetexplorer (unter Version 9) im Gegensatz zu Safari, Mozilla Firefox, Opera und Chrome keine einfache Möglichkeit bieten, Rundungen von Elementen direkt mit CSS zu konstruieren, verwendet das yalst-LiveSupportTool für die Rundungen der Boxelemente als auch der Buttons ein spezielles CSS-Konstrukt. Dieses ist unter:

http://www.webreference.com/programming/css_borders/index.html

erklärt.

Im Wesentlichen besteht dieses Konstrukt oben und unten (bei Button links und rechts) per CSS

definierte Linien, die mittels CSS-Eigenschaften gekürzt und übereinander gestapelt werden, so dass ein Treppeneffekt (Rundung) entsteht. Im Inneren befindet sich der Inhalt der Box mit einem normalen linken und rechten Rahmen.

Wie man diese Rundungen entfernt, wird im nächsten Beispiel erläutert. Im Beispieltheme „square“ ist dieses bereits realisiert. Wenn man also ein Design ohne Rundungen bevorzugt, wäre dieses ein guter Ausgangspunkt für die Erstellung eines neuen Themes.

Beispiel – Theming einer Roundbox mit Header

```
/* BEGIN: No Operator available box */
/* START: Kein-Operator-Verfuegbar-Box */

        /* Outer container for the No Operator available box */
        /* Auesserer Container fuer die Kein-Operator-Verfuegbar-Box */
        /* #inputchatphp_terminated_div {} */
        /* #inputchatphp_terminated_p {} */

        /* Conatiner for the No Operator available box */
        /* Container fuer die Kein-Operator-Verfuegbar-Box */
        /* #inputchatphp_terminated_roundbox_table {} */
        /* #inputchatphp_terminated_roundbox_tr {} */
        /* #inputchatphp_terminated_roundbox_td {} */
        /* #inputchatphp_terminated_roundbox_div {} */

        /* Top rounding */
        /* Obere Rundung */
        #inputchatphp_terminated_roundbox_linie1h {margin:0px; background: #000000}
        #inputchatphp_terminated_roundbox_linie2h {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#eeeeee;}
        #inputchatphp_terminated_roundbox_linie3h {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#eeeeee;}
        #inputchatphp_terminated_roundbox_linie4h {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#eeeeee;}

                /* Title of the No Operator available box */
                /* Titel der Kein-Operator-Verfuegbar-Box */
                #inputchatphp_terminated_roundbox_div_head {border-left: 1px solid #000000; border-
                        right: 1px solid #000000; border-bottom: 1px solid #000000; background-
                        color:#eeeeee;}
                /* #inputchatphp_terminated_roundbox_table_head {} */
                /* #inputchatphp_terminated_roundbox_tr_head {} */
                /* #inputchatphp_terminated_roundbox_td_head {} */
                /* #inputchatphp_terminated_roundbox_div_title {} */
                #inputchatphp_terminated_roundbox_span_title {color:#000000;}

                /* Text body of the No Operator available box */
                /* Textbody der Kein-Operator-Verfuegbar-Box */
                /* #inputchatphp_terminated_roundbox_div2 {} */
                #inputchatphp_terminated_roundbox_div_body {text-align: center; border-left: 1px solid
                        #00000; border-right: 1px solid #000000; background-color:#FFFFFF;}
                /* #inputchatphp_terminated_roundbox_table_body {} */
                /* #inputchatphp_terminated_roundbox_tr_body {} */
                #inputchatphp_terminated_roundbox_td_body {text-align: center; color:#000000;}
                #inputchatphp_terminated_roundbox_td_body a { color:#000000; text-decoration:
                        underline;}

        /* Bottom rounding */
        /* Untere Rundung */
        #inputchatphp_terminated_roundbox_linie4b {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#FFFFFF;}
```

```
        #inputchatphp_terminated_roundbox_linie3b {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#000000;}
        #inputchatphp_terminated_roundbox_linie2b {margin:0px; border-left: 1px solid #000000;
                border-right: 1px solid #000000; background-color:#FFFFFF;}
        #inputchatphp_terminated_roundbox_linie1b {margin:0px; background: #000000}

/* END: No Operator available box */
/* ENDE: Kein-Operator-Verfuegbar-Box */
```

In diesem Bespiel erkennt man die Struktur einer Roundbox. Diese wird von einem Container mit der ID `#inputchatphp_terminated_roundbox_div` umschlossen. Jeweils oben und unten (gekennzeichnet durch `/* Obere Rundung */` und `/* Untere Rundung */` erkennt man die notwendigen acht horizontalen Linien, vier oben und vier unten. Diese sind jeweils einen Pixel hoch.

Im Standardlayout werden durch Setzen von verschiedenen Rändern mittels Margins links und rechts die Rundungen konstruiert. Diese Rundungen werden in diesem Beispiel durch Änderung der CSS-Eigenschaften entfernt.

Durch die CSS-Eigenschaft `margin:0px` sind alle Linien wieder gleichlang mit der komplette Boxenbreite und sind nicht mehr gegeneinander verschoben. Die außen liegenden Linien (ganz oben und unten) werden auf die gewünschte Randfarbe der Box (hier: schwarz) gesetzt und ergeben dann die gerade Begrenzung der Box nach oben und unten.

Die weiter innen liegenden Linien, z. B. `#inputchatphp_terminated_roundbox_linie2h` erhalten durch die Randparameter `border-left: 1px solid #000000` und `border-right: 1px solid #000000` links und rechts einen schwarzen Randpixel versehen. Der Rest der Linie wird auf die Boxenfarbe gesetzt. So baut sich die gerade Begrenzung der Box nach links und rechts auf.

Genauso wird auch mit den inneren Containern für den Header `#inputchatphp_terminated_roundbox_div_head` und den Inhalt der Box `#inputchatphp_terminated_roundbox_div_body` verfahren. Auch diese erhalten mittels `border-left: 1px solid #000000` und `border-right: 1px solid #000000` einen linken und rechten Rand in der gewünschten Rahmenfarbe der Box und mit `background-color:#eeeeee` einen entsprechende Hintergrundfarbe. Zusätzlich erhält der Boxenheader mit `border-bottom: 1px solid #000000` eine Abschlusslinie nach unten.

Analog kann man auch die Buttons auf der Besucherseite von ihren Rundungen befreien. Dort ergibt sich die Rundung auch durch acht vertikale Striche. Davon liegen vier links und vier rechts vom eigentlichen Button.

Beispiel – Theming eines Buttons

```
/* BEGIN: Send button for the chat start */
/* START: Abschickenbutton zum Chatstart */

        /* #indexphp_form_submit {} */
        /* #indexphp_button_buttonbox_div {} */

        /* Left rounding */
        /* Linke Rundung */
        #indexphp_button_buttonbox_blinie1_left {height:21px; border-width: 1px 0; margin:0px;
                background:#000000;}
        #indexphp_button_buttonbox_blinie2_left {height:19px; border-width: 1px 0; margin:0px;
                background: #FFFFFF; border-color:#000000;}
        #indexphp_button_buttonbox_blinie3_left {height:19px; border-width: 1px 0; margin:0px;
                background: #FFFFFF; border-color:#000000;}
```

```
#indexphp_button_buttonbox_blinie4_left {height:19px; border-width: 1px 0; margin:0px;
        background: #FFFFFF; border-color:#000000;}

#indexphp_button_buttonbox_table {background: #FFFFFF !important; border-color:#000000
        !important;}
#indexphp_button_buttonbox_td {background: #FFFFFF !important; border-color:#000000
        !important;}

                /* Actual button */
                /* Eigentlicher Button */
                #indexphp_button {color:#000000; background: #FFFFFF !important;}

/* Right rounding */
/* rechte Rundung */
#indexphp_button_buttonbox_blinie4_right {height:19px; border-width: 1px 0; margin:0px;
        background: #FFFFFF; border-color:#000000;}
#indexphp_button_buttonbox_blinie3_right {height:19px; border-width: 1px 0; margin:0px;
        background: #FFFFFF; border-color:#000000;}
#indexphp_button_buttonbox_blinie2_right {height:19px; border-width: 1px 0; margin:0px;
        background: #FFFFFF; border-color:#000000;}
#indexphp_button_buttonbox_blinie1_right {height:21px; border-width: 1px 0; margin:0px;
        background:#000000;}

/* END: Send button for the chat start */
/* ENDE: Abschickenbutton zum Chatstart */
```

5. Der Javascript-Abschnitt

5.1 Allgemeine Struktur

Themes ermöglichen es, in die Besucherfenster eigene Skripte zu integrieren. Damit ist es möglich, zusätzliche Funktionalitäten in die Fenster einzufügen.

Der Bereich für die css-Definitionen ist durch <javascripts> und </javascripts> gekennzeichnet:

```
<javascripts>
        <javascript location="head" file="index.php">
                <content>
                        Javascripts (head) im oberen Teil der Seite index.php
                </content>
        </javascript>
        <javascript location="body" file="index.php">
                <content>
                        Javascripts (body) im unteren Teil der Seite index.php
                </content>
        </javascript>
</javascripts>
```

Weiterhin können bereits mitgelieferte Skripte genutzt werden, um graphischen Elemente aus den webbasierten Einstellungen sowie die mit der Installation mitgelieferten Header und Icons zu überschreiben.

5.2 Sichtbarkeitsänderung der Seite beim Javascript-basiertem Bildertausch

Insbesondere, wenn alle Icons auf der Seite gesetzt werden sollen bzw. ein größeres Hintergrund verwendet werden soll, kann es sein, dass die Standardelemente kurz sichtbar werden. Das liegt am unterschiedlichen Timing des Renderns der Seite und des javascript-basierten Austausches der Bilder.

Beispiel – Sichtbarkeit und Bildertausch in einem Theme

(a) Den Body der Seite per CSS auf „unsichtbar“ setzen.

```
#indexphp_body {visibility:hidden}
```

Diese css-Eigenschaften, die im css-Abschnitt des Themes (siehe oben) definiert werden muß, setzt die ganze Seite index.php (den Body) initial auf unsichtbar mittels visibility:hidden.

(b) Hinterlegung des Bildes „logout.png“ im Bilderabschnitt

```
<image name="logout.png">
        <content>
        <![CDATA[
        base64-codeirtes Bild
        ]]>
        </content>
</image>
```

Um das Bild „logout.png“ nutzen zu können, muss es im Bilderbereich base64-codiert hinterlegt werden (siehe Abschnitt 7).

(c) Definition des Bildes im Header der Seite mittels der JS-Funktion use_image

```
var button_logout=use_image('logout.png');
```

Hier wird eine Variable „button_logout“ generiert, die das Bild „logout.png“ enthält. Dazu wird die mitgelieferte Funktion use_image benutzt, die bereits entsprechend in die Seite integriert ist.

(d) Tausch des Bildes im unteren Teil der Seite mittels der JS-Funktion replace_image

```
replace_image('index_logout_img',button_logout);
```

In diesem Schritt wird das Element mit der ID „index_logout_img“ durch das Bild in der Variable „button_logout“ getauscht. Dazu wird die mitgelieferte Funktion replace_image genutzt, die bereits in die Seite integriert ist.

(e) Seite anzeigen mittels der JS-Funktion make_visible nach dem Tausch der Bilder

```
make_visible('indexphp_body');
```

Hierbei wird die mitgelieferte Funktion make_visible genutzt, um die Seite index.php nach dem Tausch aller Bilder wieder anzuzeigen. Damit wird die CSS-Eigenschaft aus (a) wieder aufgehoben.

Insgesamt würde das Teil für das Theme dafür dann so aussehen:

```
<css>
	<content>
		#indexphp_body {visibility:hidden}
	</content>
</css>

…..

<javascripts>
	 <javascript location="head" file="index.php">
		<content>
			 var button_logout=use_image('logout.png');
		 </content>
	 </javascript>
	<javascript location="body" file="index.php">
		<content>
			replace_image('index_logout_img',button_logout);
			make_visible('indexphp_body');
		</content>
	</javascript>
</javascripts>
…..
<images>
	<image name="logout.png">
		<content>
		<![CDATA[
		base64-codeirtes Bild
		]]>
		</content>
	</image>
</images>
```

5.3 Austausch von Bildern mit Mouse-Over

In einem weiteren, komplizierteren Beispiel wird gezeigt, wie man mit Icons umgeht, die sich bei Mouse-Over verändern sollen. Angenommen man möchte auf der Seite content.history.php den Logoutbutton tauschen und für den Mouse-Over-Fall eine andere Graphik festlegen.

Beispiel – Sichtbarkeit und Bildertausch mit Mouseover in einem Theme

(a) Den Body der Seite per CSS auf „unsichtbar“ setzen.

```
#contenthistoryphp_body {visibility:hidden}
```

Diese css-Eigenschaften, die im css-Abschnitt des Themes (siehe oben) definiert werden muß, setzt die ganze Seite content.history.php (den Body) initial auf unsichtbar mittels visibility:hidden.

(b) Hinterlegung der Bilder „logout.png“ und „logout_red.png“ im Bilderabschnitt

```
<image name="logout.png">
        <content>
        <![CDATA[
        base64-codiertes Bild logout.png
        ]]>
        </content>
</image>
<image name="logout_red.png">
        <content>
        <![CDATA[
        base64-codeirtes Bild logout_red.png
        ]]>
        </content>
</image>
```

Um die Bilder „logout.png“ und „logout_red.png“ nutzen zu können, müssen beide im Bilderabschnitt des Themes base64-codiert hinterlegt werden (siehe Abschnitt 7).

(c) Tausch des Bildes mittels der JS-Funktion replace_image

```
if (document.getElementById('reguserincphp_logout_link'))
        {
        document.getElementById('reguserincphp_logout_link').setAttribute("onmouseover",
        "document.logout.src='timg.php?name=logout_red.png&amp;site=%site%';");
        }

if (document.getElementById('reguserincphp_logout_link'))
        {
        document.getElementById('reguserincphp_logout_link').setAttribute("onmouseout",
        "document.logout.src='timg.php?name=logout.png&amp;site=%site%';");
        }
```

In diesem Schritt werden per Javascript für Mouse-Over und Mouse-Out die entsprechenden Pfade mittels der PHP-Hilfsfunktion timg.php festgelegt. Dabei ist in diesem Fall die ID reguserincphp_logout_link zuständig, da in diesem speziellen Fall die komplette Navigation für registrierte Benutzer zentral für alle betroffenen Skripte in einer reguser.inc.php festgelegt wird. Dieser Tausch sollte in dem Javascript für den unteren Teil der Seite festgelegt werden.

(d) Seite anzeigen mittels der JS-Funktion make_visible nach Tausch der Bilder

```
make_visible('contenthistoryphp_body');
```

Hierbei wird die mitgelieferte Funktion make_visible genutzt, um die Seite content.history.php nach dem Tausch aller Bilder wieder anzuzeigen. Damit wird die CSS-Eigenschaft aus (a) wieder aufgehoben.

Insgesamt würde das Teil für das Theme dafür dann so aussehen:

```
<css>
	<content>
		#contenthistoryphp _body {visibility:hidden}
	</content>
</css>

…..

<javascripts>
	<javascript location="body" file="index.php">
		<content>
			if (document.getElementById('reguserincphp_logout_link'))
				{
				document.getElementById('reguserincphp_logout_link').
				setAttribute("onmouseover","document.logout.src='timg.php?
				name=logout_red.png&amp;site=%site%';");
				}

			if (document.getElementById('reguserincphp_logout_link'))
				{
				document.getElementById('reguserincphp_logout_link').
				setAttribute("onmouseout","document.logout.src='timg.php?
				name=logout.png&amp;site=%site%';");
				}
		</content>
	</javascript>
</javascripts>

….....

<images>
	<image name="logout.png">
		<content>
		<![CDATA[
		base64-codiertes Bild logout.png
		]]>
		</content>
	</image>
	<image name="logout_red.png">
		<content>
		<![CDATA[
		base64-codeirtes Bild logout_red.png
		]]>
		</content>
	</image>
</images>
```

5.2 Integration von zusätzlichen, sprachabhängigen Texten

Im nächsten Beispiel wird gezeigt, wie wir einen zusätzlichen Text vor und nach dem Zeichenzähler im Besucherchat einfügen. Dieser wird im Skript input.chat.php generiert.

Beispiel – Integration von sprachabhängigen Texten mittels Javascript

(a) Integration der Texte in den Sprachabschnitt von input.chat.php (siehe auch 6)

```
<langs>
…..

<lang file="input.chat.php" language="de">
        <content>
        <![CDATA[
        ……
        vorzeichenzaehler = "noch "
        nachzeichenzaehler = " Zeichen"
         ]]>
         </content>
</lang>
<lang file="input.chat.php" language="en">
        <content>
         <![CDATA[
        ….
        vorzeichenzaehler = ""
        nachzeichenzaehler = " characters remaining"
        ]]>
        </content>
</lang>
<lang file="input.chat.php" language="es">
        <content>
        <![CDATA[
        ….
        vorzeichenzaehler = ""
        nachzeichenzaehler = " caracteres restantes"
        ]]>
        </content>
</lang>
<lang file="input.chat.php" language="fr">
        <content>
        <![CDATA[
        ……
        vorzeichenzaehler = ""
        nachzeichenzaehler = " caractères restants"
        ]]>
        </content>
</lang>
<lang file="input.chat.php" language="it">
        <content>
        <![CDATA[
        ….
        vorzeichenzaehler = ""
        nachzeichenzaehler = " caratteri rimanenti"
        ]]>
        </content>
</lang>
……
</langs>
```

In diesem Schritt wurden zwei neue Sprachvariablen vorzeichenzaehler und nachzeichenzaehler definiert. Diese müssen dabei in allen vorhandenen Sprachen in den Sprachabschnitt der index.chat.php (siehe auch Abschnitt 6) in dem jeweiligen Theme integriert werden. Wie man sieht, können dabei auch leere Strings für einzelene Variablen definiert werden, wenn einer der beiden Texte in einer der Sprachen nicht benötigt wird.

Nun kann man im Javascriptabschnitt desselben Themes auf diese zugreifen.

(b) Integration der Texte mittels Javascript vor und hinter den Zeichenzähler

```
if (document.getElementById('inputchatphp_text_remaining_form'))
	{
	var geschwister=document.getElementById('inputchatphp_text_remaining_form');
	var vorgeschwister=document.createTextNode('%lang(vorzeichenzaehler)%');
	var eltern=geschwister.parentNode;
	eltern.insertBefore(vorgeschwister,geschwister);
	var nachgeschwister=document.createTextNode('%lang(nachzeichenzaehler)%');
	eltern.appendChild(nachgeschwister);
	}
```

In diesem Schritt mittels Javascript vor und hinter dem inputchatphp_text_remaining_form Element die entsprechenden Texte eingefügt. Dieses muss natürlich im unteren Javascriptabschnitt für das entsprechende Skript index.chat.php erfolgen.

Der Ausdruck %lang(nachzeichenzaehler)% ist hierbei ein theme-interner Platzhalter für die Sprachvariablen nachzeichenzaehler aus dem Sprachabschnitt desselben Themes. Da wir dort ja schon die Texte für die verschiedenen Sprachen erzeugt haben, werden jetzt vom Skript je nach Sprache die richtigen Texte automatisch integriert.

Insgesamt würde der Teil für das Theme dafür dann so aussehen:

```
<langs>
	…..

	<lang file="input.chat.php" language="de">
		<content>
		<![CDATA[
		…...
		vorzeichenzaehler = "noch "
		nachzeichenzaehler = " Zeichen"
		 ]]>
		 </content>
	</lang>
	<lang file="input.chat.php" language="en">
		<content>
		 <![CDATA[
		….
		vorzeichenzaehler = ""
		nachzeichenzaehler = " characters remaining"
		]]>
		</content>
	</lang>
```

```
		<lang file="input.chat.php" language="es">
			<content>
			<![CDATA[
			….
			vorzeichenzaehler = ""
			nachzeichenzaehler = " caracteres restantes"
			]]>
			</content>
		</lang>
		<lang file="input.chat.php" language="fr">
			<content>
			<![CDATA[
			…...
			vorzeichenzaehler = ""
			nachzeichenzaehler = " caractères restants"
			]]>
			</content>
		</lang>
		<lang file="input.chat.php" language="it">
			<content>
			<![CDATA[
			….
			vorzeichenzaehler = ""
			nachzeichenzaehler = " caratteri rimanenti"
			]]>
			</content>
		</lang>
		…...
</langs>
…...
<javascripts>
…...
		<javascript location="body" file="input.chat.php">
			<content>
			….....
			if (document.getElementById('inputchatphp_text_remaining_form'))
			{
			 var geschwister=document.getElementById('inputchatphp_text_remaining_form');
			 var vorgeschwister=document.createTextNode('%lang(vorzeichenzaehler)%');
			 var eltern=geschwister.parentNode;
			 eltern.insertBefore(vorgeschwister,geschwister);
			 var nachgeschwister=document.createTextNode('%lang(nachzeichenzaehler)%');
			 eltern.appendChild(nachgeschwister);
			}
			</content>
		</javascript>
…...
</javascripts>
```

6. Der Sprachabschnitt

6.1 Allgemeine Struktur

Themes erlauben es, sämtliche Textausgaben auf der Besucherseite für alle verfügbaren Sprachen anzupassen. Dazu sind alle Texte der verschiedenen Sprachen im Theme vorhanden.

Der Bereich für die css-Definitionen ist durch <langs> Textvariablen </langs> gekennzeichnet:

Beispiel – Textausgaben für das Skript privacy.php (Datenschutzhinweise)

```
<langs>
…..
        <lang file="privacy.php" language="de">
                <content>
                <![CDATA[
                ; Datenschutz-Hinweise
                ; datenschutzhinweise = "Datenschutz-Hinweise"
                ; drucken = "Text ausdrucken"
                ; fenster_schliessen = "Fenster schlie&szlig;en"
                 ]]>
                 </content>
        </lang>
        <lang file="privacy.php" language="en">
                <content>
                 <![CDATA[
                ; privacy information window
                ; datenschutzhinweise = "Privacy Policy"
                ; drucken = "Print text"
                ; fenster_schliessen = "Close window"
                ]]>
                </content>
        </lang>
        <lang file="privacy.php" language="es">
                <content>
                <![CDATA[
                ; Privacidad de datos
                ; datenschutzhinweise = "Privacidad de datos"
                ; drucken = "Imprimir texto"
                ; fenster_schliessen = "Cerrar ventana"
                ]]>
                </content>
        </lang>
        <lang file="privacy.php" language="fr">
                <content>
                <![CDATA[
                ; privacy information window
                ; datenschutzhinweise = "Charte de confidentialit&eacute;"
                ; drucken = "Imprimer le texte"
                ; fenster_schliessen = "Fermer la fen&ecirc;tre"
                ]]>
                </content>
        </lang>
```

```
        <lang file="privacy.php" language="it">
                <content>
                <![CDATA[
                ; privacy information window
                ; datenschutzhinweise = "Indicazioni sulla protezione dei dati"
                ; drucken = "Stampare testo"
                ; fenster_schliessen = "Chiudere finestra"
                ]]>
                </content>
        </lang>
…...
</langs>
```

Das obige Beispiel macht anhand der Textausgaben für der Seite „Datenschutz-Hinweise“ (privacy.php) klar, wie das Strukturierung für die Sprachen funnktioniert. Sämtliche Textausgaben aller Seiten sind, wie oben bereits erläutert, durch den <langs>-Tag eingeschlossen.

6.2 Die Sprachen

Die Unterstrukturierung der entsprechenden Seite für die fünf verschiedenen Sprachen erfolgt durch:

```
<lang file="privacy.php" language="it">
        <content>
        <![CDATA[
        Textvariablen
        ]]>
        </content>
</lang>
```

Hierbei wird das entsprechende Skript im file- und die Sprache im language-Parameter (de – deutsch; en – englisch; es – spanisch, fr – französisch, it - italienisch) definiert. In diesem Beispiel werden also die italienischen Texte in der Datei privacy.php geändert.

Um den Inhalt zu markieren und eventuelle Probleme mit Sonderzeichen zu umgehen, wird der eigentliche Inhalt durch

```
<content>
<![CDATA[
```

und

```
]]>
</content>
```

geklammert.

Die Texte werden in den Sprachdateien und im Theme als Variablen an das entsprechende Skript übergeben. Wie aus den Beispielen schon hervorgeht, sind die meisten Textausgaben, wie auch die Beschreibungen mit dem Kommentarzeichen

```
;
```

auskommentiert. Um z. B. den englischen Titel des Datenschutz-Hinweis-Fenster zu ändern, muss man das Kommentarzeichen löschen und den Text entsprechend ändern.

Aus dem Ausdruck:

```
; datenschutzhinweise = "Privacy Policy"
```

wird dann also:

```
datenschutzhinweise = "My Own Text"
```

Eine Sonderstellung nehmen die Abschnitte „notify“, „ratings“ und „transcript“ am Anfang des Sprachabschnittes ein, da diese von verschiedenen Skripten im yalst-LiveSupportTool gleichermaßen genutzt werden. Insbesondere der „transcript“-Teil ist dabei interessant, da dieser die Textausgaben für die E-Mail-Transkripte festlegt, die nach einem Chat an den Besucher verschickt werden.

Wenn z. B. in den deutschen E-Mail-Transkripten Operatoren Supporter heißen sollen, würden sich folgende Änderungen ergeben:

Beispiel – Ersetzung von Texten im Chattranskripten

```
<lang file="transcript" language="de">
	<content>
	<![CDATA[
		; Transkript-Versand
		; datei = "Datei"
		; push = "Push"
		; transfer = "<Transfer> Chat-Transfer"
		operator = "Supporter"
		operatoren = "Supporter"
		; min = "Min."
		; am = "am"
		; um = "um"
		; uhr = "Uhr"
		; dauer = "Dauer"
		; title = "Transkript des Supportgesprächs auf"
		; titledu = "Transkript Deines Supportgesprächs auf"
		; img = "Bild"
		; cobrowse = "Cobrowse"
		; aclose = "Chat wurde wegen Zeitkonto-Überschreitung beendet."
		keineannahme = "Chat wurde vor Annahme durch einen Supporter beendet"
		; abteilung = "Abteilung"
		; einladung = "Einladung zur Remote-Unterstützung wurde übermittelt"
		; kennwort = "Kennwort zur Remote-Unterstützung wurde übermittelt"
		; zeitkonto = "Zeitkonto"
		; ich = "Ich"
		; sie = "Sie"
		; du = "Du"
	]]>
	</content>
</lang>
```

Die auskommentierten Texte können dabei im Theme für spätere Änderungen belassen werden, da der Themeparser diese beim Upload des Themes automatisch entfernt.

7. Der Bilderabschnitt

Um eine problemlose Weitergabe verschiedenster Designs mit Graphiken in einer Datei zu ermöglichen, müssen alle Bilder im Theme als base64-codiert hinterlegt werden. Dieses erfolgt im Bilderabschnitt, welcher durch <images> Bilder </images> gekennzeichnet ist. In diesem Fall wurde ein Bild mit dem Namen clock.png hinterlegt.

Beispiel – Ersetzung für die Einbindung eines base64-codierten Bildes

```
<images>

…..

        <image name="clock.png">
                <content>
                <![CDATA[
iVBORw0KGgoAAAANSUhEUgAAABoAAAAZCAIAAAHXu5mhAAAACXBIWXMAAAsSAAALE
gHS3X78AAAACXRFWHRDb21tZW50AACJKo0GAAAEm0lEQVR4nG2VaSimbxTGH+QLItsHihEK
iZL1iy3GMkNZkyUUSRISM9IoZCZS8sEylhph/lMoSzJ2GbKmGYYw9rGN3VefzO+Z+19X/OfU+/bfd/
Pec59znVd5zzSzc3Nr1+/9vb2WEjX19eSJB0cHOzs7Kg2h4eHEk8vLi6OjIyPyKiIi4ujo6MWLF1
JhYeHj46OHh4f8dH19XeG3tbUVEBBgY2OTkJDw4cMH+fTs7Cw6Opq4t7e3DQ0N3LGwsCD7rq6uVl
VVsXd1dSWWIgL2/fv32dnZy8tLxVF6erq7u7uDg8O7d+9ISRobG7O1tSXi3d0dr/f19UmvXr1iRRQND
Q0eDA0NSVNTU1ZWVmw45dnV1ZVcYltbm52dHXtuJzPFjeCxvb2NiyoJfHd3d9+8eRMfH5+UlJSTk1
NaWvrp0yeBicLv5OQkKyvL39/fzMxMUjMXFxdKHRgYAESJX0FBQUhIiK6ubn19PenwGmCwqKur
MzEx6e3tXVpakv3evn2bmppKAB48qhlbDqenp4FGvhcSampqiGpkZKR+r7Oz8/z8PLAAnqoO2O39Yy
A1OTn548cPUKF4VR1KV7ifmZkZHR0FntPTU06UTxV+ZDk+Pp6WlhYXFwcxr1+/rq6u/vr1qwBP4Y
dTZ2dnZGSkj4+Pk5MTbPn5+RUVFVE+lSpwJjgcRUVFQa6Ojo6yCENDw9zc3I6ODvKW/UArMzOT
ADg9e/YMOLS1tflnTflNTU2Dg4Nyvd+fUtMTHR0dBRKwDQ1NcWCk5iYmImJCWJJFEj61tbWSj97e
3vhygnyJUUAlsAahIVaHp8aJ/CJAxjJ+VVUVCB4TkV+wlhzUltbS1PI+VEvysUVdUhPLTY2lhe4VIEf5I
Bfa2srDWdqaooHAgMUnJCqYEXBB65zc3P9/f3wS5eS+/39/f7+/hM+lPyS6+bm5sbGBh7MgX/wq25kjd
zBtaenh06gB8iBzi8pKaEVP378SJuiJlJQ3vaPcFzCVYDQ3d0NkegmLCwsKCgoODg4PDycbXJyMudgX
V5ejqybm5sXFxd//vypHlQRjiMAIB2kBzZ0nqenJ+PG2NhYT09PiaeBgYGlpaWvr29eXh4RW1pagFN0v
Ao8RAoQlZWVUBcYGOjl5QXIQohQR8uh6uXlZcpnLchEb25ubsXFxYwRkiBNAJUHJ3mh1sbGxpSU
lNDQUJIiIxFITCAMKLW0tGhOseVcBEVhIAt9nz9/XltbOz8/l/jRx/n5+Uyely9fCj8xtJTykqeXJFEBGnx
4eOCEp8KNXiEcOieIrGTCkSqsZWdnUyztKPy4TV3V1PLfHyMiW54yTXGDK1oHNFZWVqBFEiS0t
7ejCcaft7e3vr7+X8X+Zcpizc3NMzIyiIUeGLlysYIKviEwhQLQF/kzBESOggpKIwT/SiowCwsL+OVdLi
CWIFc1phA6tTByaEEy5VslOPm/EYgO/PLli0iW4pS9oZIxVYMLPc9Y7+rqQv2IoKysDPUyM54/f46ek
fH79+9FRiIQr/xDxuq9Ib6zwMFncXh4mOEKcbQxYAMTsNBhoP5Xtwr7DSMlwbxxh/1VAAAAAElFT
kSuQmCC
                ]]>
                </content>
        </image>

…..

</images>
```

Im obigen Beispiel sehen wir ein solches Bild. Das einzelne Bild ist dabei wieder durch den <image>-Tag geklammert und enthält seinen Namen als Parameter. Dieser ist wichtig, um mittels Javascript-Funktionen (siehe 5):

```
var button_logout=use_image('logout.png');
```

oder direkt in den CSS-Eigenschaften (siehe 3) über das PHP-Hilfsskript timg.php:

```
#indexphp_body {background-image: url(timg.php?name=clock.png&amp;site=%site%);}
```

ansprechen zu können.

Um den Inhalt zu markieren und eventuelle Probleme mit Sonderzeichen zu umgehen, wird der eigentliche Inhalt durch

```
<content>
<![CDATA[
```

und

```
]]>
</content>
```

geklammert.

Für die base64-Codierung gibt es offline als auch im Internet eine Menge Möglichkeiten, wie z. B. der base64-Encoder unter:

http://www.opinionatedgeek.com/dotnet/tools/Base64Encode/Default.aspx