



WebGate Anywhere 3.1.1

Ergänzungen zum Administrationshandbuch

Copyright © 2005 by:

Innovation Gate GmbH
40880 Ratingen
Halskestr. 19
Tel.: 02102 - 77 160 - 0
www.innovationgate.com
info@innovationgate.com

Apache und Apache Tomcat Copyright © 1999-2004 The Apache Software Foundation
WebSphere Copyright © IBM Corporation 1996, 2003.

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.

Für Schäden, die durch die Benutzung des Handbuches entstehen, kann Innovation Gate leider keine Haftung übernehmen.

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

Version:	3.1.2023
letzte Änderung:	27. Juni 2005
Status:	- Release -

Inhaltsverzeichnis

1. Einleitung.....	3
1.1. Über WebGate Anywhere 3.1 und 3.1.1.....	3
1.2. Über dieses Handbuch.....	3
1.3. Ergänzende Informationen.....	3
2. Spezielles zur aktuellen Version.....	4
2.1. Migration von früheren Versionen.....	4
2.2. Neue Features.....	4
3. Neue Datenbank-Implementierungen.....	7
3.1. JDBC-Database with Enhanced Access.....	7
3.2. QueryableSource.....	10
3.3. WGA Content Store for Oracle.....	11
4. Der WGA Job Manager.....	12
4.1. Einführung und Terminologie.....	12
4.2. Task-Typen.....	12
4.3. Konfiguration des Job Managers.....	15
4.4. Manuelles Starten von Jobs.....	16
5. Sicherheits-Features.....	17
5.1. Allgemein.....	17
5.2. Verschlüsselung von Administrator-Kennwörtern.....	17
5.3. Deaktivierung von Features.....	17
5.4. Festlegung eines Ports für administrative Arbeiten.....	17
6. Weitere neue Features.....	19
6.1. Benutzerdefinierter Titel für Datenbanken.....	19
6.2. Neue DB-Option „MonitorLastChange“.....	19
6.3. Neue WGA-Cache-Funktionen.....	20
6.4. Flexiblere Sprachdoktrin für Navigatoren und Kontextwechsel.....	20
6.5. Die TMLScript-Konsole.....	21

1. Einleitung

1.1. Über WebGate Anywhere 3.1 und 3.1.1

WebGate Anywhere 3 ist die dritte Generation von Enterprise Content Management Systemen aus dem Haus Innovation Gate. Es handelt sich um eine Softwareplattform, basierend auf der Java 2 Enterprise Edition, zum performanten Authoring und Publishing von Web-Content jeglicher Art.

Webgate Anywhere 3.1 ist die erste funktionelle Erweiterung auf der WGA3-Plattform. Sie fügt dieser im Wesentlichen folgende Features hinzu:

- einen frei konfigurierbaren Job Manager
- eine neue komfortable Anbindung an benutzerdefinierte relationale Datenbanken
- Sicherheits-Features
- Mehrsprachiges Browser Interface mit vielen Erweiterungen

Das erste Maintenance-Release 3.1.1, auf welches sich dieses Handbuch bezieht, enthält neben Bugfixes folgende kleinere Erweiterungen:

- Unterstützung für WGA Content Stores auf Oracle Datenbanksystemen ab Version 9.2
- Neue Administrationsfunktionen wie die TMLScript-Konsole und eine Übersicht der aktuell verarbeiteten URL-Requests
- Verbesserte Web-Performance wenn WGA Content Stores mit Nicht-WGA-Clients bearbeitet werden
- Intelligenter WGA-Caches mit Größenbegrenzung
- Eine flexiblere Doktrin bzgl. Sprachauswahl in Navigatoren und WebTML-Kontextwechseln

1.2. Über dieses Handbuch

Dieses Handbuch ist als Ergänzung zum Administrationshandbuch „WGA3 Installation und Administration“ gedacht, welches den Stand von WGA in der Version 3.0.2 reflektierte.

Diese Handbuch wird darauf aufbauend neue Features in WGA 3.1 bzw. WGA 3.1.1 erläutern. Es befindet sich auf dem Stand von WGA 3.1.1 Patch 1.

1.3. Ergänzende Informationen

Basis für die Administration von WGA3 bleibt das Administrationshandbuch „WGA3 Installation und Administration“, welches WGA auf dem Stand von Version 3.0.2 beschreibt.

Ebenfalls für WGA3 ist ein **Autorenhandbuch** erhältlich, welches die Arbeit als Content-Autor mit WGA beleuchtet.

Eine grundsätzliche Erläuterung der Design-Konzepte in WGA finden sie im **WGA 3.1 Designer-Handbuch**. Des weiteren gibt es das **WebTML-Tutorial**, welches Web-Designern, die mit WGA arbeiten, einen einfachen Einstieg in WebTML ermöglicht.

Generell eine umfangreiche Informationsquelle für Designer und Administratoren ist das **WGA Developer Center**, welches sie im Web unter folgender Adresse finden:

<http://dev.innovationgate.de>

Neben umfangreichen Referenzen zu WebTML und TMLScript finden sie hier viele Artikel zu speziellen Themen von WGA sowie ein Entwickler-Forum, in welchem sie konkrete Fragen stellen können.

2. Spezielles zur aktuellen Version

2.1. Migration von früheren Versionen

Folgende Dinge sind bei einer Migration von Version 3.x auf Version 3.1.1 zu beachten:

- Die DB-Option "AnonymousAccessLevel" für SQL Content Stores ist nun wirkungslos. Einzig und allein der Eintrag für "anonymous" in der ACL der Content Store ist nun effektiv und regelt den Zugriff anonymer Benutzer auf eine SQL Content Store. Dieser war aufgrund eines Fehlers in WGA bislang ineffektiv. Vor ein Migration ist die Zugriffsbeschränkung der DB-Option also ggf. in einen ACL-Eintrag umzuschreiben.
- Änderungen in WGA Content Stores die durch Nicht-WGA-Clients vorgenommen werden (z.B. Lotus Notes) werden nun alle im Hintergrund und mit niedriger Priorität verarbeitet. Das führt dazu, dass solche Änderungen erst mit einigen Sekunden Verzögerung aktiv werden. Diese Verzögerung wird größer je stärker der WGA-Server unter Last steht.

Bei einer Migration von früheren Versionen als WGA 3 gelten zusätzlich die Migrationsangaben aus dem WGA3-Administrationshandbuch.

2.2. Neue Features

Im Folgenden werden die neuen Features von WebGate Anywhere 3 aufgelistet.

2.2.1. Neue Datenbank-Implementierungen

Die „**JDBC Database with enhanced Access**“ ist eine Implementierung, welche zur Pflege von über JDBC angebenen relationalen Datenbanken verwendet werden kann. Im Gegensatz zur Implementierung „JDBC Database (Query only)“ (die ebenfalls verfügbar bleibt) ermöglicht sie die schreibende Pflege von Datensätzen per WGAPI (Kap. 3.1).

Die „**QueryableSource**“ ist eine abstrakte Implementierung, die als Basis für sehr einfache Anbindungen an beliebige Drittsysteme verwendet werden kann. Sie basiert auf der „Simple Content Source“, ermöglicht jedoch nur das Ermitteln von Inhaltsdokumenten per Abfrage (Kap. 3.2).

Die „**WGA Content Store for Oracle**“ ist eine spezielle Version der „WGA Content Store for JDBC“ für Datenbanken die auf einem Oracle-Datenbankserver der Version 9.2 oder höher betrieben werden (Kap. 3.3).

2.2.2. Datenbank-Synchronisation und Migration

Datenbank-Synchronisation und Migration wurden stark überarbeitet. Die Dokument-Prüfung der Synchronisation funktioniert nun „inkrementell“, d.h. nach einer initialen Synchronisation werden nur noch die Dokumente verglichen, die sich seit der letzten Synchronisation geändert haben.

Dies kann seit WGA 3.1.1 bei Bedarf durch die Option „Force Full Compare“ im Scheduler-Task „Synchronisation“ deaktiviert werden.

2.2.3. Erweiterungen der Autoren-Schnittstellen

Das Browser Authoring Interface ist nun mehrsprachig und wird je nach konfigurierter Sprache des Browsers in Deutsch oder Englisch dargestellt.

Die Design-Funktionen des Browser Interfaces wurden in einen separaten Bereich, den so genannten „WGA Designer“, migriert. Dieser ist auf der Startseite über eine separate Schaltfläche „Design“ erreichbar.

Im Browser Authoring Interface werden archivierte Inhaltsdokumente nun erst nach Aufforderung angezeigt. Dies führt dazu, dass weniger Inhaltsdokumente insgesamt

geladen werden, was die Gesamt-Geschwindigkeit und den Ressourcen-Verbrauch des Systems verbessert.

Des Weiteren existiert nun eine Such-Funktion im Browser Authoring Interface, welche Content-Dokumente nach verschiedenen Kriterien heranzieht.

2.2.4. Administration

Verschiedene Features, betreffend Sicherheit sowohl von WGA-Installation als auch der WGA-Laufzeit, sind implementiert worden.

- Sicherheitsrelevante Features, wie das Browser Interface und die Administrations-Schnittstellen, können auf einzelnen WGA-Instanzen deaktiviert werden.
- Die Passwörter von WGA-Administratoren werden in der WGA-Konfiguration nun verschlüsselt abgelegt und nicht mehr im WGA Manager angezeigt. Geändert werden sie über eine Schaltfläche „Change password“.
- Die Verwendung der administrativen Schnittstellen (Administrationsseite, WGA Manager) kann auf einen speziellen Port beschränkt werden.

Detaillierte Dokumentation finden sie in Kapitel 5

Der Datenbank-Titel kann im WGA Manager nun beliebig spezifiziert werden. Wird er dies nicht, so wird der Titel aus der Quelldatenbank verwendet (Kap. 6.1).

Eine neue Datenbank-Option „MonitorLastChange“ ermöglicht es, die Überwachung des Änderungs-Datums einer Datenbank abzuschalten und somit überflüssige Cache-Verwürfe zu verhindern (Kap. 6.2).

In WGA 3.1.1 wurden folgende Administrationsfunktionen hinzugefügt:

- Die „TMLScript Console“ ermöglicht die Ad-Hoc-Ausführung von beliebigem TMLScript-Code im Kontext einer Datenbank (Kap. 6.5).
- Die neue Übersichtsseite „Current Requests“ im „Tools“-Bereich der Administrationsseite gibt einen Überblick über die Server-Requests, mit deren Verarbeitung WGA aktuell beschäftigt ist.

2.2.5. Job-Manager

WGA bietet nun eine Möglichkeit, so genannte „Jobs“ zu definieren. Ein Job ist eine Abfolge von einzelnen Aufgaben, die im Hintergrund der Anwendung ausgeführt werden, z.B. Synchronisationen, Ausführung von TMLScript/Java etc. Über den neuen Bereich „Jobs“ auf der Administrationsseite können diese Jobs gestartet werden (Kap. 4).

2.2.6. WebTML

In WGA 3.1.1 wurde die Sprachdoktrin für WebTML-Navigatoren und Kontextwechsel erweitert (Kap).

2.2.7. TMLScript

Die Methode „attachImage“ am TMLForm-Objekt erhält neue Parameter, mit welchen Grafikdateien beim Anhängen an Dokumente nicht nur umbenannt, sondern sogar in ihrer Größe verändert werden können.

Die Methode „writeCSV“ am TMLForm-Objekt erhält einen weiteren Parameter, mit welchem der Spalten-Delimiter der Datei auf andere Zeichen als das Komma „“ gesetzt werden kann.

Die neue Methode „attachmentSize“ am TMLForm-Objekt ermöglicht das Abfragen der Größe einer hochgeladenen Datei, bevor diese weiterverwendet wird.

Die Funktion „sortList“ erhält einen weiteren Parameter, um aufsteigend und absteigend sortieren zu können.

Diese Funktionen sind in der TMLScript-Referenz im WGA Developer Center dokumentiert.

2.2.8. Interne Architektur

Der WGA-Datenbank-Kern verwaltet nun die Anzahl der gleichzeitig geladenen Backend-Dokumente und limitiert automatisch ihre Anzahl, so dass Ressourcen-Probleme durch zu viele parallel geladene Daten verhindert werden.

Des Weiteren wurden die Backend-Zugriffe synchronisiert, um parallele Zugriffe auf Backend-Systeme, welche dieselben Daten anfordern, zu verhindern und somit den Backend-Traffic zu reduzieren.

3. Neue Datenbank-Implementierungen

Dieses Kapitel ist als Erweiterung des Kapitels 6 „Einrichten von Content-Quellen“ des WGA3 Administrations-Handbuches gedacht und erläutert neue Datenbank-Implementierungen.

3.1. JDBC-Database with Enhanced Access

3.1.1. Grundsätzliches

Die „JDBC-Database with Enhanced Access“ ist eine erweiterte Version der „JDBC-Database“, welche in diesem Release in „JDBC-Database (Query only)“ umbenannt wurde (und unter diesem Namen weiterhin zur Verfügung steht). Genauso wie diese, verschafft einem die „Enhanced Access“-Implementierung Zugriff zu einer beliebigen relationalen Datenbank, die per JDBC angebunden ist.

Derart eingebundene Datenbanken können jedoch – im Gegensatz zur alten Implementierung - über die WGAPI modifiziert werden. So ist es möglich, Tabellen-Zeilen zu erstellen, zu modifizieren und auch wieder zu löschen. Details dazu werden im Kapitel „Content-Abstraktion“ erklärt.

Um diesen Implementierungstypen verwenden zu können, muss ihr verwendeter JDBC-Treiber einige Voraussetzungen erfüllen:

- Er muss die JDBC-Spezifikation 2.0 oder höher unterstützen.
- Er muss aktualisierbare ResultSets (mit Einfügezeile) unterstützen.
- Tabellen, deren Daten modifiziert werden sollen müssen einen Primärschlüssel besitzen.

Diese Implementierung kann für alle Fälle genutzt werden, in welchen JDBC-Datenbanken nicht nur angezeigt, sondern auch modifiziert werden sollen.

Die ältere „JDBC-Database (Query only)“ hat weiterhin ihre Daseinsberechtigung, da sie besser geeignet ist, grosse Massen von Datenzeilen performant anzuzeigen. In diesem Punkt ist die neue Implementierung der alten in Geschwindigkeit und Ressourcenverbrauch unterlegen. Es besteht also kein Grund, bereits existierende JDBC-Datenquellen auf die neue Implementierung zu migrieren.

3.1.2. Spezifika

Typname

jdbc/custom/v2

Art der Datenbank

Content-Datenbank

Leistungsumfang der Implementierung

Erweiterte Query-Implementierung.

Feature	Art der Implementierung
Mehrsprachigkeit	Nein. Als einheitliche Sprache wird die der ISO-Sprachcode der J2EE-Plattform verwendet. Optional festlegbar über DB-Option „Language“.
Hierarchisches Browsen	Ja. Die Hierarchie ist jedoch flach und auf Root-Dokumente begrenzt.
Abfragbar	Ja, über die Query-Typen „sql“ und „table“.
Migrierbar in Content Store	Nein
Per BI pflegbar	Nein

Default-Login

Bei Verwendung von Connection Pooling: Default-Anmeldung, wie sie bei der Einrichtung der JDBC-Datenquelle angegeben wurde.

Web-Authentifizierung

Keine. Alle Abfragen werden über das Master-Login abgewickelt.

3.1.3. Vorbereitung des J2EE-Servers

Ob sie die erstellte Datenbank dem J2EE-Server bekannt geben müssen, hängt davon ab, ob sie das interne Connection-Pooling ihres Datenbank-Servers verwenden wollen. Dies ist eine Option, wenn:

a) Ihr Server das J2EE Standard-Connection-Pooling benutzt, bei welchem eine Datenquelle vom Typ „javax.sql.ConnectionPoolingDataSource“ unter einem JNDI-Pfad abgelegt wird.

b) Ihr verwendeter JDBC-Treiber das J2EE Standard-Connection-Pooling unterstützt, indem er eine Implementierung von „javax.sql.ConnectionPoolingDataSource“ zur Verfügung stellt.

Konsultieren sie in diesem Fall das WGA3 Administrationshandbuch für eine Anweisung zum Einrichten einer JDBC-Datenquelle auf ihrem J2EE-Server.

Sie können eine JDBC-Datenbank auch ohne Connection Pooling einrichten. Dabei wird für jede Datenbank-Sitzung eine eigene Verbindung aufgebaut, was jedoch bei starker Frequentierung sehr unperformant sein kann.

3.1.4. Einbinden der Datenbank in WGA

Um eine neue JDBC-Datenbank WGA bekannt zu machen, gehen sie folgendermaßen vor:

- Wechseln sie im WGA Manager zum Register „Content“.
- In der Baumstruktur links wählen sie „add Database“.
- Im erscheinenden Dialog wählen sie „JDBC-Datenbank with enhanced Access“.
- Ein neuer Content-DB-Eintrag wird nun eingerichtet. Folgende Angaben sind im rechten Editorfenster weiter notwendig:
 - Unter „Database-Key“ wählen sie ein identifizierendes Kürzel für diese Datenbank aus alphanumerischen Zeichen (keine Sonder- oder Leerzeichen). Über dieses Kürzel wird der WGA Publisher die Datenbank publizieren.
 - Unter „Database Path“ geben sie ENTWEDER den JNDI-Pfad zur Datenquelle ein (wenn sie das Connection Pooling ihres J2EE-Servers, z.B. Websphere nutzen) ODER den JDBC-Pfad zu Datenbank-Server und Datenbank (wenn sie kein Connection Pooling nutzen wollen).
 - Das Master-Login können sie auf „default“ belassen, wenn sie J2EE Connection Pooling verwenden. In diesem Fall wird das Login verwendet, welches sie bei Einrichtung der Datenquelle eingegeben haben. Anderenfalls geben sie hier ein Datenbank-Login ein, mit welchem der Datenbank-Server vollen Zugriff auf die Content Store erlaubt.
- Klicken sie auf „Save“, um die getroffenen Einstellungen zu bestätigen.
- Wechseln sie zu den DB-Optionen dieser neu erstellten Datenbank. Dort legen sie an:
 - Falls sie kein Connection Pooling verwenden, ist folgende weitere DB-Option notwendig:
Driver := <Klassenname ihres JDBC-Treibers>
- Klicken sie auf „Store on Server“, um die Konfigurationsänderungen an den Server zu übertragen. Die Datenbank wird ohne Neustart von WGA direkt eingebunden.

3.1.5. Abfragetypen

type="sql"

Hiermit fragen sie Daten über den nativen SQL-Dialekt ihrer Plattform ab. Die herangezogenen Daten sind jedoch nicht über die WGAPI editierbar.

type="table: tabellenname"

Über diesen Abfragetypen erhalten sie die Datensätze einer einzelnen Tabelle in einem editierbaren Format. Dabei ist der Text „tabellenname“ durch den tatsächlichen Namen der Tabelle zu ersetzen.

Als Inhalt der Abfrage können sie den WHERE-Bereich (ohne das Schlüsselwort WHERE selbst) einer SQL-Abfrage spezifizieren, welche die zu ermittelten Datensätze einengt, z.B.:

```
<tml:query type="table:kunden">kundennr = 1</tml:query>
```

Aber es sind auch leere Abfragen möglich, um alle Datensätze der Tabelle zu ermitteln.

```
<tml:query type="table:kunden"/>
```

3.1.6. Content-Abstraktion

Die relationale Datenbank wird folgendermaßen auf die WGA-Dokumentstruktur umgelegt:

Eine **Tabelle** wird als **Webseiten-Bereich** ausgewertet. Daher existiert in der WGA-Datenbank für jede Tabelle ein Bereich desselben Namens. Tabellen ohne Primärindex können nicht verwendet werden und werden dementsprechend auch nicht als Bereich angeboten.

Die **Datenzeilen einer Tabelle** werden als **Struktureintrag der obersten Ebene** dargestellt. Es gibt keine Kind-Einträge. Zu jedem Root-Dokument gibt es genau ein Inhaltsdokument, welches die Daten der Datenzeile enthält. Somit können Navigationsfunktionen (Navigatoren, hierarchische Kontextwechsel) auf diese Datenbank angewendet werden. Das Inhaltsdokument wird unter der Standard-Sprache der J2EE-Plattform abgelegt oder unter einer Sprache, die sie per DB-Option „Language“ angeben können.

Die **Spalten der Datenzeile** sind die Items dieses Inhaltsdokumentes. Sie sind per Spaltenname abrufbar.

Neue Zeilen können über die WGAPI per TMLScript erstellt werden und mit Daten versorgt werden:

```
var newContent = this.db('enhancedjdbc').getArea('tabelle1').createContent();
newContent.setItemValue('spalte1', 'xyz');
newContent.save();
```

Ebenso können existierende Datenzeilen modifiziert und gelöscht werden. Dazu müssen diese ENTWEDER über das hierarchische Browsen ODER über den Abfragetypen „table“ herangezogen worden sein.

```
var tabelle1 = this.db('enhancedjdbc').getArea('tabelle1');
var entry1 = tabelle1.getRootEntries().get(0);
var content = entry1.getReleasedContent('de');
content.setItemValue('spalte1', 'abc');
content.save();
```

Einschränkungen und Besonderheiten:

Werden neue Tabellenzeilen mit „createContent“ erstellt, die einen generierten Primärschlüssel (z.B. Auto-Inkrement) besitzen, so kann das erstellte Content-Objekt nach seiner ersten Speicherung nicht mehr zur Modifikation dieses Inhaltes verwendet werden. Stattdessen muss das Inhaltsdokument erneut herangezogen werden.

Wird ein Content-Dokument gelöscht, so ist automatisch auch der zugehörige Struktureintrag gelöscht, da beide dieselbe Tabellenzeile repräsentieren.

3.2. QueryableSource

3.2.1. Grundsätzliches

Hierbei handelt es sich nicht um eine Datenquelle, die direkt zur Verwendung gedacht ist, sondern um eine Schablone, mit welcher Java-Entwickler sehr schnell beliebige Datenquellen in WGA einbinden können, indem sie die abstrakte Implementierungsklasse „de.innovationgate.webgate.api.templates.QueryableSource“ erweitern.

Wenn der Entwickler eine eigene Implementierung auf Basis der „QueryableSource“ erstellt, kann er eine benutzerdefinierte Prozedur festlegen, wie beliebige Daten per Abfrage zu ermitteln und zurückzuliefern sind. Diese eigene Implementierung kann dann als Content-Datenbank in der WGA-Konfiguration eingebunden werden und per Abfrage Daten liefern. Diese Abfrage wird in Form eines <tml:query>-Tags gestellt, der dann intern über die WGAPI an die benutzerdefinierte Prozedur der Implementierung weitergeleitet wird.

Die Prozedur gibt ihre Daten als Liste von Datenobjekten zurück. Jedes Datenobjekt wird zu einem einzelnen Inhaltsdokument abstrahiert. Als Datenobjekte können Standard-JavaBeans oder Implementierungen von java.util.Map verwendet werden.

Eine genauere Einleitung in die Erstellung einer QueryableSource-Subklasse finden sie im WGA-Development-Center.

3.2.2. Spezifika

Typname

custom/queryablesource (Oder ein von der konkreten Implementierung festgelegter Name)

Art der Datenbank

Content-Datenbank

Leistungsumfang der Implementierung

Erweiterte Query-Implementierung.

Feature	Art der Implementierung
Mehrsprachigkeit	Nein. Als einheitliche Sprache wird der ISO-Sprachcode der J2EE-Plattform verwendet. Optional festlegbar über DB-Option „ <i>Language</i> “.
Hierarchisches Browsen	Nein
Abfragbar	Ja
Migrierbar in Content Store	Nein
Per BI pflegbar	Nein

Default-Login

Keines.

Web-Authentifizierung

Keine.

3.2.3. Vorbereitung des J2EE-Servers

Die QueryableSource selbst hat keine Kriterien, die auf dem J2EE-Server erfüllt sein müssen. Diese entstehen erst durch die Erweiterung der Klasse mit speziellen Funktionalitäten.

3.2.4. Einbinden der Datenbank in WGA

- Wechseln sie im WGA Manager zum Register „Content“.
- In der Baumstruktur links wählen sie „add Database“.
- Im erscheinenden Dialog wählen sie einen beliebigen DB-Typen.
- Ein neuer Content-DB-Eintrag wird nun eingerichtet. Folgende Angaben sind im rechten Editorfenster weiter notwendig:
 - Unter „Type“ geben sie den voll qualifizierten Klassennamen ihrer QueryableSource-Erweiterung an.
 - Weitere Eingaben sind von den Bedürfnissen ihrer Erweiterung abhängig.
- Klicken sie auf „Save“, um die getroffenen Einstellungen zu bestätigen.
- Klicken sie auf „Store on Server“, um die Konfigurationsänderungen an den Server zu übertragen. Die Datenbank wird ohne Neustart von WGA direkt eingebunden.

3.2.5. Abfragetypen

Keine. Die QueryableSource-Erweiterung hat keine Möglichkeit, auf die Angabe unterschiedlicher Abfragetypen zu reagieren.

3.2.6. Content-Abstraktion

Die QueryableSource-Implementierung gibt als Ergebnisobjekte von Anfragen entweder Java-Beans oder Maps (mit String-Schlüsselwerten) zurück. Diese werden von der QueryableSource-Architektur in Content-Dokumente umgewandelt.

Diese Content-Dokumente besitzen Items, mit welchen die Daten der Ergebnisobjekte abgefragt werden können:

- Im Fall eines JavaBeans entsprechen die Itemnamen den Namen der Bean-Eigenschaften, z.B. ruft ein Item „value“ die Bean-Methode „getValue“ auf und liefert deren Ergebnis zurück.
- Im Fall einer Map entsprechen die Itemnamen den Schlüsselwerten der Map. Der Wert des Items ist entsprechend der Wert in der Map, der unter diesem Schlüsselwert abgelegt wurde.

3.3. WGA Content Store for Oracle

Die „**WGA Content Store for Oracle**“ ist eine spezielle Version der „WGA Content Store for JDBC“ für Datenbanken die auf einem Oracle-Datenbankserver der Version 9.2 oder höher residieren.

Ihre Konfiguration ist identisch zu jener der „WGA Content Store for JDBC“, welche im WGA3 Administrationshandbuch beschrieben wird.

Für Unterstützung zur Inbetriebnahme von SQL Content Stores dieses Servertypen können sie auch folgendes Dokument im WGA Developer Center konsultieren, welches Details zum Oracle JDBC-Treiber und den notwendigen Einträgen im WGA Manager enthält:

<http://dev.innovationgate.de/WebGatePublisher/wga-developer-center/wgasql>

4. Der WGA Job Manager

4.1. Einführung und Terminologie

In vielen WGA3-Systemen stehen Aufgaben an, die ständiger periodischer Wiederholung bedürfen. Hierzu gehören z.B. die Synchronisation von Content-Datenbanken, welche eine Ziel-Datenbank immer auf dem neuesten Stand hält. Der WGA Job Manager von WGA 3.1 ist der erste Schritt hin zu einer Prozesssteuerung, welche die Definition und Ausführung solcher Aufgaben komfortabel ermöglicht.

In WGA 3.1 ist nur ein manuelles Starten dieser Aufgaben möglich. Im nächsten Feature-Release wird auch das Definieren von Zeitplänen zur automatischen Ausführung der Jobs möglich sein. Daher wird der Job Manager an manchen Stellen auch schon „Scheduler“ genannt.

Kurz beschrieben können für den Job Manager so genannte **Jobs** definiert werden. Diese Jobs bestehen aus einer Liste sequentiell abzuarbeitender **Tasks**. Ein Task ist ein einzelner Arbeitsschritt des Jobs. Es stehen verschiedene Task-Typen zur Verfügung, um unterschiedliche Aufgaben zu vollziehen:

- Synchronisation zweier Content-DBs.
- Migration einer Content-DB in eine weitere.
- Ausführung des TMLScripts in einem Script-Modul einer WGA Content Store.
- Ausführung einer Java-Klasse, welche das Interface `de.innovationgate.wgpublisher.scheduler.TaskImplementation` implementiert.

Diese Task-Typen sind über Parameter steuerbar, welche bei der Anlegung des Tasks mit Werten versorgt werden.

Jeder Task hat ein **Ergebnis**. Dies ist so etwas wie der Rückgabewert einer Prozedur der vom Task selbst gesetzt wird und in späteren Tasks Verwendung finden kann. Momentan wird das Ergebnis von keiner in WGA implementierten Funktion verwendet. Es kann jedoch im Code von TMLScript- und Java-Tasks Verwendung finden.

Als weitere Einheit gibt es die **Job-Optionen**. Dies sind Schlüssel/Wert-Paare, die in beliebiger Anzahl zu einem Job definiert werden können. Diese Job Optionen können ebenso wie die Parameter verwendet werden, um die Ausführung der Tasks zu parametrisieren. Sie besitzen jedoch folgende Vorteile gegenüber den Parametern:

Sie sind frei definierbar. Dies ist insbesondere wichtig für Task-Typen, in denen beliebiger Code, z.B. als TMLScript oder Java-Klasse, ausgeführt werden kann. So kann der Entwickler dieses Codes selbst Optionen einführen, welche das Verhalten seines Codes steuern.

Sie sind zur Laufzeit des Jobs änderbar. Ein TMLScript oder eine Java-Klasse welches als Task in einem Job ausgeführt wird, kann eine Option modifizieren und damit eventuell das Verhalten nachfolgender Tasks ändern, welche diese Option auswerten.

4.2. Task-Typen

Bevor wir an die Vorgehensweise zur Definition von Jobs kommen, sollten die Task-Typen erläutert werden, welche das Ausführungs-Potential von Jobs definieren.

4.2.1. TMLScript

Aufgabe

Dieser Task-Typ führt beliebigen TMLScript-Code aus, der in einem Skriptmodul (früher „CSS/JS-Modul“) hinterlegt wurde. Der Code wird in einer ähnlichen TMLScript-Laufzeit ausgeführt wie Event-Scripts. Dementsprechend sind dieselben Bereiche von TMLScript nutzbar bzw. nicht nutzbar wie in Event-Scripts.

Der Anwendungskontext ist natürlich wie gewohnt über das „this“-Objekt verfügbar. Das Script startet hierbei im Kontext der Datenbank, welche auch das Scriptmodul enthält, ist jedoch noch auf keinem Content-Dokument positioniert (d.h. er liegt auf einem

„Dummy“-Kontext dieser Datenbank).

Alle anderen Datenbanken der WGA-Laufzeit sind ansteuerbar und werden grundsätzlich mit dem Master Login geöffnet. Der Task hat also volle Zugriffsrechte.

```
var anotherdbByContext = this.context („db:anotherdb“);  
var anotherdbByDBMethod = this.db('anotherdb');
```

Zusätzlich erhält die TMLScript-Laufzeit ein Objekt „jobContext“, welches identisch zum JobContext-Objekt von Java-Tasks ist, die im nächsten Kapitel erläutert werden. Über das „jobContext“-Objekt kann der TMLScript-Code z.B. Job-Optionen auslesen und setzen.

```
var anOption = jobContext.getOptions().get („anOption“);  
jobContext.getOptions().put („optionName“, „optionValue“);
```

Parameter

Database containing script module: Die Design-Datenbank, welche das Scriptmodul enthält

Script module name: Der Name des auszuführenden Scriptmoduls

Ergebnis

Das Ergebnis ist der Wert der per „return“-Anweisung zurückgegeben wird. Die Rückgabe eines Ergebnisses ist optional.

4.2.2. Java

Aufgabe

Dieser Task-Typ führt eine Java-Klasse aus, welche das Interface „de.innovationgate.wgpublisher.scheduler.TaskImplementation“ implementiert.

```
public interface TaskImplementation {  
    public void execute(JobContext jobContext) throws JobFailedException;  
}
```

Das Interface besteht aus einer einzigen Methode „execute“, die im Task aufgerufen wird. Der Anwendungskontext wird dabei über das Objekt „jobContext“ als Methodenparameter zur Verfügung gestellt. Über dieses Objekt sind alle Ressourcen der WGA-Laufzeit verfügbar. So verfügt das jobContext-Objekt u.A. über Zugang zu den Job-Optionen.

```
public class JobContext {  
    // Scheduler-Informationen. Noch nicht implementiert  
    public getQuartzContext();  
  
    // Kernobjekt der WGA-Laufzeit  
    public WGACore getWgaCore();  
  
    // Ergebnis dieses Tasks  
    public Object getResult();  
  
    // Setzt das Ergebnis  
    public void setResult(Object object);  
  
    // Ein Log-Objekt zum Ausgeben von Ausführungs-Informationen  
    public Logger getLog();  
  
    // Die Job-Optionen als Map-Objekt  
    public Map getOptions();  
  
    // Liefert das Ergebnis des vorherigen Tasks  
    public Object getPreviousResult();  
}
```

Eine Javadoc-Dokumentation zu diesem Objekt wird im WGA Developer Center zur Verfügung gestellt.

Die implementierende Klasse muss zusätzlich einen Default-Konstruktor ohne Parameter besitzen, damit sie von der WGA-Laufzeit instanziiert werden kann.

Die Instanz der Task-Implementierung wird bereits bei Speicherung der Job-Definition erstellt. Der ausgeführte Task ruft die execute-Methode dieser Instanz auf, um die Verarbeitung zu beginnen. Da wiederholte Ausführungen des Tasks immer wieder dieselbe Instanz verwenden, besteht die Möglichkeit, in den Feldern der Instanz Daten zu speichern, welche mehrere Ausführungen des Tasks überdauern.

Das Objekt wird erst dann verworfen, wenn die WGA-Konfiguration aktualisiert wird oder die WGA-Laufzeit endet.

Damit WGA ihre Implementierung verwenden kann, müssen sie diese in eine JAR-Datei verpacken. Die JAR-Datei können sie dann als Bibliothek im WGA Manager einbinden (Unter dem Hierarchiebaum-Eintrag „Mappings“ im Bereich „Mappings“, Feld „Libraries for element and encoder classes“). Details zu diesem Vorgehen finden sie im WGA3 Administrationshandbuch im Kapitel zum WGA Manager.

Parameter

Implementation class: Der voll qualifizierte Klassenname ihrer Implementierung

Ergebnis

Wird über die Methode `jobContext.setResult()` gesetzt. Das Setzen ist optional.

4.2.3. Synchronisation

Aufgabe

Führt eine Synchronisation zwischen zwei Datenbanken durch. Siehe „Synchronisation“ im WGA3 Administrationshandbuch.

Parameter

Source db: Quell-DB der Synchronisation. Wird keine Quell-DB angegeben, so wird versucht, die Job-Option „sourcedb“ auszulesen.

Target db: Ziel-DB der Synchronisation. Wird keine Quell-DB angegeben, so wird versucht, die Job-Option „targetdb“ auszulesen.

Force full compare: Ist diese Option gesetzt, so führt die Synchronisation bei jeder Ausführung einen kompletten Vergleich der Stände von Quell- und Zieldatenbank durch und ermittelt so die notwendigen Operationen. Ist sie nicht gesetzt, so wird nach der ersten Synchronisation auf den inkrementellen Modus gewechselt, der nur die jeweils neuen Änderungen seit der letzten Synchronisation berücksichtigt.

Ergebnis

keines.

4.2.4. Migration

Aufgabe

Migriert den Datenbestand einer Datenbank komplett in eine andere. Siehe „Migration“ im WGA3 Administrationshandbuch.

Parameter

Source db: Quell-DB der Migration. Wird keine Quell-DB angegeben, so wird versucht, die Job-Option „sourcedb“ auszulesen.

Target db: Ziel-DB der Migration. Wird keine Quell-DB angegeben, so wird versucht, die Job-Option „targetdb“ auszulesen.

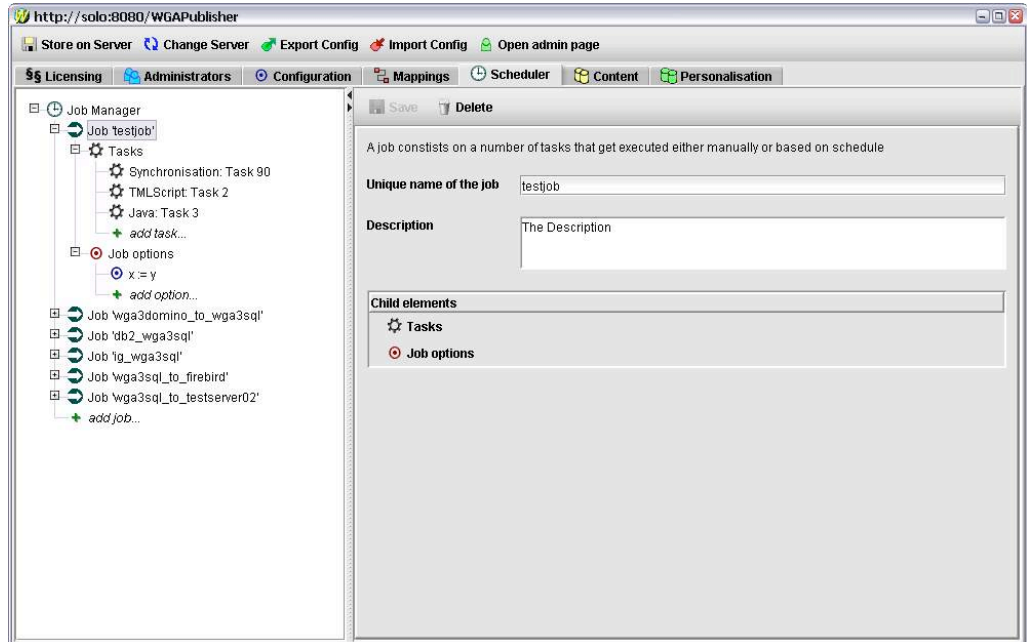
Ergebnis

keines.

4.3. Konfiguration des Job Managers

4.3.1. Definieren von Jobs

Jobs werden im WGA Manager definiert. Dort existiert in WGA 3.1 ein neuer Bereich „Scheduler“.



Gehen sie wie folgt vor, um einen neuen Job anzulegen:

- Klicken sie im Hierarchie-Baum auf „add job...“
- Vergeben sie im nun angezeigten Editor einen eindeutigen Namen für den Job. Legen sie ggf. eine Beschreibung fest. Diese Beschreibung wird in der Oberfläche zum Starten von Jobs angezeigt.
- Klicken sie auf „save“.
- Gehen sie wie folgt vor, um Tasks für ihren neuen Job zu definieren:
 - Klicken sie in den „Child Elements“ auf „Tasks“, dann auf die Aktion „add Task“.
 - Wählen sie einen Task-Typen.
 - Geben sie in den zum Task angezeigten Eingabefeldern die Parameter des Tasks ein.
 - Klicken sie auf „save“.
- Gehen sie wie folgt vor, um Job-Optionen für ihren neuen Job zu definieren:
 - Klicken sie in den „Child Elements“ auf „Job Options“, dann auf die Aktion „add Option“.
 - Geben sie in den Eingabefeldern Name und Wert der Option ein.
 - Klicken sie auf „save“.

Nachdem sie ihre Jobs definiert haben, können sie die gesamte WGA-Konfiguration per „Store on Server“ an den Server übertragen. Die neuen Jobs sind sofort ohne WGA-Neustart verfügbar.

4.3.2. Weitere Konfigurationsoptionen

Zum Hierarchie-Eintrag „Job-Manager“ des Bereichs „Scheduler“ existiert das Feld **Logging Directory**. Wird hier ein Verzeichnispfad eingegeben, so werden die Joblogs in diesem Verzeichnis permanent als Textdatei gespeichert und stehen für spätere Analysen zur Verfügung. Um Namenskollisionen zwischen den Dateinamen zu vermeiden, wird

jeder Log-Dateiname aus Jobname, Job-Laufzeit und einer laufenden Nummer gebildet.

4.4. Manuelles Starten von Jobs

Die Administrationsseite von WGA 3.1 hat einen neuen Bereich „Jobs“ erhalten.

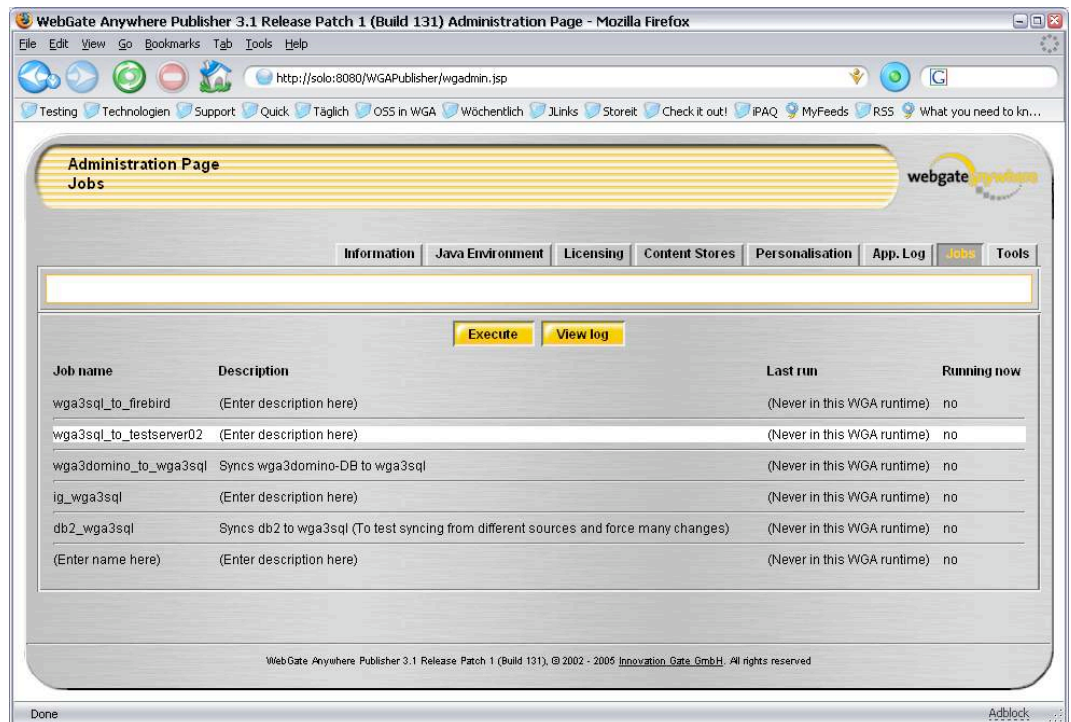


Abbildung 1 Der Bereich "Jobs" auf der Administrationsseite

Hier werden die definierten Jobs aufgelistet und mit ihrer Beschreibung dargestellt. Des Weiteren wird unter **Last run** die letzte Laufzeit des Jobs dokumentiert. Die Spalte **Running now** deklariert, ob ein Job gerade im Moment läuft.

Um einen Job zu starten, markieren sie ihn auf dieser Seite (er wird dann weiß hervorgehoben) und klicken sie auf die Schaltfläche „Execute“. Bestätigen sie die folgende Sicherheitsabfrage.

Nun wird ein zusätzliches Fenster geöffnet, in welchem das Log des gerade gestarteten Jobs angezeigt wird. Die Anzeige wird ständig aktualisiert, um immer die aktuellsten Einträge zu zeigen, bis der Job durchgelaufen ist.

Jobs laufen grundsätzlich im Hintergrund und sind nach Start nicht mehr von der Administrationsseite abhängig. Sie können die Seite trotz laufendem Job schließen. Wenn sie später die Seite wieder öffnen und der Job immer noch läuft, können sie über die Schaltfläche **View log** die zuvor geschlossene Anzeige wieder öffnen (nachdem sie diesen Job erneut markiert haben). Ist der Job durchgelaufen, so erhalten sie über **View log** die Gesamtanzeige des Logs für den zuletzt durchgeführten Lauf.

5. Sicherheits-Features

5.1. Allgemein

Für WGA 3.1 wurde spezielles Augenmerk auf Sicherheits-Features gelegt, welche die Sicherheit des Systems erhöht. Dazu gehören sowohl Verschlüsselungen als auch gezielte Deaktivierung von Features, die auf Live-Systemen nicht benötigt werden und nur als Angriffspunkt missbraucht werden könnten.

5.2. Verschlüsselung von Administrator-Kennwörtern.

WGA 3.1 verschlüsselt abgelegte Kennwörter von WGA-Administratoren mit einem Hashing-Verfahren. Bei Neuinstallationen ist das Kennwort des Default-Administrators bereits verschlüsselt.

Um auf existierenden Systemen in den Genuss dieses Features zu kommen, müssen die Kennwörter der vorhandenen Administratoren über den WGA Manager neu gesetzt werden. Wenn sie den Bereich „Administrators“ im WGA Manager für WGA 3.1 sehen, werden sie feststellen, dass die Kennwörter der Administratoren nicht mehr angezeigt werden.

Stattdessen wird eine Schaltfläche „Set password...“ angeboten, um das Passwort neu zu definieren. Nehmen sie dies vor, um für existierende Administratoren eine verschlüsselte Passwortablage zu erreichen.

5.3. Deaktivierung von Features

Im Bereich „Configuration“ des WGA Managers existiert ein neuer Hierarchie-Eintrag „Feature Activation“. Über diesen Eintrag können sie für bestimmte Systeme Features deaktivieren, die dort missbraucht werden könnten. Die deaktivierbaren Features sind:

- Das WGA Browser Interface.
- Die Administrationsseite.
- Der Download des WGA Managers (Verbindungen mit existierenden WGA Managern sind nach wie vor möglich um das System weiter konfigurieren zu können).
- Die Startseite.
- Das WebDAV-Interface.

Um die jeweiligen Features zu deaktivieren bzw. wieder zu aktivieren, betätigen sie die Auswahlfelder im Dialog, den sie über den „Feature Activation“-Eintrag erreichen. Die Einstellungen werden nach WGA-Neustart aktiv.

5.4. Festlegung eines Ports für administrative Arbeiten

Ebenfalls in diesem Bereich befindet sich ein Eintrag **Restrict administrative tools to port**. Wird in diesem Eintrag eine Portnummer eingetragen, so werden (nach WGA-Neustart) folgende Features nur noch auf dem angegebenen Port unterstützt:

- Aufruf der Administrationsseite per Browser
- WGA Manager-Verbindungen mit dem WGA-Server

Seien sie vorsichtig bei der Aktivierung dieses Features, um sich nicht selbst von der Administration des WGA-Servers auszusperrern. Überprüfen sie in jedem Fall zuvor die Verfügbarkeit der administrativen Features auf dem festzulegenden Port!

Dieses Feature ermöglicht die Restriktion administrativer Tätigkeiten, z.B. auf die Benutzer eines Intranets. Denkbar ist z.B. folgendes Szenario:

- Der J2EE-Server wird darauf konfiguriert, WGA über zwei Ports zu publizieren. 80 (für Webzugriffe) und 8080 (für administrative Arbeiten).
- Eine Firewall blockiert den Zugriff auf den J2EE-Server per Port 8080 und lässt nur Port 80 zu. Alle Benutzer jenseits der Firewall haben keinen Zugriff auf WGA-Features über den Port 8080.

- Per WGA Manager wird WGA angewiesen, administrative Features nur noch auf Port 8080 anzubieten. WGA-Manager-Zugriff und die Benutzung der Administrationsseite ist jetzt faktisch nur noch diesseits der Firewall möglich.

6. Weitere neue Features

6.1. Benutzerdefinierter Titel für Datenbanken

Vor WGA 3.1 wurde der Datenbank-Titel grundsätzlich aus dem Datenbank-Backend ermittelt. Da nicht alle Datenbank-Typen auswertbare Titel besitzen (z.B. JDBC-Datenbanken), wurden statt dessen technische Informationen über Ort und Typ der Datenbank angezeigt.

Dies ist jedoch nicht wirklich der Sinn des Datenbank-Titels, der weniger über den technischen Hintergrund der Datenbank informieren sollte (was gerade bei Einsatz von WGA in den Hintergrund treten sollte), sondern über ihren Nutzen und die enthaltenen Daten.

Daher bietet WGA 3.1 die Möglichkeit im WGA Manager einen benutzerdefinierten Titel zu vergeben.



Abbildung 2: Festlegung des Titels im WGA-Manager

Dazu navigieren sie im WGA-Manager zu einer Content-Datenbank im Bereich „Content“. Hier finden sie ein neues Feld „Titel“. Geben sie hier einen Titel ein, den sie WGA-weit verwenden wollen. Nach Reconnect der Datenbank ist dieser Titel aktiv.

Die Eingabe des Titels ist jedoch optional. Lassen sie den Datenbank-Titel leer, um das alte Verhalten zu belassen.

6.2. Neue DB-Option „MonitorLastChange“

Das letzte Änderungsdatum einer Datenbank hat in WGA eine besondere Bedeutung, da davon die Lebensdauer der diversen Caches abhängig ist. Werden die Daten einer Datenbank über ein WGA-externes Tool (z.B. Lotus Notes) verändert, so wird dieses Änderungsdatum bei jeder Speicherungs-Operation neu gesetzt. Darauf reagiert WGA mit der Verwerfung umfangreicher CACHEDATEN.

Je öfter sich dieses Datum ändert, desto kürzer und damit ineffektiver ist die Lebensdauer des Caches, da die Daten öfter aus dem Backend-System neu ermittelt werden müssen. Je nach dessen Geschwindigkeit bzw. der Geschwindigkeit des eingesetzten Kommunikations-Protokolls, ist das mehr oder weniger kritisch. Daher wird für WGA empfohlen, Autoren- und Publizierungssysteme zu trennen und nur bei Bedarf zu synchronisieren, um die Änderungshäufigkeit für letzteres System klein zu halten.

Anders sieht die Sache aus, wenn nur WGA-eigene Autorenschnittstellen wie z.B. das Browser-Interface benutzt werden, um die Datenbanken zu pflegen. Da hier WGA selbst die Änderung an den Daten vornimmt, kann es seine Caches selektiver und somit intelligenter pflegen. Die Überwachung des Änderungsdatums ist in diesem Fall nicht notwendig, bzw. kann nur stören. Schließlich gibt es über das Ändern von Content-Daten hinaus noch andere Prozeduren, welche das Änderungsdatum verwerfen, die jedoch nicht zu einem Cache-Verwurf führen sollten (z.B. das Ändern der ACL).

Daher führt WGA 3.1 eine neue DB-Option „MonitorLastChange“ ein, welche für alle Datenbank-Implementierungen verwendet werden kann. Wird diese Option auf den Wert „false“ gesetzt, so wird die Überwachung des Änderungsdatum – und somit auch die Möglichkeit eines umfangreichen Cache-Verwurfs – unterbunden. Dies kann für

Datenbanken mit einem langsamen Zugriff signifikante Performance-Steigerungen bedeuten.

6.3. Neue WGA-Cache-Funktionen

Ab WGA 3.1.1 besitzt der WGA-Cache für Item- und Metadatenwerte folgende neue Funktionalität:

Die maximale Größe des WGA-Caches im Hauptspeicher ist konfigurierbar und wird standardmäßig auf 2000 Cache-Einträge begrenzt. Ein Eintrag ist dabei in der Lage alle Metadaten-Werte eines beliebigen Dokumentes oder alle Item-Werte eines Inhaltsdokumentes zu speichern. Ein Content-Dokument belegt also, da es Metadaten und Items beinhaltet, zwei Cache-Einträge. Alle anderen Dokumente belegen lediglich einen.

Grundsätzlich wird WGA die Items bzw. Metadaten zu jedem geladenen Dokument cachen, wodurch mit jedem neuen Dokument die Anzahl an verwendeten Cache-Einträgen steigt. Ist die Maximalgröße erreicht, so wird der Eintrag, welcher *am Längsten nicht verwendet wurde* aus dem Cache entfernt. Werden irgendwann die Daten wieder benötigt welche in diesem Cacheeintrag gespeichert wurden, so müssen sie erneut aus der Datenbank ermittelt werden.

Die maximale Größe des Caches kann über die DB-Option „*MemoryCacheSize*“ pro Datenbank konfiguriert werden:

```
MemoryCacheSize := 10000
```

Diese Einstellung ermöglicht es beispielsweise der betroffenen Datenbank, 10.000 Einträge parallel im Speicher zu halten.

Resultat dieses neuen Verhaltens ist: Der maximale Hauptspeicherverbrauch von WGA wird reduziert und kann durch gezieltes Setzen von *MemoryCacheSize* an die vorhandene Größe des Java-Heaps angepasst werden. Dadurch wird ein Einsatz auf vergleichsweise hauptspeicherarmen Systemen möglich.

Auf Systemen mit einem großen Java-Heap kann es vorkommen, dass WGA durch die Begrenzung des Hauptspeicher-Caches diesen nicht mehr effektiv nutzt, was sich negativ in der Performance niederschlägt. In diesem Fall ist ein Hochsetzen von *MemoryCacheSize* empfehlenswert, damit mehr Einträge im Hauptspeicher erlaubt werden.

6.4. Flexiblere Sprachdoktrin für Navigatoren und Kontextwechsel

Eine Problematik von Navigatoren in WGA war bislang die Tatsache, dass sie keine Dokumente anzeigten, die nicht in der bevorzugten Sprache des Webbenutzers vorliegen. In vielen Situationen ist dies jedoch erwünscht und andere Sprachen könnten herangezogen werden.

Ab WGA 3.1.1 können Navigatoren dann, wenn ein Dokument nicht in der bevorzugten Sprache vorliegt, folgende anderen Sprachen verwenden:

- Die Sprache des aktuell dargestellten Dokumentes

Beispiel: Ein Benutzer hat die bevorzugte Sprache „Deutsch“. Momentan wird ihm ein Dokument in der Sprache „Englisch“ angezeigt. Der Navigator dieser Seite verwendet normalerweise die bevorzugte Sprache um die Titel der Hauptdokumente anzuzeigen. Stößt dieser auf ein Dokument welches nicht in deutsch, jedoch in englisch vorliegt, so kann er nun die englische Version anzeigen, weil dies die Sprache des aktuell angezeigten Dokumentes ist.

- Die Default-Sprache der Datenbank (wenn das neue Attribut `allowdefaultlang="true"` im Navigator-Tag gesetzt wurde)

Beispiel: Eine Datenbank besitzt die Default-Sprache „Englisch“, weil in dieser Sprache die meisten Dokumente vorliegen oder weil angenommen wird, dass dies eine Sprache ist welche die meisten Besucher lesen können. Ein Benutzer hat die bevorzugte Sprache „Deutsch“ und liest ein deutsches Dokument. Der zugehörige Navigator kann nun auch solche Dokumente anzeigen, die nicht in Deutsch jedoch in Englisch vorliegen um ein komplettes Bild der Website und ihres Inhaltskataloges zu vermitteln.

Eine ähnliche Funktionalität wurde bei normalen WebTML-Kontextwechseln per context-Attribut implementiert. Wird per context-Ausdruck auf einen anderen Kontext gewechselt ohne explizit die Sprache zu bestimmen, so wählt WGA die Sprachversion des Ziels nach folgender Priorität:

1. Die bevorzugte Sprache des Benutzers
2. Die Sprache des momentan angezeigten Dokumentes

Anders als Navigatoren besitzen WebTML-Kontextwechsel nicht die Möglichkeit, auf die Default-Sprache der Datenbank zurückzugreifen.

6.5. Die TMLScript-Konsole

Die TMLScript-Konsole ist ein neues Administrationswerkzeug, welches dem Benutzer ermöglicht, beliebige TMLScript-Ausdrücke auszuführen und deren Ergebnis zu ermitteln. Dies kann einerseits für den WebTML-Designer hilfreich sein um das Verhalten eines für Scripts zu überprüfen, welches für eine Webseite geschrieben wurde. Andererseits kann der Administrator hier vielseitig in das System eingreifen, indem er auf WGAPI-Funktionen zurückgreift.

Um die TMLScript-Konsole aufzurufen wählen sie auf der Administrationsseite den Register „Content Stores“. Wählen sie aus der Liste die Datenbank, in deren Kontext sie TMLScript ausführen wollen und klicken sie dann auf die Schaltfläche „TMLScript Console“.

Um einen TMLScript-Ausdruck auszuführen geben sie wie folgt vor:

- Wählen sie im Feld „Context“ einen Kontextausdruck, genauso wie sie im context-Attribut von WebTML-Tags verwendet werden. Ihr TMLScript-Ausdruck wird in diesem Kontext ausgeführt. Wenn sie das Feld leer lassen wird der Ausdruck kontextlos ausgeführt.
- Geben sie ihren auszuwertenden Ausdruck im Feld „TMLScript-Code to execute“ an. Dies kann entweder eine TMLScript-Expression sein, wie sie in z.B. in Konditionen von <tml:case>-Tags zum Einsatz kommen oder ein komplettes Skript inklusive Steueranweisungen und Rückgabe per return-Anweisung, wie sie es in <tml:script>-Tag verwenden.
- Geben sie bei „Script-Type“ an, ob sie eine Expression oder ein Script verwenden
- Klicken sie auf „Execute“

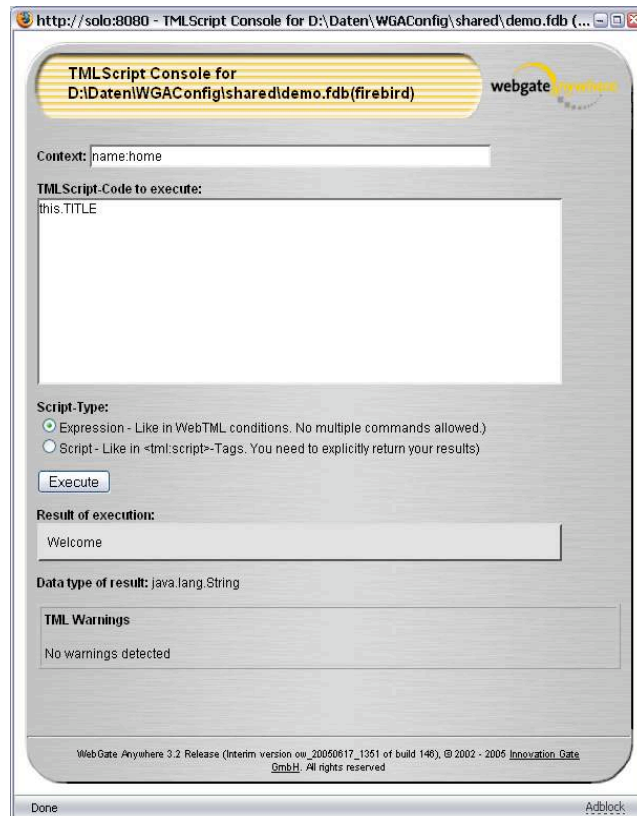


Abbildung 3: Die TMLScript-Konsole

Ihr Script-Ausdruck wird nun ausgeführt. Das Feld „Result of Execution“ wird ggf. das Ergebnis ihres Ausdruckes ausgeben. Dies ist bei TMLScript-Expressions das Ergebnis des Ausdruckes, bzw. bei Scripts der Wert welcher per return-Anweisung zurückgegeben wurde.

In der Liste der WebTML-Warnings können sie überprüfen ob ihr Ausdruck Warnungen produziert hat.