

WebGate Anywhere

Whitepaper und Benutzerhandbuch „Portal Server“

Copyright © 1999 by: Innovation Gate GmbH
40880 Ratingen
Harkortstr. 21-23
Tel.: 02102 - 77 16 00
www.innovationgate.de
info@innovationgate.de

Alle Rechte vorbehalten, kein Teil des Handbuches sowie des dazugehörigen Programms darf in irgendeiner Form (Druck, Fotokopie oder sonstige Verfahren) ohne schriftliche Genehmigung von Innovation Gate reproduziert oder vervielfältigt werden.

Der Kunde erhält das Programm inklusive Handbuch und Datenträger.

Für die Fehlerlosigkeit des Programms und für Schäden, die durch die Benutzung des Programms oder dieses Handbuches entstehen, kann Innovation Gate leider keine Haftung übernehmen.

Die Benutzung des Programms erfolgt ausschließlich aufgrund der Lizenzbestimmungen, die dem Produkt beiliegen.

Änderungen am Handbuch und den beschriebenen Produkten bleiben jederzeit und ohne vorherige Ankündigung vorbehalten.

Version: 1.9
letzte Änderung: 11. August 2003

Inhaltsverzeichnis

1. Der Weg zur dynamischen Website.....	3
1.1 Personalisierung als Schlüssel zum Erfolg.....	3
2. Der WebGate Anywhere Portal Server	4
2.1 Die Personalisierungskomponente.....	5
2.2 Die WGA Portal Services.....	5
3. WebGate Anywhere Personalisierung	5
3.1 Wo werden Profildaten abgelegt?.....	6
3.2 Wie werden Benutzer identifiziert?.....	7
3.3 Benutzerklassen.....	8
3.3.1 Verwenden von Benutzerregeln.....	8
3.4 Abfragen von Benutzerdaten per TML.....	9
3.5 Abfragen von Benutzerdaten per TMLScript.....	9
3.6 Speichern von Profildaten.....	9
3.7 Personalisierungsmode "Custom".....	10
3.7.1 Aktivieren des "Custom"-Modus.....	10
3.7.2 Programmieren von Erstellung und Zuweisung von Profilen.....	10
3.7.3 Beispiel: Maske zur Erstellung eines Benutzerprofils.....	11
3.7.4 Beispiel: Maske zur Zuweisung eines "Custom"- Benutzerprofils als Anmeldung.....	12
3.7.5 Weitere WebTML/TMLScript-Funktionalitäten rund um die Personalisierung und den "Custom"- Modus.....	13
4. WebGate Anywhere Portal Services.....	14
4.1 Was sind Portlets?.....	14
4.2 Registrieren von Portlets.....	14
4.3 Einfügen von Portlets in das Templatedesign.....	15
4.4 Zugriff auf die Konfiguration eines Portlets.....	15
4.5 Portlet Modes.....	16
4.6 Verschachteln von Portlets.....	17

1. Der Weg zur dynamischen Website

Es ist noch nicht lange her, da erwartete man selbst von der Website eines professionellen Unternehmens wenig mehr als oberflächliche Informationen zu seinen Produkten und Dienstleistungen, welche vielleicht durch Bilder oder herunterladbare Broschüren angereichert wurden. Die einzige Möglichkeit zur Interaktion bestand oft nur aus einer publizierten E-Mail-Adresse, über welche man den Kontakt vertiefen konnte. Die Unternehmens-Website war aus betrieblicher Sicht nicht mehr als eine Visitenkarte, somit das Internet insgesamt nicht viel mehr als ein Branchenbuch.

Dieses Bild hat sich in den vergangenen Jahren außerordentlich gewandelt. Mit der Erkenntnis, dass die Website nicht nur als Aushängeschild taugt, sondern ganze Bereiche des täglichen Geschäfts unterstützen oder gar übernehmen kann - wie z.B. Kontaktaufnahme mit Interessenten, Interne Kommunikation, Informationen für Kunden, ja sogar Verkauf und Distribution selbst - stieg der Anspruch gegenüber dem Nutzwert einer Website und somit auch gegenüber den technischen Voraussetzungen für deren Realisierung. Spezielle Konzepte zur dynamischen Darstellung und Interaktion wurden entwickelt, damit die moderne Website ihre Aufgabe möglichst effektiv erfüllen konnte. Das Web wurde „intelligent“.

1.1 *Personalisierung als Schlüssel zum Erfolg*

Ein Schlüsselbegriff, welcher immer wieder im Kontext der „intelligenten Website“ auftaucht ist die Personalisierung. Er beschreibt im Wesentlichen das Vorhaben, den Inhalt der eigenen Website auf den spezifischen Besucher anzupassen. Dies kann auf verschiedenste Weise geschehen. Einige Websites begnügen sich damit, den Namen des Besuchers zu erfragen, um ihn persönlicher begrüßen zu können. Andere bieten ihm an, ein Interessenprofil zusammenzustellen und zu speichern, damit der Inhalt der Website sich an dieses dynamisch anpassen kann. Wieder andere werten aus, welche Seiten ein Besucher wählt, um daraus auf ein spezielles Interesse schließen und ihm gezielter Angebote unterbreiten zu können. Der Grundgedanke ist aber überall derselbe: Man möchte den Besucher gezielter ansprechen und so den Erfolg einer Website, was auch immer ihr individueller Zweck sein mag, maximieren.

Einige Anwendungsformen von Websites bieten sich speziell für die Personalisierung an. Im Themenbereich „Intranet“ gibt es mannigfaltige Anwendungsgebiete, um die dargestellten

Informationen auf den Besucher - in diesem Fall den Mitarbeiter – anzupassen. Mitarbeiter unterschiedlicher Abteilungen und Standorte interessieren sich naturgemäß für unterschiedliche Inhalte und wollen ihre jeweiligen Favoriten bevorzugt präsentiert sehen.

Das Internet-Portal, eine andere sehr populäre Website-Anwendung, ist ohne Personalisierung kaum sinnvoll umsetzbar. Ein Portal bietet in der Regel eine derartige Menge und Bandbreite von Informationen an, dass der Benutzer ohne Hinterlegung eines Interessenprofils kaum eine Chance hat, schnell einen Überblick über die für ihn relevanten Inhalte zu erlangen. Zudem wird von einem Portal erwartet, dass der Benutzer dessen Layout nach seinen speziellen Vorlieben verändern kann. Diese Einstellungen muss der Benutzer natürlich speichern können, damit er nicht bei jedem Besuch aufs Neue konfiguriert.

Durch diese Beispiele wird ersichtlich, dass eine Personalisierungs-Lösung, die dieser Bandbreite an Anforderungen gerecht werden will, einem sehr universellen Ansatz folgen muss. Sie muss dem Website-Designer alle denkbaren (und momentan noch undenkbaren) Möglichkeiten der besucherspezifischen Anpassung von Content ermöglichen und ihm eine Plattform für seine spezielle Interpretation der Personalisierung bieten. Im Kontext von WebGate Anywhere, der Content Management-Lösung der InnovationGate GmbH, will die WebGate Anywhere Personalisierung diesen Anforderungen gerecht werden.

2. Der WebGate Anywhere Portal Server

Der WGA Portal Server erweitert den WGA CMS Server um zwei neue Komponenten:

- Die Personalisierungskomponente
- Die WGA Portal Services

Diese Komponenten werden im folgenden grundsätzlich beschrieben. Weitere Details und Codebeispiele sind in Bereich Tips&Tricks im Developer Center (<http://www.innovationgate.de>) zu finden.

Personalisierung und Portal Services arbeiten eng zusammen mit den ebenfalls in WGA 2.1 neu eingeführten Konzepten

- TML-Actions
- TML-Forms

Diese Konzepte werden in separaten Abschnitten beschrieben.

2.1 Die Personalisierungskomponente

... bietet die Möglichkeit, jeden User mit einem persönlichen Profil zu versehen. In diesem Profil können beliebige Daten über den Benutzer abgelegt werden, etwa Name, Interessen, Abteilung etc.

Diese Informationen können von Designer einer Website oder eines Intranets dazu verwendet werden, dem User genau diejenigen Informationen zu präsentieren, die seinem Profil entsprechen. So können z. B. Vertriebsmitarbeitern Vertriebsinformationen und Projektmitarbeitern Projektinformationen angeboten werden.

Ziel ist es dabei immer, die Gesamtmenge an Informationen einer Site für jeden Benutzer individuell so weit zu reduzieren, daß er in kürzester Zeit die "passenden" Daten für seine Arbeit zur Verfügung hat.

2.2 Die WGA Portal Services

... sind entwickelt worden, um Websites oder Intranets als "Unternehmensportale" ausbauen zu können.

Das wesentliche Konzept von Portalen ist es, dem Benutzer persönliche Sichten auf Unternehmensdaten und Unternehmensanwendungen zu ermöglichen. Diese "Sichten" werden im Portal in Form von "portlets" eingebunden.

Portlets sind eigenständige Module mit definierter Funktionalität, die im Portal beliebig zusammengestellt werden können.

Portlets besitzen in der Regel benutzerabhängige Konfigurationen, die im Benutzerprofil abgespeichert werden. So kann z. B. in einem CRM-Portlet der augenblicklich ausgewählte Kunde in der (persönlichen) Portletkonfiguration abgespeichert werden.

Für die Verwaltung dieser "Portletkonfigurationen" stehen komfortable TMLScript-Funktionen zur Verfügung.

3. WebGate Anywhere Personalisierung

Die Personalisierungskomponente bietet die Möglichkeit, jeden User mit einem persönlichen Profil zu versehen.

Wie entstehen die Daten in den Profildokumenten?

Es gibt grundsätzlich vier Wege, Informationen im Benutzerprofil abzuspeichern:

1. Benutzerdaten über Eingabeformulare

Der einfachste und flexibelste Weg ist es, dem Benutzer ein Eingabeformular im Web zu präsentieren, in dem nach den notwendigen Daten gefragt wird.

Dieser Weg setzt natürlich die "Kooperationsbereitschaft" des Benutzers voraus. In der Regel sollte daher auf der Site die Eingabe der Daten entsprechend motiviert werden.

2. Benutzerdaten über "Aktionen"

In WGA wurde eine Verfahren realisiert, mit dem auf einfache Weise durch Klicken eines Links oder Buttons "Aktionen" gestartet werden können. Diese Aktionen können dafür verwendet werden, bestimmte Informationen im Benutzerprofil zu hinterlegen (Benutzer hat diese Aktion angeklickt). Denkbar ist z. B., mit Hilfe von Aktionen "Ungelesenmarkierungen" im Intranet nachzubilden, indem die Aktion speichert, welches Dokument gelesen wurde.

3. Statistikinformationen

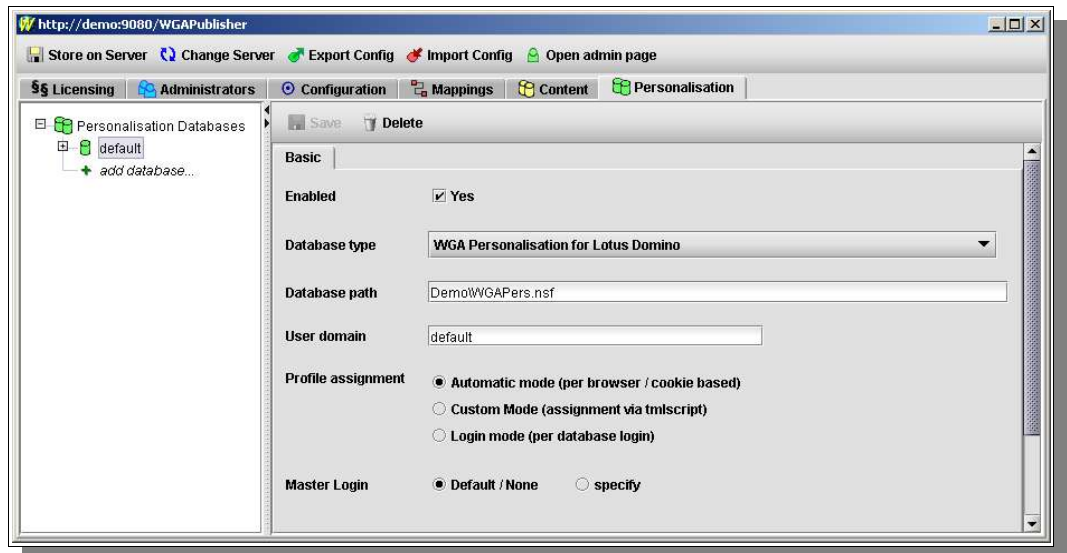
Neben den "freiwilligen" Daten sammelt WGA automatisch bestimmte Statistikinformationen, wie die Anzahl der Sessions, die zuletzt besuchte URL oder die Anzahl Seiten, die der Benutzer besucht hat. Diese Informationen können getrennt oder zusammen mit den "freiwilligen" Daten dazu verwendet werden, User zu "klassifizieren" (dazu später mehr).

4. Zusatzdaten aus Backendsystemen

Sind die Webuser "bekannt", z. B. indem Sie sich per Name oder ID identifiziert haben, können die Profile auch durch beliebige Daten aus Backendsystemen (SAP, Notesanwendungen, etc.) angereichert werden. Dies geschieht über periodische Agenten, die über alle Profildokumente iterieren und Lookups in die gewünschten Backendsysteme ausführen.

3.1 Wo werden Profildaten abgelegt?

In welcher Datenbank Profileinformationen abgelegt werden, kann in der WGA Konfiguration eingestellt werden. Dazu gibt es im WGA-Admin-Client (neu ab WGA 2.1) eine entsprechende Seite:



In WGA 2.1 werden die Profildaten in einer speziellen Lotus Domino Datenbank abgelegt - der Personalisierungsdatenbank. Spätere Versionen von WGA werden die Möglichkeit besitzen, diese Daten alternativ in SQL-Datenbanken abzuspeichern.

3.2 Wie werden Benutzer identifiziert?

In WGA 2.1 gibt es drei Möglichkeiten der Benutzeridentifizierung: "Automatic", "Custom" und "Login". Diese Methoden werden im WGA-Admin-Client zusammen mit der Datenquelle eingestellt.

Bei Einstellung "**Automatic**" erhalten Benutzer automatisch einen Cookie, der fest mit einem Benutzerprofil verbunden ist. Ein entsprechendes Benutzerprofil wird neu angelegt, wenn der Benutzer das erste mal die Site besucht, also noch keinen WGA-Cookie besitzt.

Bei Einstellung "**Custom**" findet keine automatische Zuordnung eines Benutzerprofils statt. Statt dessen ist der Web-Designer selbst dafür verantwortlich, das Benutzerprofil zu "aktivieren". Dazu stehen entsprechende Mechanismen zur Identifizierung und zum Erzeugen neuer Benutzerprofile zur Verfügung, die weiter unten beschrieben sind.

Die Identifizierung des Benutzers geschieht über Benutzername und Passwort, die im Benutzerprofil abgespeichert sind und von WGA überprüft werden.

Der Designer ist hier auch dafür verantwortlich, ggf. ein neues Benutzerprofil neu anzulegen.

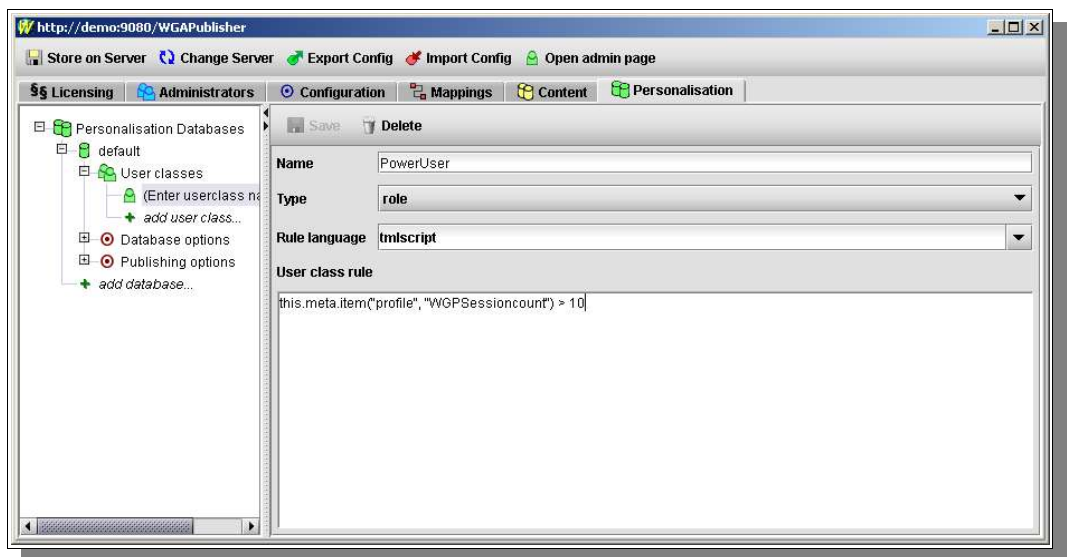
Bei Einstellung "**Login**" erhalten nur diejenigen Benutzer ein Profil, die sich an WGA (und damit an dem dahinterliegenden Datenbanksystem) anmelden. Auch in diesem Fall wird ein neues Profildokument automatisch angelegt, fall der Benutzer

sich das erste mal auf der Site anmeldet.

3.3 Benutzerklassen

In realen personalisierten Websites ist es in der Regel nicht sinnvoll, das Design eines Template von einem konkreten Benutzer anhängig zu machen. Daher können Benutzer zu "Klassen" zusammengefaßt werden. Das Template zeigt dann für alle "Benutzer einer Klasse" entsprechende Informationen an.

Benutzerklassen werden ebenfalls mit dem WGA-Admin-Client angelegt und verwaltet:



Im obigen Beispiel wird eine Benutzerklasse "Poweruser" definiert. Poweruser sind User, die mindestens 10 mal die Site besucht haben.

Die Definition einer Benutzerklasse geschieht über die Eingabe einer "Regel" (englisch "Rule"), d. h. einer Funktion, die True oder False zurückliefert. In diesem Beispiel heißt die Regel: "this.meta.item("profile", "WGPSessioncount") > 10".

Die Regeln werden jedesmal dann ausgewertet, wenn sie "angefragt" werden.

3.3.1 Verwenden von Benutzerregeln

Zur Verwendung von Benutzerregeln wurden in WGA zwei neue Condition-Attribute eingeführt, die in den Tags <tml:if> und <tml:case> verwendet werden können. Das folgende Beispiel zeigt ein Item nur dann an, wenn der Benutzer "Mitglied" der Klassen "Kunden" oder "Mitarbeiter" ist:

```
<tml:case hasanyclass="Kunde,Mitarbeiter">
    <tml:item name="Kundeninformationen"/>
</tml:case>
```

Der Wert des Attributes **hasanyclass** ist eine kommaseparierte Liste von "Benutzerklassen". Die Condition ist "wahr", wenn der augenblickliche User mindestens einer der angegebenen Klassen angehört. Im obigen Beispiel werden die Kundeninformationen also allen Kunden und Mitarbeitern angezeigt.

Das zweite Conditionattribute "**hasallclasses**" fragt ab, ob der Benutzer Mitglied aller angegebenen Benutzerklassen ist. Die Klassen werden durch Komma getrennt eingegeben:

```
<tml:case hasallclasses="Kunde,Poweruser">
    <tml:item name="wichtige_Kundeninformationen_fuer_Powerkunden"/>
</tml:case>
```

In diesem Beispiel wird die Information nur angezeigt, wenn der User sowohl Kunde als auch Poweruser ist.

3.4 Abfragen von Benutzerdaten per TML

Zur Abfrage von Benutzerfeldern wird das Attribut `type="Profile"` im `<tml:item>`-Tag verwendet:

```
<tml:item name="feld" type="profile">
```

3.5 Abfragen von Benutzerdaten per TMLScript

Zur Abfrage von Profildaten per TMLScript existiert das Objekt `this.profile` mit den Methoden `item(feldname)` sowie `itemList(feldname)`:

`this.profile.item(feldname)` liefert das Feld "feldname" des Benutzerprofils zurück.

`this.profile.itemList(feldname)` liefert das Feld "feldname" des Benutzerprofils als Liste zurück.

3.6 Speichern von Profildaten

Das Speichern von Profildaten geschieht grundsätzlich über folgenden TMLScript-Methoden:

```
this.profile.setItem(Feldname, Wert)
```

```
this.profile.save()
```

In Zusammenhang mit `<tml:form>` und "Actions" können auch sämtliche Daten eines Eingabeformulars im Benutzerprofil abgespeichert werden.

Beispiel:

```
<tml:form id="myform" source="profile">
    Name: <tml:input name="name"/><br>
    Alter: <tml:input name="alter"/><br>
```

```
<a href="<tml:url type="action">  
    this.tmlform.storeInProfile()</tml:url">Speichern</a>  
</tml:form>
```

In diesem Beispiel werden die Daten aus dem Eingabeformular in den Feldern "name" und "alter" im Profildokument gespeichert.

3.7 Personalisierungsmode "Custom"

Der WGA Publisher 2.1.1 führt nun einen neuen Personalisierungsmodus namens "Custom" ein. Inhalt dieses Modus ist, dass es keine automatische Zuweisung von Profilen zu Benutzern gibt. Stattdessen wird die Erstellung und Zuweisung von Profilen vom Designer selbst programmiert. Dies geschieht über spezielle Methoden in TMLScript. Damit eröffnet dieser Modus dem Benutzer eine Vielzahl unterschiedlicher Personalisierungsverhalten selbst zu implementieren. Ein benutzerdefinierter Login-Mechanismus (unabhängig von der Datenbank-Anmeldung) ist genauso möglich wie die Profilerstellung auf Basis von Informationen aus Drittsystemen.

3.7.1 Aktivieren des "Custom"-Modus

Um den Custom-Modus der Personalisierung zu nutzen, öffnen sie den WGA Manager und wählen eine Personalisierungsdatenbank an. Dort wählen sie den neuen Modus "Custom" aus der Optionsgruppe verfügbarer Modi aus.

3.7.2 Programmieren von Erstellung und Zuweisung von Profilen

Zur Steuerung des "Custom"-Modus sind im Kern nur zwei neue Methoden in TMLScript notwendig.

this.createUserProfile(String name, String password)

- Erstellt ein neues Benutzerprofil des übergebenen Namens und weist es dem aktuellen Benutzer zu.

this.assignUserProfile(String name, String password)

- Weist ein existierendes Benutzerprofil dem aktuellen Benutzer zu.

Der Name eines Benutzerprofils ist sein eindeutiges Erkennungsmerkmal. Er wird einem neuen Profil direkt mit den Parametern der Methode "createUserProfile" zugeordnet. Es kann keine zwei Profile desselben Namens geben. Würde versucht, ein neues Profil unter einem bereits vergebenen Namen zu erstellen, so würde die Methode "createUserProfile"

dies verweigern und einen Fehlercode ausgeben. Über den Namen wird das Profil auch in der Methode "assignUserProfile" wieder gefunden.

Optional kann einem neuen Profil über den Parameter "password" ein Passwort zugeordnet werden, dessen Kenntnis notwendig ist, um dieses Profil wieder zuordnen zu können. Entsprechend muss dieses Passwort auch passend zum gewünschten Profil wieder in der Methode "assignUserProfile" angegeben werden. Ist dieser Passwortschutz nicht erwünscht, so kann der Passwortparameter sowohl in "createUserProfile" und "assignUserProfile" als null belassen werden.

Die Funktionalität konkret an einem Beispiel: Eine Website soll eine benutzerdefinierte Login-Funktionalität besitzen. Der Benutzer soll sich selbst ein Profil per Webmaske erstellen können mit frei wählbarem Benutzernamen und Passwort. In diesem Profil sollen seine benutzerspezifischen Konfigurationen für die Website gespeichert werden.

Desweiteren soll ihm über eine Loginmaske ermöglicht werden, dieses Profil und seine Daten wiederzuerlangen.

3.7.3 Beispiel: Maske zur Erstellung eines Benutzerprofiles

Hier eine sehr einfache Version der Maske, mit welcher das Profil erstellt werden kann:

```
<tml:action id="createProfile">

var result = this.createUserProfile(this.tmlform.user, this.tmlform.pwd);
if (result > 0) {
    this.setvar("msg", "Could not create profile: " + result);
}
else {
    this.setvar("msg", "Profile " + this.profile.NAME + " successfully created");
}

</tml:action>
<tml:item name="msg"/>

<tml:form id="create">
User: <tml:input name="user"/><br/>
Password: <tml:input name="pwd"/><br/>
<a href="<tml:url action="createProfile"/>">Create</a>
</tml:form>
```

Diese Maske besteht aus einer TML-Aktion, welche das Profil erstellen soll, sowie aus einem TML-Formular über welches der

Benutzer den Namen und das Passwort des neuen Profils angeben kann und welches schliesslich per URL die TML-Aktion ansteuert.

Betrachten wir zunächst das TML-Formular im unteren Block. Hier gibt es die Felder "user" und "pwd", über welche der Besucher die notwendigen Informationen angeben kann. Darunter baut `<tml:url>` einen Link auf, der die TML-Aktion "createProfile" aufruft. Da sich der Tag innerhalb eines Tags `<tml:form>` befindet, wird der Formularinhalt im TMLScript-Code der Aktion verfügbar sein.

In der Aktion wird die Methode "createUserProfile" aufgerufen. Die notwendigen Parameter Name und Kennwort werden direkt dem TML-Formular entnommen, dessen Daten im TMLScript-Objekt "this.tmlform" hinterlegt sind. Die Methode gibt einen Fehlercode zurück, der über Erfolg oder Misserfolg der Operation Auskunft gibt. Ist die Erstellung gelungen, so ist der Code 0 und im Item "msg" wird eine Bestätigung der Erstellung hinterlegt, welche unterhalb der Aktionsdefinition ausgegeben wird. Beachten sie, dass hier direkt auf das Profil verwiesen wird, welches sich wie gewohnt unter "this.profile" in TMLScript befindet. Die Daten des Profils können ab diesem Zeitpunkt auch über die anderen, herkömmlichen Zugriffsmethoden, wie z.B. `<tml:item type="profile"/>` abgerufen werden. Diese Zuweisung gilt nun solange, wie der Browser-Session des Benutzers dauert. Entweder er schliesst seinen Browser und somit diese Session oder die Session erreicht ihre maximale Lebensdauer (welche in der Konfiguration des J2EE-Servers definiert wird).

War die Erstellung nicht erfolgreich, gibt "createUserProfile" einen Fehlercode > 0 aus. Details zu den einzelnen möglichen Fehlern finden sie in der Dokumentation der Methode im WGA Info Center.

3.7.4 Beispiel: Maske zur Zuweisung eines "Custom"-Benutzerprofils als Anmeldung

Nun kann über eine weitere Maske eine Anmeldung realisiert werden, in welcher der Benutzer den Namen seines zuvor erstellten Profils angeben kann, zusammen mit dem vergebenen Passwort, und so wieder an sein Profil erlangt. Auch hier wieder ein simples Beispiel:

```
<tml:action id="getProfile">
var result = this.assignUserProfile(this.tmlform.user, this.tmlform.pwd);
if (result > 0) {
this.setvar("msg", "Could not log in: " + result);
```

```
}  
</tml:action>  
  
<tml:form id="login">  
User: <tml:input name="user"/><br/>  
Password: <tml:input name="pwd"/><br/>  
<a href="<tml:url action="getProfile"/>">Login</a>  
</tml:form>
```

Auch dieses Beispiel besteht wieder aus einem TML-Formular und einer TML-Aktion. Das Formular ist fast identisch zu jenem aus dem vorhergehenden Beispiel, enthält es doch dieselben Felder und einen Aktions-Link. Der Link ruft jetzt jedoch eine Aktion namens "getProfile" auf, die per Methode "assignUserProfile" versucht, ein existierendes Profil zu ermitteln und dieses dem aktuellen Benutzer zuzuordnen. Name und Passwort des gewünschten Formulars werden wieder direkt dem Objekt "this.tmlform" entnommen.

War die Operation erfolgreich, so gibt "assignUserProfile" wieder 0 zurück und unter "this.profile" ist ab sofort das wiedergefundene Profil verfügbar. War sie nicht erfolgreich (existierte kein Profil des angegebenen Namens oder das Passwort war falsch), so wird ein anderer Code >0 zurückgegeben. Details zu den einzelnen möglichen Fehlern finden sie in der Dokumentation der Methode im WGA Info Center.

3.7.5 Weitere WebTML/TMLScript-Funktionalitäten rund um die Personalisierung und den "Custom"- Modus

Um das Vorhandensein eines Profils beim aktuellen Benutzer zu überprüfen (vor der manuellen Zuweisung ist "this.profile == null" und kein Profil verfügbar), gibt es folgendes neues Attribut in Konditionstags:

```
<tml:if hasprofile="true|false">
```

sowie folgende TMLScript-Methode:

```
this.hasprofile() == true|false
```

Um die Zuweisung eines Profils zum Benutzer wieder zu entfernen, existiert folgende TMLScript-Methode:

```
this.closeUserProfile()
```

Danach ist "this.profile == null". Das Profil ist jedoch nicht gelöscht und kann über einen erneuten Aufruf von "assignUserProfile" wieder zugewiesen werden.

Über folgende Methode kann das Passwort eines

zugewiesenen Profiles geändert werden:

this.profile.setPassword(String newPassword)

Damit diese Änderung wirksam wird, muss das Profil jedoch im Anschluss mit "this.profile.save()" gespeichert werden.

Detaillierte Informationen zu diesen Funktionalitäten sind im WGA Info Center unter den jeweiligen Dokumentationsabschnitten zu finden.

4. WebGate Anywhere Portal Services

Die WGA Portal Services sind entwickelt worden, um Websites oder Intranets als "Unternehmensportale" ausbauen zu können.

Das wesentliche Konzept von Portalen ist es, dem Benutzer persönliche Sichten auf Unternehmensdaten und Unternehmensanwendungen zu ermöglichen. Diese "Sichten" werden im Portal in Form von "portlets" eingebunden.

Portlets sind eigenständige Module mit definierter Funktionalität, die im Portal beliebig zusammengestellt werden können.

Portlets besitzen in der Regel benutzerabhängige Konfigurationen, die im Benutzerprofil abgespeichert werden. So kann z. B. in einem CRM-Portlet der augenblicklich ausgewählte Kunde in der (persönlichen) Portletkonfiguration abgespeichert werden.

Für die Verwaltung dieser "Portletkonfigurationen" stehen kompofotable TMLScript-Funktionen zur Verfügung.

4.1 Was sind Portlets?

Portlets bestehen aus einem TML-Bibliothekseintrag sowie einem dem Portlet zugewiesenen Bereich im Benutzerprofil.

Die Funktionalitäten eines Portlets werden also in TML bzw. TMLScript programmiert.

Alle Portletfunktionen stehen über das global verfügbare Objekt **this.portlet** zu Verfügung.

4.2 Registrieren von Portlets

Bevor Portlets verwendet werden können, müssen Sie bei WGA "registriert" werden. Durch diese Registrierung wird dem Portlet ein eigenständiger Set an Variablen im Benutzerprofil zugewiesen.

Zur Registrierung von Portlets wird die Funktion

key=this.portlet.registerPortlet(TML-Name, Portlet-Titel)

verwendet.

Die Funktion erhält als Parameter den Namen eines TML-Bibliothekseintrages sowie eines frei wählbaren Titels.

Der Rückgabewert der Funktion ist ein Schlüssel, über den nach der Registrierung auf das Portlet zugegriffen werden kann. Dieser Schlüssel wird in der Regel in einem Feld des Benutzerprofils abgespeichert, so daß er zukünftig zur Verfügung steht.

Ein Portlet muß also für jeden Benutzer nur einmal registriert werden. Es ist aber durchaus Möglich und sinnvoll, das gleiche Portlet mehrfach zu registrieren. Jedes dieser Portlets besitzt dann zwar die gleiche Funktionalität, jedoch je eine eigene Konfiguration.

4.3 Einfügen von Portlets in das Templatedesign

Ein Portlet wird in ein TML-Template über den `<tml:include>`-Tag eingebunden:

```
<tml:include type="portlet" key="key des Portlets"/>
```

WGA sucht dabei in einer internen Portlettable nach dem angegebenen Portlet-Key und included den TML-Bibliothekseintrag, der bei der Registrierung angegeben wurde.

Beispiel:

```
<tml:script>
if (this.profile.item("myPortlet")== "")
{
key=this.registerPortlet("Kunden", "Meine Kunden");
this.profile.setitem("myPortlet", key);
this.profile.save();
}
</tml:script>
<tml:include type="portlet" key="{this.profile.item('myPortlet')}" />
```

In diesem Beispiel wird zunächst überprüft, ob das Portlet bereits registriert wurde. Ist dies nicht der Fall, wird das Portlet registriert und der Key in dem Feld "myPortlet" im Benutzerprofile gespeichert.

Danach wird das Portlet über `<tml:include>` in das Template eingefügt.

4.4 Zugriff auf die Konfiguration eines Portlets

Innerhalb des TML-Elementes, das die Funktionalität eines Portlets abbildet, kann wahlweise über TML-Tags oder per TMLScript auf Portlet-Daten zugegriffen werden.

Zum Zugriff per TML wird der `<tml:item>` Tag mit dem Attribut `type="portlet"` verwendet:

```
<tml:item type="portlet" name="Feldname"/>
```

Der Zugriff per TMLScript geschieht über das Objekt `this.portlet`:

`this.portlet.item(Feldname)` liefert das Portletfeld als String zurück

`this.portlet.itemList(Feldname)` liefert das Portletfeld als Liste zurück.

`this.portlet.setItem(Feldname, Wert)` speichert einen Wert in einem Feld der Portletkonfiguration

`this.portlet.save()` speichert die Änderung der Portletkonfiguration dauerhaft im Benutzerprofil ab.

Zum Speichern von Formulardaten in einer Portletkonfiguration, steht im Zusammenhang mit TML-Forms und TML-Actions darüber hinaus die Funktion

`this.tmlform.storeInPortlet()`

zur Verfügung. Beispiel:

```
<tml:form id="myportletform" source="portlet">
Feld 1: <tml:input name="feld1"/><br>
<a href="<tml:url type="action">this.tmlform.storeInPortlet();this.setMode("view")
</tml:url">speichern</a>
</tml:form>
```

4.5 Portlet Modes

Jedes Portlet kann in verschiedenen "Modi" angezeigt werden. Welche "Modes" ein Portlet unterstützt, kann frei vom Portletdesigner bestimmt werden.

Typische Beispiele sind `Mode="view"` (zur Anzeige von Informationen) und `Mode="edit"` (zur Konfigurieren eines Portlets).

Zum Setzen eines Portletmodes wird die Funktion

`this.portlet.setMode(mode)`

verwendet. Diese Funktion wird in der Regel innerhalb von TML-Actions verwendet.

Beispiel:

```
<a href="<tml:url type="action">this.portlet.setMode("edit")</tml:url">Portlet
konfigurieren</a>
```

Zum Abfragen des Portletmodes kann entweder die Funktion

`this.portlet.getMode()`

oder die Condition `portletmode="mode"` in den TML-Tags `<tml:if>` und `<tml:case>` verwendet werden.

Beispiel:

```
<tml:select>
  <tml:case portletmode="view">
    Anzeige des Portlets im View-Mode
  </tml:case>
  <tml:case portletmode="edit">
    Anzeige des Portlets im Edit-Mode
  </tml:case>
  <tml:caseelse>
    Fehlermeldung "unbekannter Modus"
  </tml:caseelse>
</tml:select>
```

Bevor der Modus eines Portlets mit `this.portlet.setMode()` gesetzt wurde, ist der "Default-Modus" immer "view".

4.6 Verschachteln von Portlets

Das Konzept von Portlets in WGA ist darauf ausgelegt, daß Portlets ineinander verschachtelt werden können und Portlets damit hierarchisch verwaltet werden.

Jedes Portlet besitzt ein "Parent-Portlets" und in der Regel mehrere "Child-Portlets".

Besonders sinnvoll ist diese Verschachtelung bei "Container-Portlets": Portlets, deren Aufgabe es ist, ein Set von benutzerdefinierten "Anwendungs-Portlets" zu verwalten. Das Container-Portlet ist für die optische Darstellung der Child-Portlets verantwortlich (z. B. zweispaltig, dreispaltig etc.), die Child-Portlets bieten die eigentliche Anwendungslogik.

Für Child-Portlets ist es notwendig, auf die Konfiguration des Parent-Portlets zugreifen zu können. Dazu steht die Funktion

`portlet=this.portlet.getParentPortlet()`

zur Verfügung.

Parent-Portlets können andererseits auch auf die Konfiguration der Child-Portlets (oder eines beliebigen anderen Portlets) über die Funktion

`portlet=this.portlet.getPortletByKey(portlet-key)`

zugreifen.