Kernsoftware

Tutorial

Version:	**0.5**
Stand:	**09.08.2006**
Autor:	**beck et al. projects GmbH**
Produktzustand:	**Vorgelegt**
Produktname:	**VRZ3 Kernsoftware Tutorial**

Projektleiter:	**Herr Dr. Pfeifle**
Projektträger:	**Regierungspräsidium Tübingen** **Landesstelle für Straßentechnik** **Heilbronner Str. 300-302** **70469 Stuttgart**
Ansprechpartner:	**Frau Dempe**

Kernsoftware: Tutorial

beck et al. projects GmbH
Martin Hilgers
Thomas Müller
Philippe Schrettenbrunner

Änderungsübersicht

Version	Datum	Stand
Version 0.1	03.07.2006	Schnelleinstieg vorgelegt
Version 0.2	11.07.2006	Einarbeitung Anmerkungen LST
Version 0.3	18.07.2006	Abnahme Schnelleinstieg
Version 0.4	21.07.2006	2. Meilenstein: 50% fertiggestellt
Version 0.5	09.08.2006	Tutorial vorgelegt

Kernsoftware: Tutorial

beck et al. projects GmbH
Martin Hilgers
Thomas Müller
Philippe Schrettenbrunner

Zusammenfassung

Das Ziel des Kernsoftware-Tutorials ist den Einstieg in die Kernsoftware zu vereinfachen. Dazu ist das Tutorial in drei Teile gegliedert.

Das erste Kapitel *Schnelleinstieg* gibt einen Überblick über die Kernsoftware. Neben der Installation der Kernsoftware-Auslieferung und einigen theoretischen Grundlagen wird in diesem Kapitel kurz auf die Verwendung der Kernsoftware-API eingegangen. Vorkenntnisse werden nicht benötigt — mit Ausnahme des Abschnitts über das Java Kernsoftware-API.

Im folgenden Kapitel, *Grundlagen der Kernsoftware*, werden die Konzepte der Kernsoftware erläutert. Insbesondere wird auf die Funktionen des *Datenverteilers*, der *Konfiguration* und der *Parametrierung* eingegangen. Dieses Kapitel wendet sich hauptsächlich an Systemadministratoren und Softwareentwickler.

Im letzten Kapitel, *Kernsoftware-Praxis* stehen sowohl die Programmierung von Kernsoftware-Applikationen, als auch Änderungen am Datenmodell der Kernsoftware und die Konfiguration der mitgelieferten Skripte im Vordergrund. Für den Abschnitt über das Kernsoftware-API werden fundierte Kenntnisse einer objektorientierten Programmiersprache, vorzugsweise Java, und XML benötigt. Ein weiterer Schwerpunkt des Kapitels liegt in der Beschreibung der mit der Kernsoftware ausgelieferten Tools.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API
Application Programming Interface

GTM
Generischer Testmonitor

ID
Identifikationsnummer

JVM
Java Virtual Machine

PID
Permanente Identifikation

SWE
Softwareeinheit

XML
Extensible Markup Language

Kapitel 1 Schnelleinstieg

Dieses Kapitel beschreibt grundlegende Eigenschaften der Kernsoftware.

Nach Installation und Start der Kernsoftware werden die zum Arbeiten mit der Kernsoftware notwendigen Konzepte erklärt. Anschließend wird der Generische Testmonitor (GTM) vorgestellt, und es folgen erste Programmierbeispiele.

1.1 Installation Kernsoftware

Um die Kernsoftware zu installieren muss das gelieferte oder von Verkehrsrechnerzentralen des Bundes[1] (Zugangsdaten sind erforderlich) heruntergeladene ZIP-Archiv in ein beliebiges Verzeichnis entpackt werden.

Linux:

Wird ein Linux-Betriebssytem verwendet, kann dies durch den Befehl

```
unzip kernsoftware-system-xx-xx-xxxx.zip
```

geschehen, wobei `xx-xx-xxxx` für die Version der Kernsoftware steht.

Windows:

Mit einem Windows-Betriebssystem kann die Kernsoftware ab Windows XP mit dem windowseigenen Packer extrahiert werden. Dazu genügt ein Doppelklick auf das Archiv und anschließend ein Klick auf *"Alle Dateien entpacken..."* oder *"Extract all files".*

Wird ein Windows-Betriebssystem in einer älteren Version als Windows XP verwendet, so muss zum Extrahieren des Kernsoftware-Archivs ein externer Packer verwendet werden (z.B. Filzip[2]).

1.2 Wichtige Bestandteile der Installation

Sobald das Archiv in ein Verzeichnis entpackt wurde, findet sich dort die folgende Verzeichnisstruktur[3]:

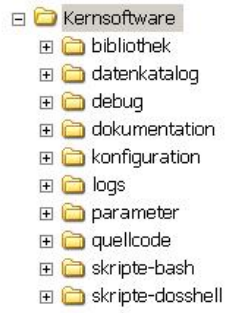


Abbildung 1-1: Verzeichnisstruktur der Kernsoftwareinstallation

Es folgt eine Auflistung der wichtigsten Verzeichnisse und ihres Inhalts:

[1] http://zid.almo-traffic.de/
[2] http://www.filzip.com/de/download.html
[3] Der Ordner *debug* wird erst bei Bedarf (im Normalfall beim ersten Start der Kernsoftware) erstellt, ist also nicht von Anfang an vorhanden.

- *bibliothek*

  Alle für den Betrieb notwendigen Libraries sind hier als JAR-Archive abgelegt.

- *datenkatalog*

  Der XML-Datenkatalog, der das Datenmodell der Kernsoftware festlegt, ist hier in einem XML-Format zu finden. In dieser Form ist der Datenkatalog jedoch nicht geeignet, um von Benutzern eingesehen zu werden. Stattdessen kann der HTML-Datenkatalog von der VRZ3 Portal Webseite[4] heruntergeladen werden. Dieser enthält alle Informationen über das Datenmodell in übersichtlicher Form. Weitere Informationen zum Datenkatalog sind in Abschnitt 2.1.3.1 zu finden.

- *debug*

  Abhängig vom eingestellten *Debug*-Modus sind hier Logdateien der Kernsoftware einzusehen. Die Dateien liegen im Klartextformat vor, können also prinzipiell mit jedem beliebigen Texteditor eingesehen werden.

  Die aktuellste Variante ist immer mit `xxx-0-0.log.yyy` bezeichnet, wobei `xxx` für den Namen der Applikation und `yyy` für das konfigurierte Ausgabeformat steht.

- *dokumentation*

  Hier findet sich die Dokumentation zu den zur Kernsoftware gehörigen Komponenten, u.a. auch die Javadoc-Dokumentation der Kernsoftware.

- *konfiguration*

  Das Datenmodell, das von der Kernsoftware verwendet wird, befindet sich in Form einer Datei, der `config.xml`, in diesem Verzeichnis. Änderungen an dieser Datei beeinträchtigen das Systemverhalten, daher sollte vorher Abschnitt 2.1.3.1 gelesen werden.

- *parameter*

  Für Daten, die während der Laufzeit geändert werden und persistent gehalten werden sollen (Parametrierung) wird hier jeweils eine Datei angelegt. Diese hält dann, sobald der entsprechende Parameter gesetzt wird, die getätigten Einstellungen auch nach einem Neustart des Systems vor.

- *quellcode*

  Quelltexte der Kernsoftware finden sich hier in den entsprechenden ZIP-Archiven.

- *skripte-bash* und *skripte-dosshell*

  Diese Verzeichnisse enthalten die Linux- bzw. Windows-Shellskripte zum

  - Starten der Kernsoftware (Abschnitt 1.3),
  - Starten des Generischen Testmonitors (Abschnitt 1.5),
  - Konvertieren des Datenkatalogs (Abschnitt 2.1.3.2),
  - Browsen des Datenmodells (siehe Abschnitt 3.2.3).

1.3 Starten und Beenden der Kernsoftware

Wird die Kernsoftware gestartet, so ergibt sich folgende Minimalkonfiguration:

[4] http://zid.almo-traffic.de/index.php?id=30

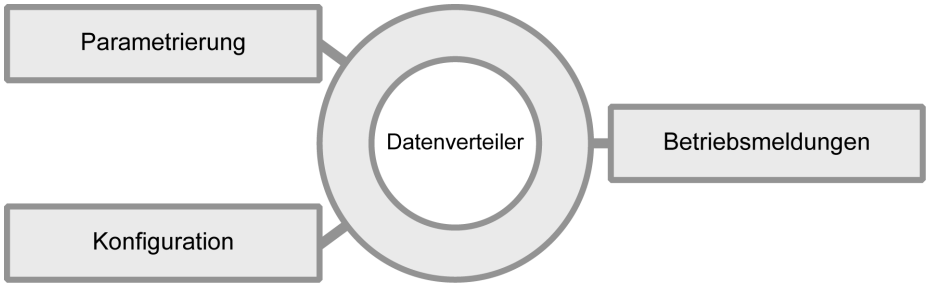


Abbildung 1-2: Komponenten der Kernsoftware nach dem Start

Die Softwareeinheiten (SWE) *Parametrierung*, *Datenverteiler*, *Betriebsmeldungen* und *Konfiguration* werden gestartet. Ein vollständiger Überblick über die Komponenten der Kernsoftware ist in Abschnitt 1.4.1 zu finden.

1.3.1 Starten und Beenden unter Windows

- *Starten*

  Im Verzeichnis der Kernsoftware befindet sich der Ordner *skripte-dosshell*. Durch einen Doppelklick auf die Batch-Datei `KernsoftwareSystem.bat` kann die Kernsoftware gestartet werden.

- *Beenden*

  Durch Schließen des Batch-Fensters wird die Kernsoftware gestoppt.

1.3.2 Starten und Beenden auf Linux-Rechnern

- *Starten*

  Im Verzeichnis der Kernsoftware befindet sich der Ordner `skripte-bash`. Dort ist die ausführbare Datei `KernsoftwareSystem.sh` zu finden. Durch Ausführen des Skriptes wird die Kernsoftware gestartet.

- *Beenden*

  Die Kernsoftware läuft als Hintergrundprozess. Zum Stoppen können die vier Prozess-Ids der Kernsoftware mit dem Befehl `ps aux` ermittelt werden. Die Prozesse der Kernsoftware können an der Zeichenkette `bibliothek/runtime.jar` erkannt werden. Anschließend müssen die Prozesse mit dem Befehl `kill <pid1> <pid2> <pid3> <pid4>` gestoppt werden.

1.4 Grundlagen Kernsoftware

Die Kernsoftware wurde als Basissystem für Verkehrsrechnerzentralen entwickelt. Sie bietet Applikationen eine einheitliche, offene Schnittstelle, über die Daten über ein Netz ausgetauscht werden können. Einzellösungen und geschlossene Gesamtsysteme sollen künftig durch die hersteller- und plattformunabhängige Kernsoftware abgelöst werden, um die Systemlandschaft zu vereinheitlichen. Die folgende Abbildung zeigt exemplarisch eine Verkehrsrechnerzentrale mit diversen Applikationen und einigen angebundenen Unterzentralen unter Verwendung der hier beschriebenen Kernsoftware.

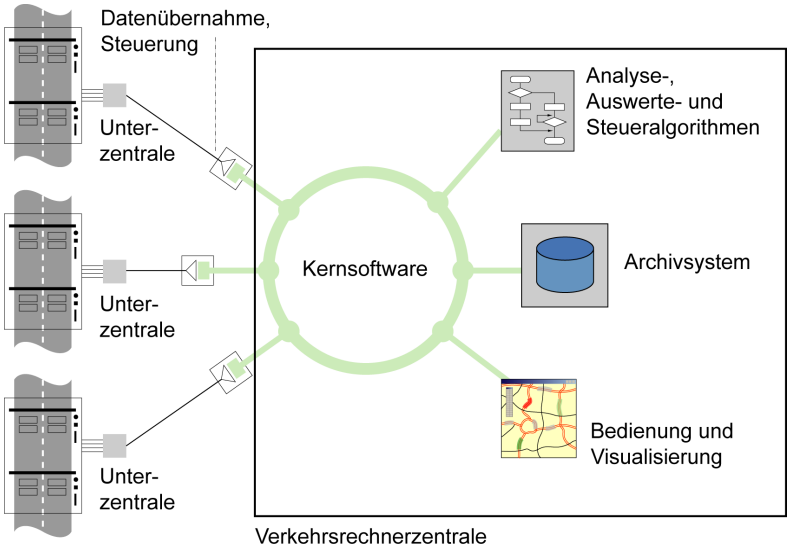


Abbildung 1-3: Anbindung von Applikationen an den Datenverteiler

1.4.1 Komponenten

Die Kernsoftware besteht aus den Komponenten *Datenverteiler*, *Konfiguration*, *Parametrierung*, *Betriebsmeldung* und den *Kernsoftware-Tools*.

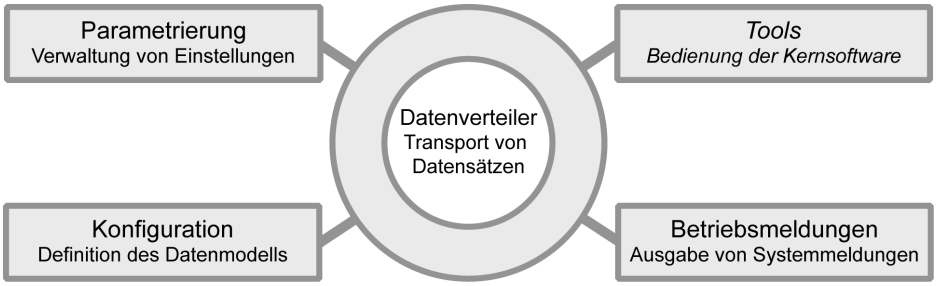


Abbildung 1-4: Bestandteile der Kernsoftware

- Der *Datenverteiler* übernimmt den Transport von Datensätzen über ein TCP/IP Netz.
- Die *Konfiguration* legt die Struktur dieser Datensätze fest.
- Über die *Parametrierung* können systemweite Einstellungen verwaltet werden.
- Die *Betriebsmeldungen* erlauben das standardisierte Absetzen von Textnachrichten über den Datenverteiler.
- Mit den *Kernsoftware-Tools* können grundlegende Funktionen der Kernsoftware verwendet werden. Das vielseitigste dieser Tools, der *GTM*, wird im Abschnitt 1.5 vorgestellt.

Für den Betrieb sind nur die ersten vier Komponenten notwendig. Diese werden über das mitgelieferte Skript gestartet. Die Kernsoftware-Tools können bei Bedarf gestartet werden.

1.4.2 Datenverteiler

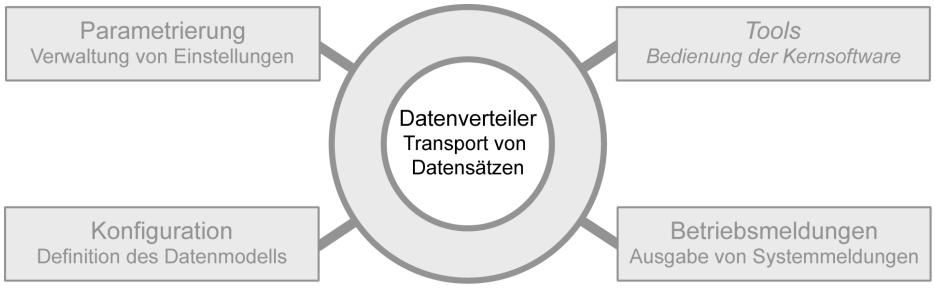


Abbildung 1-5: Komponente Datenverteiler

Der Datenverteiler ist eine kommunikationsorientierte Middleware. Eine Applikation legt Daten auf den Datenverteiler, ohne einen Empfänger anzugeben. Der Empfänger empfängt den Datensatz, ohne dessen Urheber zu kennen. Diese Art des Datenaustauschs nennt man auch *publish-subscribe*[5] Kommunikation. Jede Applikation, die Daten senden oder empfangen will, muss sich als Sender oder Empfänger beim Datenverteiler anmelden. Dazu teilt sie dem Datenverteiler die *Datenidentifikation* der Daten mit, die gesendet oder empfangen werden sollen. Die Datenidentifikation kennzeichnet die Datensätze eines Kommunikationskanals eindeutig, und besteht aus vier Elementen:

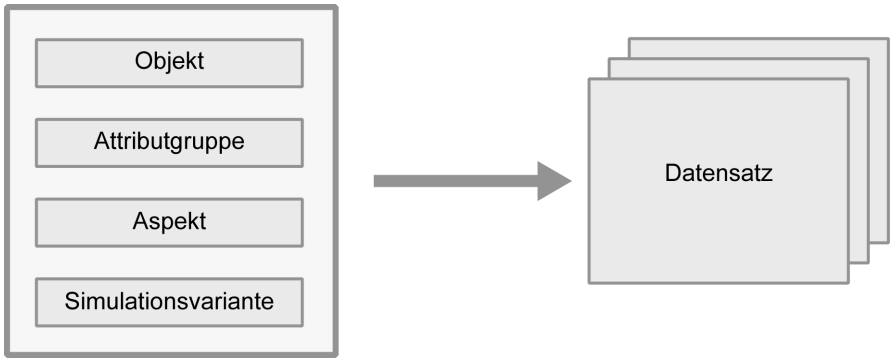


Abbildung 1-6: Die Datenidentifikation kennzeichnet Datensätze eines Kommunikationskanals

- Das Objekt repräsentiert einen Gegenstand oder ein Konzept aus der realen Welt, z.B. einen Messquerschnitt oder einen Stau.
- Die Attributgruppe gibt Auskunft über die Struktur der Datensätze.
- Der Aspekt bezeichnet eine Sicht auf die Daten, etwa, ob es sich um aggregierte Daten handelt.
- Die Simulationsvariante zeigt, ob es sich um Daten handelt, die zu Simulationszwecken in das System eingespielt wurden.

Jeder dieser Bestandteile kann in mehreren verschiedenen Datenidentifikationen verwendet werden, beispielsweise können zwei unterschiedliche Datenidentifikationen die gleiche Attributgruppe verwenden.

[5] http://en.wikipedia.org/wiki/Publish/subscribe

1.4.3 Konfiguration

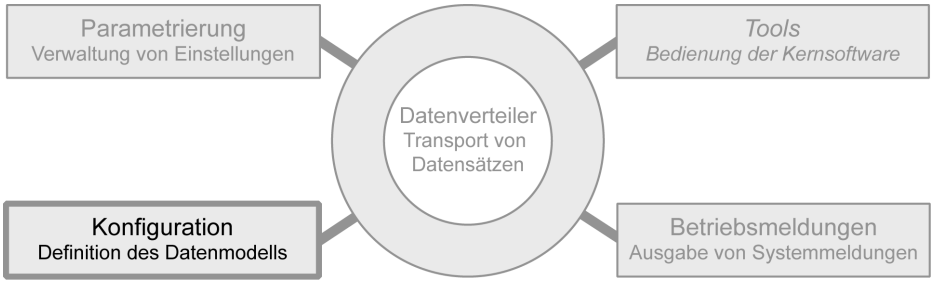
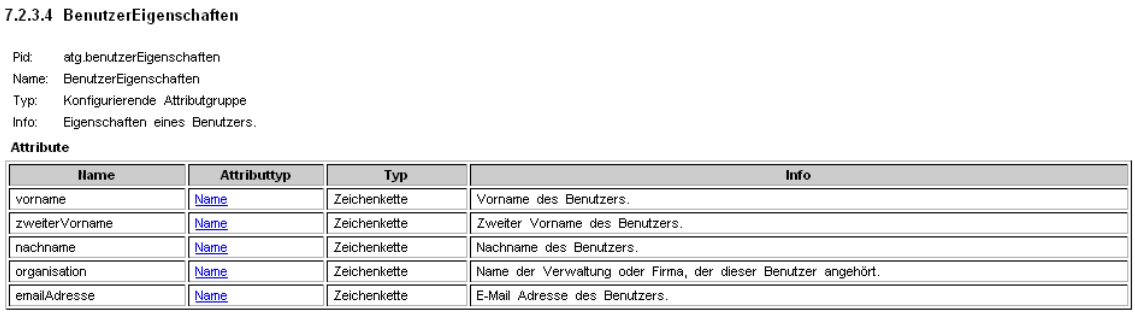


Abbildung 1-7: Komponente Konfiguration

Die Softwareeinheit *Konfiguration* verwaltet das *Datenmodell* der Kernsoftware. Das Datenmodell legt fest, welche Datenidentifikationen existieren und bestimmt deren Struktur. Alle Anfragen das Datenmodell betreffend werden ausschließlich an die Konfiguration gerichtet. Die Kommunikation dabei erfolgt ebenfalls über den Datenverteiler. Ebenso ist die Konfiguration die einzige Softwareeinheit, die Änderungen am Datenmodell vornehmen darf.

Das Datenmodell selbst wird im *Datenkatalog*, der in unterschiedlichen Darstellungsformen vorliegt, definiert. Für die Einsicht durch Benutzer der Kernsoftware eignet sich der *HTML-Datenkatalog*[6]. Dieser ist nicht Lieferbestandteil der Kernsoftware, sondern muss separat heruntergeladen[7] werden.

Die folgende Abbildung zeigt exemplarisch anhand der Attributgruppe `atg.Benutzereigenschaften` wie das Datenmodell im HTML-Datenkatalog dargestellt wird. Die Attributgruppe modelliert die Eigenschaften eines Benutzers, der einen Vornamen, Nachnamen, usw. besitzt. Sie kann verwendet werden, um Benutzerinformationen über den Datenverteiler zu übertragen.



7.2.3.4 BenutzerEigenschaften

Pid: atg.benutzerEigenschaften
Name: BenutzerEigenschaften
Typ: Konfigurierende Attributgruppe
Info: Eigenschaften eines Benutzers.

Attribute

Name	Attributtyp	Typ	Info
vorname	Name	Zeichenkette	Vorname des Benutzers.
zweiterVorname	Name	Zeichenkette	Zweiter Vorname des Benutzers.
nachname	Name	Zeichenkette	Nachname des Benutzers.
organisation	Name	Zeichenkette	Name der Verwaltung oder Firma, der dieser Benutzer angehört.
emailAdresse	Name	Zeichenkette	E-Mail Adresse des Benutzers.

Abbildung 1-8: Darstellung einer Attributgruppe im HTML-Datenkatalog

Das Datenmodell der Kernsoftware ist sehr flexibel. Es kann den Gegebenheiten entsprechend angepasst werden. Es ist zu beachten, dass zur Laufzeit[8] nur bestimmte Änderungen, z.B. das Erzeugen von speziellen Objekten, am Datenmodell vorgenommen werden können. Ein tieferer Eingriff in das Datenmodell, wie z.B. die Änderung der Struktur eines Datensatzes, ist über die Softwareeinheit Konfiguration nicht möglich, sondern erfordert das manuelle Editieren des Datenkatalogs (siehe Abschnitt 3.3).

[6] ../akvrz/DatK/start.html
[7] http://zid.almo-traffic.de/uploads/media/DatK.zip
[8] "Unter der *Laufzeit* versteht man in der Informatik die Zeitspanne, innerhalb deren ein kompiliertes oder interpretiertes Programm im Arbeitsspeicher residiert und ausgeführt wird [...].", siehe Wikipedia [http://de.wikipedia.org/wiki/Laufzeit_(Informatik)].

1.4.4 Parametrierung

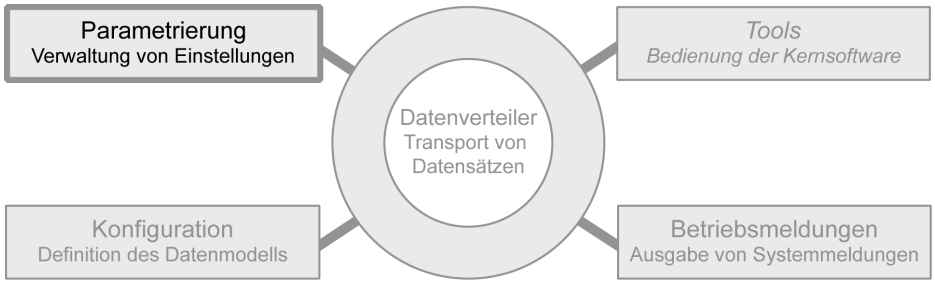


Abbildung 1-9: Komponente Parametrierung

Die Parametrierung verwaltet systemweite Einstellungen, die sich zur Laufzeit ändern können. Diese Einstellungen können nicht im Datenmodell hinterlegt werden, da in diesem nur nicht-änderbare Datensätze hinterlegt werden können. Über die Parametrierung kann z.B. festgelegt werden, welche Daten archiviert werden sollen. Diesen Vorgang nennt man *parametrieren*.

> **Anmerkung**
>
> Für den Betrieb der Kernsoftware-Komponenten müssen zunächst keine Einstellungen parametriert werden.

1.4.5 Betriebsmeldungen

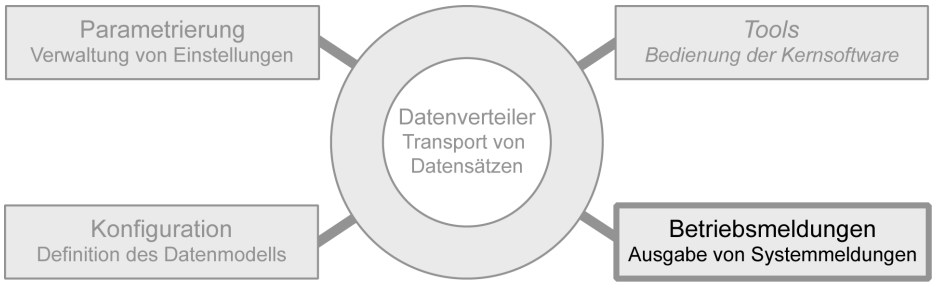


Abbildung 1-10: Komponente Betriebsmeldungen

Diese Softwareeinheit dient der Verwaltung der von den Applikationen gesendeten *Betriebsmeldungen*. Eine Betriebsmeldung ist eine Textnachricht, die wie jeder andere Datensatz auch vom Datenverteiler transportiert wird. Ein Archivsystem nach [TANFARS] verwendet die Betriebsmeldungen beispielsweise, um den Anwender auf Probleme hinzuweisen.

> **Anmerkung**
>
> Die Komponenten der Kernsoftware versenden selbst keine Betriebsmeldungen.

1.4.6 Kernsoftware-Tools

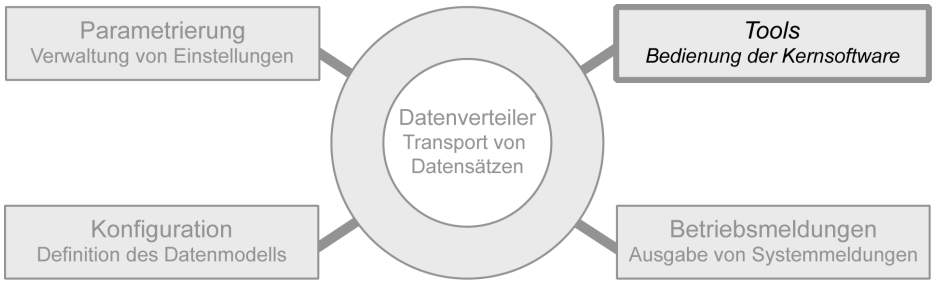


Abbildung 1-11: Komponente Kernsoftware Tools

Die wichtigsten Tools der Kernsoftware sind:

- *Generischer Testmonitor (GTM)*

  Mit dem GTM können grundlegende Funktionen der Kernsoftware verwendet werden, z.B. Daten senden und empfangen, Einstellungen parametrieren und Archivanfragen stellen. Der GTM wird im Abschnitt 1.5 vorgestellt.

- *Datengenerator*

  Der Datengenerator ermöglicht das Senden von im XML-Format vorliegenden Datensätzen. Die Verwendung des Datengenerators wird in Abschnitt 3.2.1.4 beschrieben.

- *Datenmodellbrowser*

  Das Datenmodell der Kernsoftware kann mit diesem Tool eingesehen werden. In Abschnitt 3.2.3 finden sich weitere Informationen zum Datenmodellbrowser.

1.4.7 Senden und Empfangen von Daten

Der Transport von Daten ist die Hauptaufgabe der Kernsoftware. Im Folgenden soll anhand eines Beispiels gezeigt werden, wie der Datenverteiler und die angeschlossenen Applikationen beim Senden und Empfangen von Daten vorgehen.

Beispiel: Ein Messquerschnitt misst das Verkehrsaufkommen eines Straßensegments. Ein Tool zur Analyse möchte diese Daten empfangen und weiterverarbeiten.

1. Der Messquerschnitt meldet sich beim Datenverteiler als Quelle für die Daten an. Dazu stellt er die korrekte Datenidentifikation, bestehend aus Objekt, Attributgruppe, Aspekt und Simulationsvariante, zusammen.

2. Das Analysetool meldet sich beim Datenverteiler als Empfänger für diese Datenidentifikation an.

3. Der Messquerschnitt legt einen Datensatz auf den Datenverteiler.

4. Der Datenverteiler stellt den angemeldeten Empfängern, hier dem Analysetool, den Datensatz zu.

   Sendet der Messquerschnitt mehrere Datensätze, so wiederholen sich die Schritte 3 bis 4 dementsprechend oft. Die Datensätze werden dem Empfänger in der gleichen Reihenfolge übergeben, in der sie vom Messquerschnitt gesendet werden.

5. Der Messquerschnitt und das Analysetool melden sich beim Datenverteiler ab.

> **Anmerkung**
>
> Die Reihenfolge, in der sich Empfänger und Sender an- und abmelden, ist beliebig.

1.5 Generischer Testmonitor (GTM)

Der GTM ist ein Werkzeug zur manuellen Steuerung, Konfiguration und Test der Kernsoftware und Archivsystem. Außerdem bietet er grundlegende Funktionen wie das Versenden von Datensätzen über den Datenverteiler, usw. Im Folgenden sollen in einem kurzen Überblick die Werkzeuge zur Verwendung der Kernsoftware erklärt werden. Genaueres findet sich im Kapitel Abschnitt 3.2.1.

Nach dem Start der Kernsoftware kann der GTM mit dem Skript `GenerischerTestmonitor.bat` in der Windows- , bzw. `GenerischerTestmonitor.sh` in der Linux-Variante gestartet werden. Beide Skripte finden sich im Skriptverzeichnis der Kernsoftwareinstallation. Die folgende Abbildung zeigt das Hauptfenster des GTM:

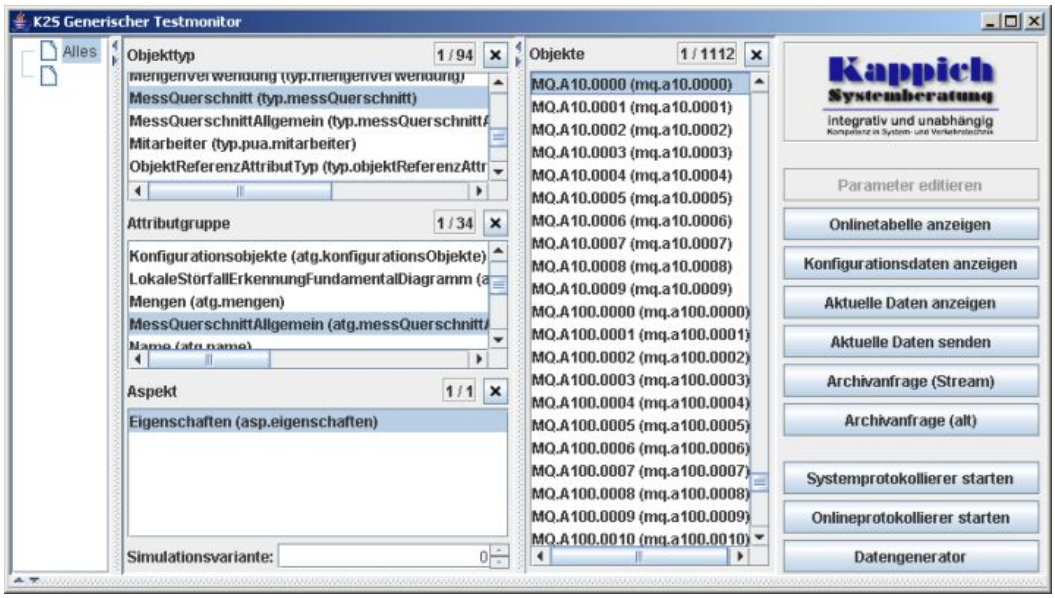


Abbildung 1-12: Generischer Testmonitor

Nachdem das Hauptfenster angezeigt wird, muss auf den Eintrag *Alles* in der linken Liste geklickt werden. Dadurch werden die übrigen Listen mit Einträgen gefüllt. Damit die Buttons auf der rechten Seite des Fensters aktiviert werden, muss zuerst eine Datenidentifikation ausgewählt werden. Dazu ist wie folgt vorzugehen:

1. Objekttyp auswählen (optional),
2. Attributgruppe selektieren,
3. Aspekt wählen,
4. Simulationsvariante angeben, und
5. konkretes Objekt wählen.

Anmerkung

Bei der Auswahl ist zu beachten, dass falls ein Objekttyp, eine Attributgruppe oder ein Aspekt gewählt wird, die jeweils anderen Listen so aktualisiert werden, dass nur noch die für die Auswahl verfügbaren Elemente selektierbar sind. Sollen wieder alle Elemente zur Verfügung stehen, muss der kleine *X*-Button am rechten oberen Listenrand gedrückt werden.

Die folgenden Abschnitte geben einen kurzen Überblick über die Buttons auf der rechten Seite des GTM-Hauptfensters.

1.5.1 Parameter editieren

Mit Hilfe dieses Tools wird die Parametrierung durchgeführt, die zur Konfiguration des Systems und angeschlossener Applikationen dient. Zum Beispiel kann so die Archivsystemparametrierung erzeugt und angepasst werden.

Genauere Erklärungen zur Parametrierung sind für den Schnelleinstieg nicht relevant und sind im Theorieteil (ab Kapitel 2) zu finden.

1.5.2 Onlinetabelle

Mit diesem Werkzeug können über den Datenverteiler transportierte Datensätze einer Datenidentifikation tabellarisch dargestellt werden. Das Tool muss dazu entsprechend auf die gewünschte Datenidentifikation eingestellt werden (siehe auch Abbildung 1-22). Eine typische Ausgabe der Onlinetabelle ist in Abbildung 1-13 dargestellt:

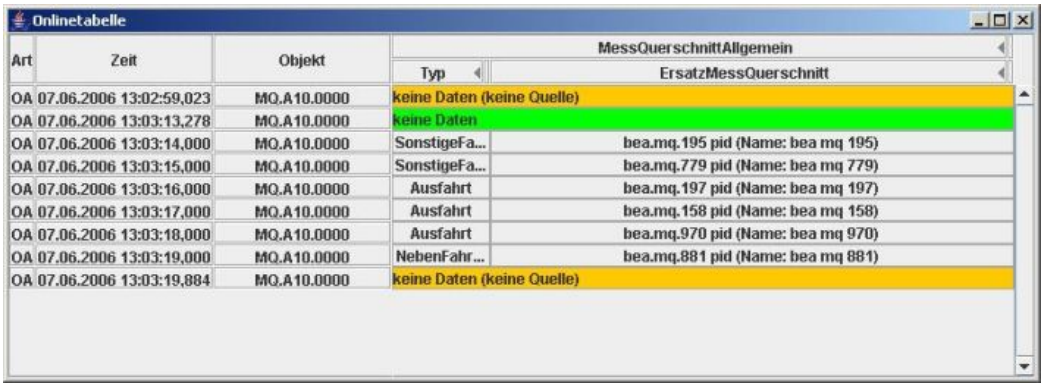


Abbildung 1-13: Ausgabe der Onlinetabelle

1.5.3 Konfigurationsdaten anzeigen

Diese Funktion erlaubt das Abrufen der Daten, die im Datenmodell zu Objekten und Attributgruppen hinterlegt sind. Beispielsweise sind für das Objekt `straße.B191` und die Attributgruppe `atg.straße` der Straßentyp und die Straßennummer als *konfigurierende* Daten hinterlegt.

Abbildung 1-14: Konfigurierende Daten

1.5.4 Aktuelle Daten anzeigen

Das Tool zum Anzeigen aktueller Daten realisiert eine ähnliche Funktionalität wie die Onlinetabelle (Abschnitt 1.5.2) oder der Onlineprotokollierer (Abschnitt 1.5.9). Es wird jedoch immer nur der momentan aktuellste Datensatz einer Datenidentifikation angezeigt. Das Tool muss dazu entsprechend eingestellt werden.

Abbildung 1-15: Aktuelle Daten anzeigen

Das Einstellen des Tools erfolgt analog zum Onlineprotokollierer (Abschnitt 1.5.9).

1.5.5 Aktuelle Daten senden

Mit diesem Modul kann ein neuer Datensatz für die gewählte Datenidentifikation erstellt und über den Datenverteiler verschickt werden. Diese Daten können dann z.B. mit dem Modul *Aktuelle Daten anzeigen* (siehe Abschnitt 1.5.4) betrachtet werden.

Für das Modul *Aktuelle Daten senden* gelten die gleichen Bedingungen für eine gültige Datenidentifikation wie für das Modul *Aktuelle Daten anzeigen*. Es muss also genau eine Attributgruppe, ein Aspekt und ein Objekt ausgewählt werden.

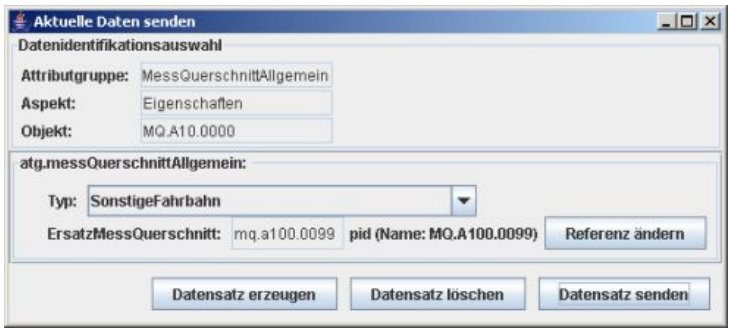


Abbildung 1-16: Aktuelle Daten versenden

1.5.6 Archivanfrage (Stream)

Mit diesem Werkzeug können streambasierte Archivanfragen durchgeführt werden. Ein Archiv liefert Datensätze einer zuvor ausgewählten Datenidentifikation und eines bestimmten Zeitbereichs. Eine genauere Beschreibung der Archivanfrageparameter erfolgt im Abschnitt 3.2.1.2.

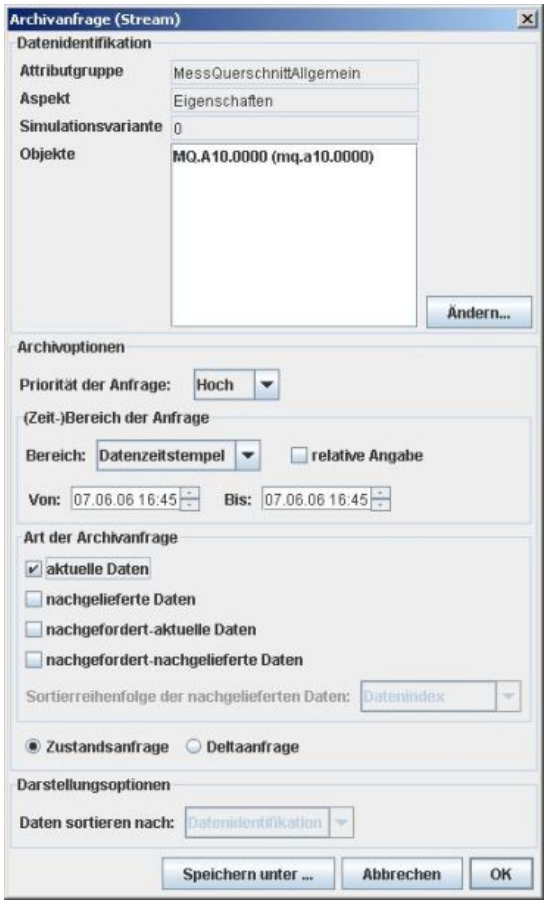


Abbildung 1-17: Parametrierung der streambasierten Archivanfrage

> **Wichtig**
>
> Bei der Verwendung der Archivanfrage muss zunächst eine *Art der Archivanfrage* selektiert, bzw. deselektiert und erneut selektiert werden, da sonst der *OK*-Button nicht aktiviert wird.

Das Ergebnis der Archivanfrage wird in einer Tabelle angezeigt und entspricht der Ausgabe der Onlinetabelle (siehe Abbildung 1-13).

1.5.7 Archivanfrage (alt)

Hier können nicht-streambasierte Anfragen an das Archivsystem gestellt werden. Archivsysteme nach [TANFARS] unterstützen diese Schnittstelle jedoch nicht, sondern nur die streambasierte Anfrage (Abschnitt 1.5.6).

1.5.8 Systemprotokollierer

Der Systemprotokollierer kann ebenfalls Anfragen an das Archiv stellen. Der Systemprotokollierer unterscheidet sich von der *Archivanfrage (Stream)* in der Darstellung der abgefragten Daten und einer verringerten Anzahl von Einstellungsmöglichkeiten. Die Darstellung der Daten erfolgt analog zum Onlineprotokollierer, siehe Abschnitt 1.5.9.

Eine genauere Beschreibung ist im Abschnitt 3.2.1.3 zu finden.

1.5.9 Onlineprotokollierer

Der Onlineprotokollierer ist ein Werkzeug ähnlich der Onlinetabelle. Mit ihm können die über den Datenverteiler gesendeten Daten mitgeschnitten werden. Die Ausgabe erfolgt entweder auf dem Bildschirm oder in eine Datei.

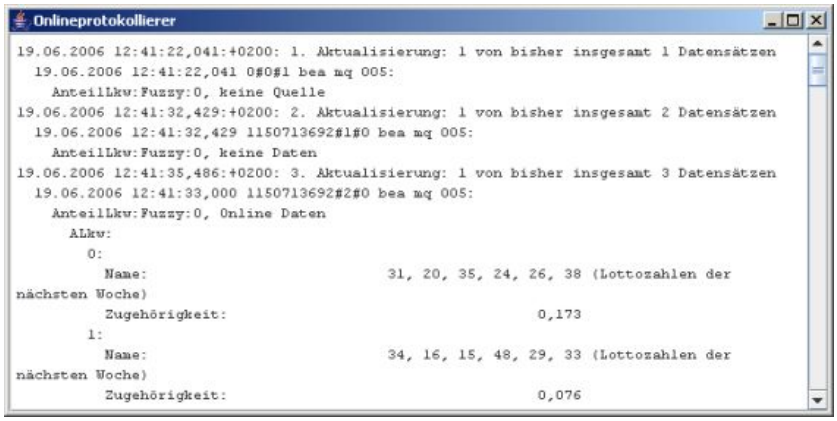


Abbildung 1-18: Ergebnisfenster des Onlineprotokollierers

Im Unterschied zur Onlinetabelle (Abschnitt 1.5.2) erfolgt die Ausgabe hier im Klartextformat, siehe Abbildung 1-18.

Eine genauere Beschreibung des Onlineprotokollierers findet sich im Praxisteil des Tutorials in Abschnitt 3.2.1.1. Dort wird detailierter auf die unterstützten Ausgabeformate und Einstellungsmöglichkeiten eingegangen.

1.5.10 Datengenerator

Mit dem entsprechend eingestellten Datengenerator lassen sich zufällige Datensätze zu einer Datenidentifikation erzeugen. Diese können z.B. für Tests verwendet werden.

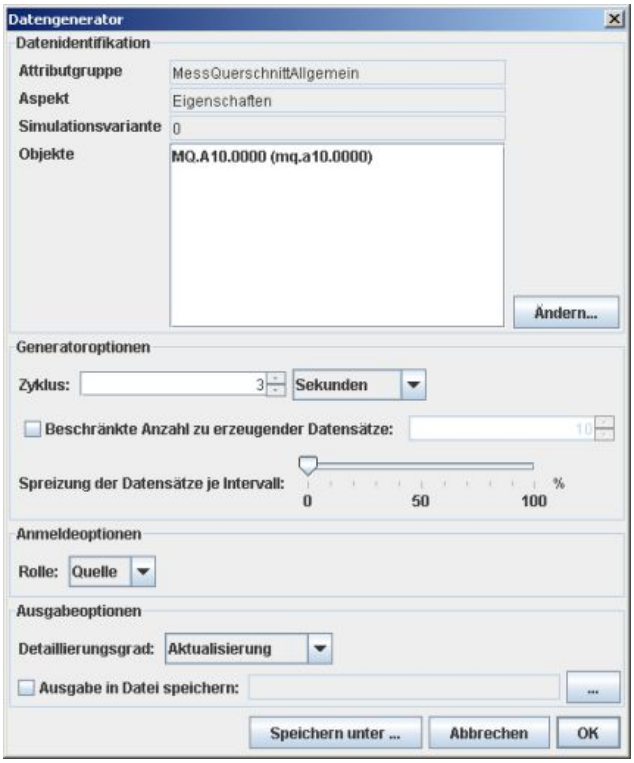


Abbildung 1-19: Datengenerator einstellen

Datensätze mit Zufallswerten werden nach Anklicken des *OK*-Buttons erzeugt und auf den Datenverteiler gelegt. Sollen die erzeugten Daten zur späteren Wiederverwendung gespeichert werden, muss ein Dateiname angegeben werden, unter dem sie auf der Festplatte abgelegt werden. Werden die Daten nicht mehr benötigt, muss die Datei manuell aus dem Dateisystem gelöscht werden.

Details zu den Einstellmöglichkeiten des Datengenerators sind in Abschnitt 3.2.1.4 beschrieben.

1.5.11 Senden und Empfangen

Nachdem nun das Funktionsprinzip des Generischen Testmonitors bekannt ist, kann ein erstes Szenario ausprobiert werden. Im Folgenden soll der GTM benutzt werden, um Datensätze zu generieren, zu senden und sie anschließend zu empfangen.

1. *Wählen einer Datenidentifikation*

   Im Startbildschirm des GTM wird ein Objekttyp, eine Attributgruppe, ein Aspekt, die Simulationsvariante und ein konkretes Objekt (siehe Abbildung 1-20) gewählt.

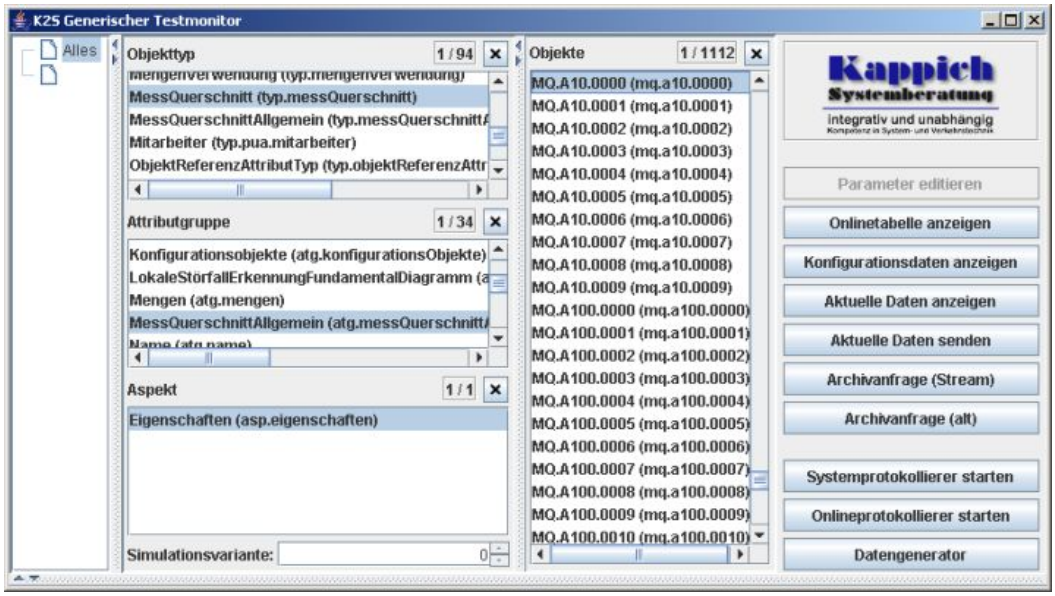


Abbildung 1-20: Generischer Testmonitor

2. *Daten senden*

   Um die Daten zu erzeugen und zu senden, muss aus dem GTM der Datengenerator durch Klick auf den Button *Datengenerator* gestartet und wie in Abbildung 1-19 eingestellt werden.

   Der Datengenerator wird nach dem Anklicken des *OK*-Buttons nun in Abständen von drei Sekunden Datensätze für den Messquerschnitt `mq.a10.0000` erzeugen und diese auf den Datenverteiler legen. Zur Kontrolle erscheint im Ergebnisfenster des Datengenerators folgende Ausgabe.

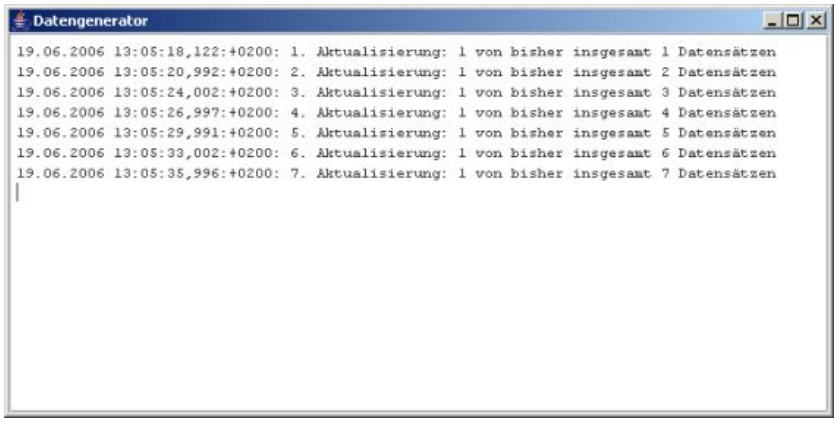


Abbildung 1-21: Ergebnisfenster des Datengenerators

3. *Daten empfangen*

   Um Datensätze, die vom Datengenerator oder anderen Applikationen erzeugt werden, zu empfangen, können verschiedene Tools des Generischen Testmonitors zum Einsatz kommen. An dieser Stelle soll jedoch nur auf die Verwendung der Onlinetabelle eingegangen werden, da mit dieser die Ausgabe sehr anschaulich dargestellt wird. Für das Beispiel muss die Onlinetabelle wie in Abbildung 1-22 eingestellt werden.

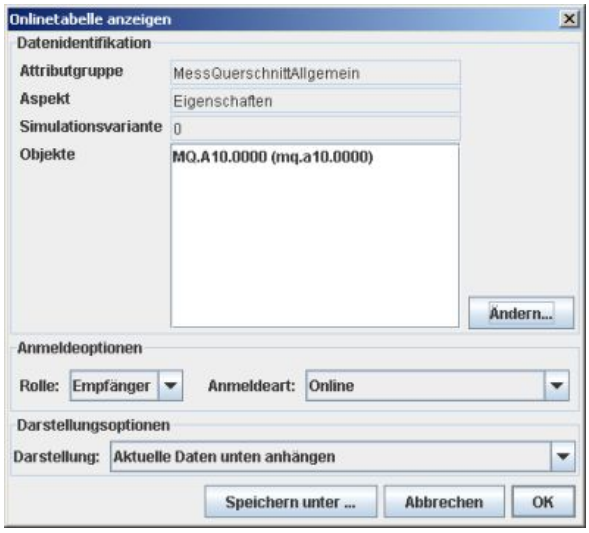


Abbildung 1-22: Einstellungen für die Onlinetabelle

Falls alle Schritte korrekt durchgeführt wurden, empfängt die Onlinetabelle nun Daten, da sie als Empfänger auf die vom Datengenerator erzeugten und auf den Datenverteiler gelegten Daten angemeldet ist. Die Ausgabe erfolgt in Tabellenform und sollte in etwa der in Abbildung 1-23 entsprechen.

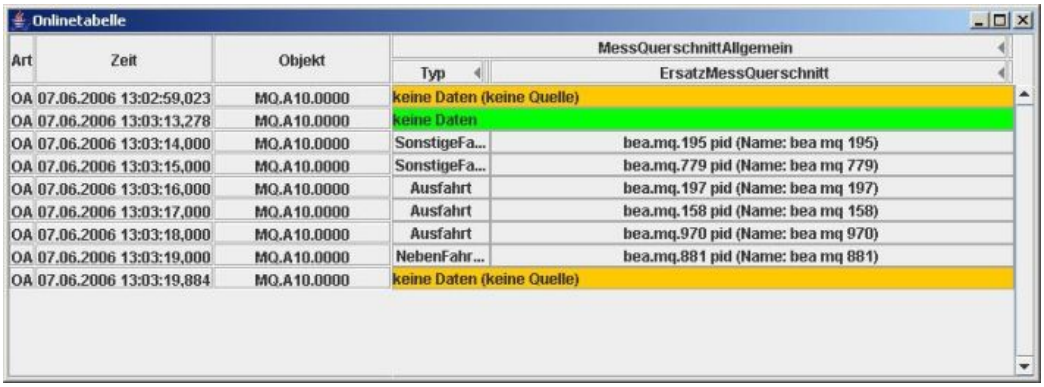


Abbildung 1-23: Ausgabe der Onlinetabelle

4. *Beenden*

   Um Sender und Empfänger zu beenden, muss nur das jeweilige Fenster geschlossen werden.

1.6 Programmierbeispiele

In diesem Kapitel soll anhand eines einfachen Beispiels gezeigt werden, wie eine Applikation zum Empfang von Daten über den Datenverteiler implementiert werden kann. Die Applikation wird empfangene Datensätze auf der Konsole ausgeben.

Weitere Beispiele für Applikationen, die z.B. das Erzeugen und Senden von Daten übernehmen, Applikationen zum Empfangen der Parametrierung usw. werden in Abschnitt 3.5 eingeführt.

Für die Implementierung wird die Java-API der Kernsoftware [APIBIB] verwendet. Eine Beschreibung der API findet sich im Verzeichnis *dokumentation* der Kernsoftwareinstallation.

Die vollständigen Quelltexte der hier verwendeten Klassen können im Abschnitt 1.7 eingesehen werden.

1.6.1 Verbindung zum Datenverteiler herstellen

Eine Verbindung zum Datenverteiler kann beispielsweise wie in der folgenden Funktion `connect()` realisiert werden.

Im String `davHost` wird die Adresse des Datenverteilers, z.B.

```
-datenverteiler=hauptsystem:8083
```

angegeben, `davUsername` und `davPassword` enthalten Authentifizierungsinformationen über den anzumeldenden Benutzer. Darüber hinaus sind noch andere Parameter für die Verbindung zum Datenverteiler möglich (siehe Abschnitt 3.5).

```
public class DavConnection {
    public static ClientDavInterface connect(String davHost, String davUsername,
            String davPassword) {
        ClientDavInterface dav = null;
        try {
            // Argumente
            ArgumentList argList = new ArgumentList(new String[] { davHost });
            ClientDavParameters cdp = new ClientDavParameters(argList);

            // DAV-Verbindung
            dav = new ClientDavConnection(cdp);
            dav.connect();
            dav.login(davUsername, davPassword);
        }
        catch (Exception e) {
            System.out.println("Fehler: " + e);
        }
        return dav;
    }

    ...
}
```

Ausnahmen und Fehler, die beim Verbindungsaufbau auftreten können, werden ebenfalls in Abschnitt 3.5 genauer erklärt.

1.6.2 Verbindung zum Datenverteiler beenden

Folgendes Listing zeigt beispielhaft die Implementierung einer einfachen Funktion zum Trennen der Datenverteilerverbindung.

Der Parameter `davConnection` stellt ein Objekt, über das die Verbindung zum Datenverteiler besteht, bereit (könnte z.B. durch die Funktion `connect()` erzeugt werden). `withErrors` zeigt an, ob die Trennung aufgrund eines Fehlers stattfindet und mit `message` kann eine Nachricht spezifiziert werden, die beim Beenden ausgegeben wird.

```
public class DavConnection {

    ...

    public static void disconnect(ClientDavInterface davConnection, boolean withErrors,
            String message) {
        davConnection.disconnect(withErrors, message);
    }
}
```

Die Funktion `davConnection.disconnect(boolean withErrors, String message)` könnte auch direkt aus der aufrufenden Applikation ausgeführt werden, der Übersichtlichkeit halber ist sie aber über die Klasse `DavConnection` verfügbar, die alle Aktionen bezüglich der Datenverteilerverbindung kapselt.

1.6.3 Daten empfangen

Um Daten mit einer Applikation zu Empfangen sind prinzipiell folgende Schritte nötig:

- Verbindung zum Datenverteiler herstellen
- Anmelden des Empfängers
- Empfangen der Daten
- Abmelden des Empfängers
- Trennen der Datenverteilerverbindung

1.6.3.1 Vorbereitung zum Empfang von Daten

Damit Daten mit einer Applikation empfangen werden können, müssen zunächst passende Daten erzeugt und versandt werden. Dies kann auf mehrere Arten geschehen, zum Beispiel:

- *Senderapplikation*

  Die exemplarische Implementierung einer Senderapplikation ist in Abschnitt 3.5.3 nachzuschlagen.

- *GTM mit Datengenerator*

In diesem Beispiel wird der Datengenerator aus dem GTM gestartet und zur Erzeugung von Testdatensätzen verwendet. Im GTM wird der Objekttyp `MessQuerschnitt` mit Attributgruppe `MessQuerschnittAllgemein`, Aspekt `Eigenschaften` und Simulationsvariante `0`, sowie das Objekt `mq.a10.0000` gewählt. Der Datengenerator muss wie in Abbildung 1-19 eingestellt werden.

Genaueres zu den Einstellmöglichkeiten des GTM, bzw. Datengenerators findet sich in Abschnitt 3.2.

> **Wichtig**
>
> Die verwendete Datenidentifikation des Senders muss mit derjenigen des nachfolgend implementierten Empfängers übereinstimmen, da dieser sonst keine Daten empfangen wird.

1.6.3.2 Verbindung zum Datenverteiler herstellen und Initialisierung

Die Verbindung zum Datenverteiler wird über die Methoden der gerade implementierten Klasse `DavConnection` hergestellt. Danach werden verschiedene Variablen initialisiert, die später für die Anmeldung als Empfänger beim Datenverteiler benötigt werden.

Details zum Verbindungsaufbau, zur Initialisierung, mögliche Optionen und andere relevante Implementierungsdetails werden im Praxisteil des Tutorials in Abschnitt 3.5.1 erklärt.

```
public class Empfaenger implements ClientReceiverInterface {

    ...

    /** Verbindung zum Datenverteiler herstellen */
    public void connect() {
        // Adresse und Port des Datenverteilers
        String davAdresse  = "-datenverteiler=localhost:8083";
        // Benutzername
        String davBenutzer = "Tester";
        // Passwort
        String davPasswort = "geheimespasswort";

        // Verbinden
        datenverteiler = DavConnection.connect(davAdresse, davBenutzer, davPasswort);

        System.out.println("Verbunden.");
    }
```

```
    ...

    /** Handles initialisieren */
    public void init() {

        // PID des Objekts
        String objectPid = "mq.a10.0000";
        // PID der Attributgruppe
        String objectAtg = "atg.messQuerschnittAllgemein";
        // PID des Aspekts
        String objectAsp = "asp.eigenschaften";
        // Simulationsvariante
        short  objectSmv = 0;

        // Empfaengeroptionen
        anmeldeoptionen = ReceiveOptions.normal(); // Wie soll empfangen
                                                   // werden? Alles.
        empfaengerrolle = ReceiverRole.receiver(); // Rolle der Applikation:
                                                   // Empfaenger

        // Bestandteile der Datenidentifikation initialisieren
        systemobjekt        = datenverteiler.getDataModel().getObject(objectPid);
        attributgruppe      = datenverteiler.getDataModel().getAttributeGroup(objectAtg);
        aspekt              = datenverteiler.getDataModel().getAspect(objectAsp);
        simulationsvariante = objectSmv;

        datenbeschreibung   = new DataDescription(
                                  attributgruppe,
                                  aspekt,
                                  simulationsvariante);

        System.out.println("Initialisiert.");
    }

    ...

}
```

1.6.3.3 Anmelden als Empfänger

Nach der Initialisierung muss die Applikation als Empfänger für die spezifizierten Daten angemeldet werden.

```
public class Empfaenger implements ClientReceiverInterface {

    ...

    /** Anmeldung zum Empfang von Daten */
    public void subscribe() {
        // Anmelden
        datenverteiler.subscribeReceiver(
                Empfaenger.this,
                systemobjekt,
                datenbeschreibung,
                anmeldeoptionen,
                empfaengerrolle);

        System.out.println("Angemeldet.");
    }

    ...
}
```

1.6.3.4 Empfang der Daten

Die Methode `update()` des Interface `ClientReceiverInterface` wird jedesmal ausgeführt, wenn der Datenverteiler einen Datensatz zustellt.

Meldet sich die Quelle ab, liefert `data.getDataState()` des nächsten Datensatzes den Wert `DataState.NO_SOURCE` und es wird `notify()` aufgerufen. Damit wird die `main()`-Methode benachrichtigt und anschließend fortgefahren wie in Abschnitt 1.6.3.5.

```
public class Empfaenger implements ClientReceiverInterface {

    ...

    /** Ausgabe der empfangenen Daten */
    public void update(ResultData[] resultData) {
        System.out.println("Update.");

        // Ausgeben
        for (ResultData data : resultData) {
```

```
            // Empfang beenden, wenn keine Quelle mehr vorhanden ist
            if (data.getDataState() == DataState.NO_SOURCE)
                // Benachrichtigen der main()-Methode
                synchronized (this) { notify(); }
            System.out.println("\t" + data);
        }
    }

    ...
}
```

1.6.3.5 Abmelden des Empfängers und Trennen der Datenverteilerverbindung

Nachdem der Datenempfang abgeschlossen ist, kann der Empfänger abgemeldet und die Verbindung zum Datenverteiler getrennt werden.

```
public class Empfaenger implements ClientReceiverInterface {

    ...

    /** Abmeldung vom Empfang der Daten */
    public void unsubscribe() {
        // Abmelden
        datenverteiler.unsubscribeReceiver(
                Empfaenger.this,
                systemobjekt,
                datenbeschreibung);

        System.out.println("Abgemeldet.");
    }

    ...

    /** Verbindung zum Datenverteiler trennen */
    public void disconnect() {
        // Trennen
        DavConnection.disconnect(
                datenverteiler,
                false,
                "");

        System.out.println("Getrennt.");
    }

    ...

}
```

1.7 Programmlistings

Hier sind die vollständigen Listings der in diesem Kapitel vorgestellten Klassen noch einmal aufgeführt.

1.7.1 DavConnection.java

Vollständiges Listing des Verbindungsmanagers:

```
package de.bast.vrz3.tutorial;

import stauma.dav.clientside.ClientDavConnection;
import stauma.dav.clientside.ClientDavInterface;
import stauma.dav.clientside.ClientDavParameters;
import sys.funclib.ArgumentList;

/**
 * Tutorial : Schnelleinstieg
 * Die Klasse kapselt das Verbinden zum und Trennen vom Datenverteiler.
 *
 * @author beck et al. projects GmbH
 */
public class DavConnection {

    /**
     * Verbindung mit dem Datenverteiler herstellen
     *
     * @param davHost
     *            Adresse des Datenverteilers, z.B.
     *            "-datenverteiler=hauptsystem:8083"
     * @param davUsername
     *            Benutzer, der angemeldet werden soll
     * @param davPassword
     *            Passwort des Benutzers
     *
     */
```

```
    public static ClientDavInterface connect(String davHost, String davUsername,
            String davPassword) {
        ClientDavInterface dav = null;
        try {
            // Argumente
            ArgumentList argList = new ArgumentList(new String[] { davHost });
            ClientDavParameters cdp = new ClientDavParameters(argList);

            // DAV-Verbindung
            dav = new ClientDavConnection(cdp);
            dav.connect();
            dav.login(davUsername, davPassword);
        }
        // Fehler
        catch (Exception e) {
            System.out.println("Fehler: " + e);
        }
        return dav;
    }

    /**
     * Verbindung zum Datenverteiler trennen
     *
     * @param davConnection
     *            Handle für die Verbindung, z.B. durch connect(...) erzeugt.
     * @param withErrors
     *            Gibt an, ob die Verbindung wegen eines Fehlers beendet wird.
     * @param message
     *            Kann eine Nachricht enthalten, die beim Beenden der Verbindung
     *            ausgegeben wird.
     */
    public static void disconnect(ClientDavInterface davConnection, boolean withErrors,
            String message) {
        davConnection.disconnect(withErrors, message);
    }
}
```

1.7.2 Empfaenger.java

Vollständiges Listing der Empfängerklasse:

```
package de.bast.vrz3.tutorial;

import stauma.dav.clientside.ClientDavInterface;
import stauma.dav.clientside.ClientReceiverInterface;
import stauma.dav.clientside.DataDescription;
import stauma.dav.clientside.ReceiveOptions;
import stauma.dav.clientside.ReceiverRole;
import stauma.dav.clientside.ResultData;
import stauma.dav.configuration.interfaces.Aspect;
import stauma.dav.configuration.interfaces.AttributeGroup;
import stauma.dav.configuration.interfaces.SystemObject;
import dav.daf.DataState;

/**
 * Tutorial : Schnelleinstieg
 * Diese Klasse relalisiert einen einfachen Empfänger.
 *
 * @author beck et al. projects GmbH
 */
public class Empfaenger implements ClientReceiverInterface {

    // Handles für die Verbindung zum Datenverteiler
    protected ClientDavInterface     datenverteiler;
    protected SystemObject           systemobjekt;
    protected AttributeGroup         attributgruppe;
    protected Aspect                 aspekt;
    protected DataDescription        datenbeschreibung;
    protected short                  simulationsvariante;
    protected ReceiveOptions         anmeldeoptionen;
    protected ReceiverRole           empfaengerrolle;

    /** Verbindung zum Datenverteiler herstellen */
    public void connect() {
        // Adresse und Port des Datenverteilers
        String davAdresse  = "-datenverteiler=localhost:8083";
        // Benutzername
        String davBenutzer = "Tester";
        // Passwort
        String davPasswort = "geheimespasswort";

        // Verbinden
        datenverteiler = DavConnection.connect(davAdresse, davBenutzer, davPasswort);

        System.out.println("Verbunden.");
    }

    /** Verbindung zum Datenverteiler trennen */
    public void disconnect() {
```

```
        // Trennen
        DavConnection.disconnect(datenverteiler, false, "");

        System.out.println("Getrennt.");
    }

    /** Handles initialisieren */
    public void init() {

        // PID des Objekts
        String objectPid = "mq.a10.0000";
        // PID der Attributgruppe
        String objectAtg = "atg.messQuerschnittAllgemein";
        // PID des Aspekts
        String objectAsp = "asp.eigenschaften";
        // Simulationsvariante
        short  objectSmv = 0;

        // Empfaengeroptionen
        anmeldeoptionen = ReceiveOptions.normal(); // Wie soll empfangen
                                                   // werden? Alles.
        empfaengerrolle = ReceiverRole.receiver(); // Rolle der Applikation:
                                                   // Empfaenger

        // Bestandteile der Datenidentifikation initialisieren
        systemobjekt        = datenverteiler.getDataModel().getObject(objectPid);
        attributgruppe      = datenverteiler.getDataModel().getAttributeGroup(objectAtg);
        aspekt              = datenverteiler.getDataModel().getAspect(objectAsp);
        simulationsvariante = objectSmv;

        datenbeschreibung   = new DataDescription(
                                attributgruppe,
                                aspekt,
                                simulationsvariante);

        System.out.println("Initialisiert.");
    }

    /** Anmeldung zum Empfang von Daten */
    public void subscribe() {
        // Anmelden
        datenverteiler.subscribeReceiver(
                Empfaenger.this,
                systemobjekt,
                datenbeschreibung,
                anmeldeoptionen,
                empfaengerrolle);

        System.out.println("Angemeldet.");
    }

    /** Abmeldung vom Empfang der Daten */
    public void unsubscribe() {
        // Abmelden
        datenverteiler.unsubscribeReceiver(
                Empfaenger.this,
                systemobjekt,
                datenbeschreibung);

        System.out.println("Abgemeldet.");
    }

    /** Ausgabe der empfangenen Daten */
    public void update(ResultData[] resultData) {
        System.out.println("Update.");

        // Ausgeben
        for (ResultData data : resultData) {
            // Empfang beenden, wenn keine Quelle mehr vorhanden ist
            if (data.getDataState() == DataState.NO_SOURCE)
                // Benachrichtigen der main()-Methode
                synchronized (this) { notify(); }
            System.out.println("\t" + data);
        }
    }

    /** Einfache Main-Methode für den Empfänger */
    public static void main(String[] args) throws Exception {
        Empfaenger empfaenger = new Empfaenger();
        empfaenger.connect();
        empfaenger.init();
        empfaenger.subscribe();
        // Solange empfangen, bis notify() in der update()-Methode
        // aufgerufen wird.
        synchronized (empfaenger) {
            empfaenger.wait();
        }
        empfaenger.unsubscribe();
        empfaenger.disconnect();
    }
}
```

Kapitel 2 Grundlagen der Kernsoftware

Dieses Kapitel geht auf die Grundlagen der Kernsoftware ein. Es wendet sich vor allem an Systemarchitekten und Programmierer, die auf der Kernsoftware basierende Applikationen entwerfen und umsetzen. Das darauf folgende Kapitel 3 zeigt, wie dieses Wissen in der Praxis angewendet werden kann.

2.1 Datenmodell, Datenkatalog und Konfiguration

2.1.1 Begriffsabgrenzung

Datenmodell:

Das *Datenmodell* legt die Sicht des Systems auf die reale Welt fest. Ein Gegenstand oder Konzept der realen Welt und dessen Beziehungen müssen im Datenmodell definiert werden, damit die Kernsoftware und die darauf basierenden Applikationen damit arbeiten können. Beispielsweise muss jeder einzelne Messquerschnitt, der Daten über den Datenverteiler senden soll, im Datenmodell eingetragen werden.

Das von der Kernsoftware verwendete Datenmodell unterscheidet sich in gewissen Punkten von den in der Informatik geläufigen Datenmodellen. Es setzt sich aus drei Konzepten zusammen, die üblicherweise voneinander getrennt werden:

- Zum Ersten werden die Bestandteile des Datenmodells und deren Beziehungen untereinander festgelegt — man bezeichnet diesen Bereich als Metamodell.

  Analogie: Syntax und Semantik der Programmiersprache Java.

  In der Kernsoftware: Aufbau des Datenmodells, z.B. dass ein Objekt Attributgruppen besitzt, vgl. Abschnitt 1.4.2 und Abschnitt 2.1.2.

- Zum Zweiten können im Datenmodell Typen definiert und deren Eigenschaften festgelegt werden.

  Analogie: Eine Java Klasse.

  In der Kernsoftware: Typdefinitionen, z.B. der Objekttyp Messquerschnitt.

- Zum Dritten werden im Datenmodell konkrete Daten gespeichert.

  Analogie: Ein Java Objekt.

  In der Kernsoftware: Objekte, z.B. ein konkreter Messquerschnitt.

Datenkatalog:

Das Datenmodell liegt physisch in Form des *Datenkatalogs* vor. Für diesen gibt es mehrere Darstellungsformen, die sich für unterschiedliche Zwecke eignen (siehe Abschnitt 2.1.3.1).

Konfiguration:

Die SWE *Konfiguration* verwaltet das Datenmodell und kann daran bestimmte Änderungen zur Laufzeit vornehmen. Diese Änderungen werden am Datenkatalog vorgenommen. Da der Datenkatalog jedoch nur die „physische“ Repräsentation des Datenmodells ist, spricht man von

Anpassungen am Datenmodell. Eine Applikation kann die SWE Konfiguration außerdem verwenden, um Informationen über das Datenmodell, etwa welche Objekte existieren, abzurufen.

Benennung der Elemente:

Jedem Element des Datenmodells kann eine eindeutige *PID* (*permanente Identifikation*) zugewiesen werden, durch die es eindeutig identifiziert werden kann.

> **Wichtig**
>
> Die PID wird einmalig für die gesamte Lebensdauer festgelegt. Sie kann nicht mehr geändert werden. Daher sollte ein sinnvoller Bezeichner gewählt werden.

Neben dieser PID kann zudem noch ein Name vergeben werden. Der Name kann sich ändern und muss nicht eindeutig sein.

> **Anmerkung**
>
> - Im Tutorial werden meistens die Namen von Elementen des Datenmodells verwendet. An manchen Stellen werden jedoch PIDs verwendet, um Elemente eindeutig identifizieren zu können. Diese können an dem Präfix `typ`, `atg`, `att`, `asp`, oder `kv` erkannt werden.
> - Die Elemente des Datenkatalogs können im HTML-Datenkatalog oder mit dem Datenmodellbrowser (Abschnitt 3.2.3) nachgeschlagen werden.

2.1.2 Bestandteile

Die folgende Abbildung zeigt die wichtigsten Bausteine[1] des Datenmodells:

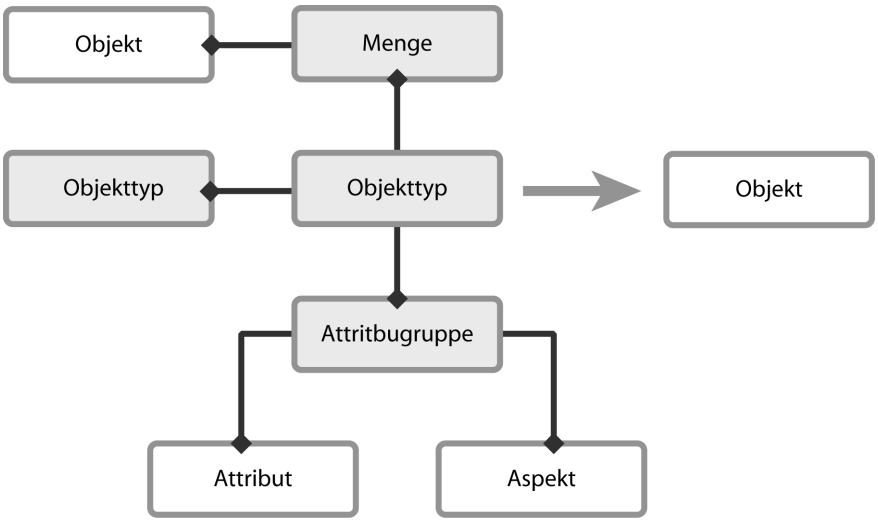


Abbildung 2-1: Bausteine des Datenmodells

[1] Zu diesen Bestandteilen des Datenmodells werden später noch Konfigurationsbereiche (Abschnitt 2.1.5.2) hinzukommen.

Objekt
Ein Objekt repräsentiert einen Gegenstand oder ein Konzept aus der realen Welt, z.B. einen Messquerschnitt oder einen Stau.

Objekttyp
Der Objekttyp eines Objekts legt dessen Eigenschaften und Beziehungen zu anderen Elementen des Datenmodells fest.

Attributgruppe
Die Attributgruppe fasst Attribute in Gruppen zusammen und legt so die Struktur von Datensätzen fest.

Attribut
Ein Attribut „ist eine Größe, die verschiedene Werte annehmen kann“ (vgl. Variable[2]).

Aspekt
Der Aspekt bezeichnet eine Sicht auf Daten, etwa, ob es sich um aggregierte Daten handelt.

Menge
Eine Menge enthält Referenzen auf Objekte.

2.1.2.1 Objekte

Ein Objekt repräsentiert einen bestimmten Gegenstand oder ein Konzept aus der realen Welt, z.B. einen bestimmten Messquerschnitt oder Stau. Die Eigenschaften und Struktur eines Objekts werden durch den ihm zugeordneten Objekttyp festgelegt. Ein Objekt ist also eine Instanz, d.h. eine konkrete Ausprägung eines Objekttyps. Jedes Objekt basiert auf genau einem Objekttyp.

Objekte werden nicht im HTML-Datenkatalog aufgeführt. Daher bieten sich GTM[3] und Datenmodellbrowser an, um Objekte eines bestimmten Objekttyps zu identifizieren.

Beispiel: Das Test-Objekt `fs.mq.a.10.0000.1üfs` ist vom Typ `typ.fahrStreifen`. Es hat daher genau die Eigenschaften dieses Objekttyps.

2.1.2.2 Objekttypen

Ein Objekttyp legt die Eigenschaften der Objekte fest, die auf ihm basieren. Ein Objekttyp kann als Abstraktion eines Gegenstandes oder Konzeptes der realen Welt verstanden werden. Beispielsweise basiert ein Objekt, das einen Stau repräsentiert, auf einem Objekttyp, der die grundlegenden Eigenschaften eines Staus, z.B. Länge, Dauer, usw., festlegt. Einem Objekttyp werden Attributgruppen und Mengen zugeordnet, um seine Eigenschaften zu beschreiben:

- Attributgruppen fassen zusammengehörige Werte zu einer Gruppe zusammen.
- Mengen enthalten Referenzen auf andere Objekte. Der Objekttyp legt fest, ob die Menge für die Objekte dieses Objekttyps *erforderlich* ist. Bei der Instanzierung eines Objekts muss eine Menge nur angelegt werden, wenn sie *erforderlich* ist.

Außerdem kann ein Objekttyp beliebig viele andere Objekttypen *erweitern*. Dadurch erbt er die Eigenschaften, also Mengen und Attributgruppen, von diesen Objekttypen. Ein Zyklus muss dabei ausgeschlossen werden, d.h. Konstrukte wie „Objekttyp A erweitert Objekttyp B und Objekttyp B erweitert Objekttyp A“ sind nicht erlaubt. Dieses Schema ist mit dem Konzept der Vererbung in objektorientierten Sprachen vergleichbar.

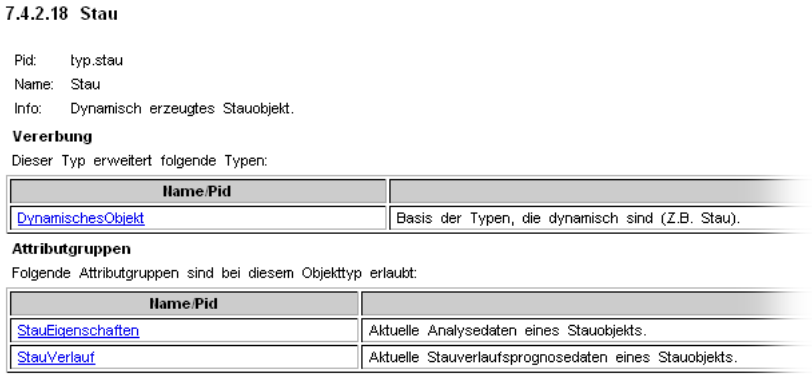
[2] http://de.wikipedia.org/wiki/Variable
[3] Dazu muss im Hauptfenster des GTM der entsprechende Objekttyp ausgewählt werden. Nach einer kurzen Wartezeit werden dann nur noch die Attributgruppen und Objekte angezeigt, die zu dem gewählten Objekttyp gehören.

Es existieren zwei Basistypen, die von allen anderen Objekttypen direkt oder indirekt erweitert werden:

Dynamischer Objekttyp (`typ.dynamischesObjekt`)

Objekte, die auf diesem Typ basieren, nennt man *dynamische Objekte.* Sie können zur Laufzeit erstellt und entfernt werden. Siehe dazu Abschnitt 2.1.3.3.

Beispiel: Der Objekttyp `typ.stau` erweitert den Objekttyp `typ.dynamischesObjekt` und ist daher ein dynamischer Objekttyp. Er erbt alle Eigenschaften dieses Objekttyps. Zusätzlich sind ihm die zwei Attributgruppen `StauEigenschaften` und `StauVerlauf` zugeordnet.



7.4.2.18 Stau

Pid: typ.stau
Name: Stau
Info: Dynamisch erzeugtes Stauobjekt.

Vererbung

Dieser Typ erweitert folgende Typen:

Name/Pid	
DynamischesObjekt	Basis der Typen, die dynamisch sind (Z.B. Stau).

Attributgruppen

Folgende Attributgruppen sind bei diesem Objekttyp erlaubt:

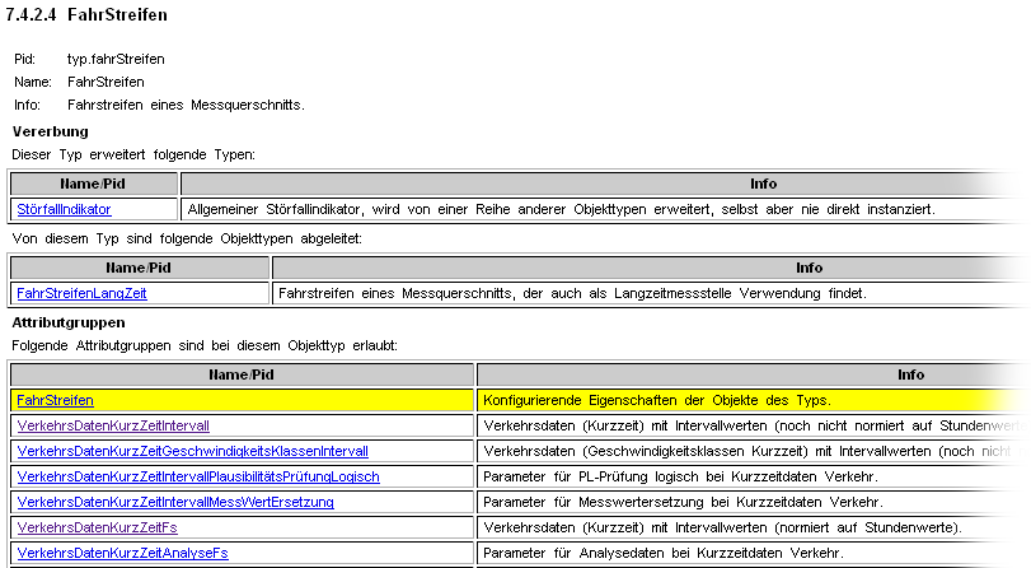
Name/Pid	
StauEigenschaften	Aktuelle Analysedaten eines Stauobjekts.
StauVerlauf	Aktuelle Stauverlaufsprognosedaten eines Stauobjekts.

Abbildung 2-2: typ.stau (HTML-Datenkatalog)

Konfigurations-Objekttyp (`typ.konfigurationsObjekt`)

Basistyp der Objekte, deren Lebenszyklus zur Laufzeit nicht beeinflusst werden kann. Die meisten Objekttypen des Datenmodells erweitern diesen Typ.

Beispiel: Der Objekttyp `typ.fahrStreifen`, der einen Fahrstreifen einer Straße modelliert, erweitert den Konfigurations-Objekttyp indirekt. Er besitzt verschiedene Attributgruppen, z.B. `FahrStreifen` und `VerkehrsDatenKurzZeitIntervall`. Der Objekttyp erweitert `StörfallIndikator`. Im HTML-Datenkatalog werden nur diejenigen Attributgruppen angezeigt, die dem Objekttypen zugeordnet wurden. Die Attributgruppen der Objekttypen, die erweitert werden, werden nicht angezeigt. Der Datenmodellbrowser kann jedoch alle Attributgruppen auflisten, die einem Objekttypen zugeordnet sind.



7.4.2.4 FahrStreifen

Pid: typ.fahrStreifen
Name: FahrStreifen
Info: Fahrstreifen eines Messquerschnitts.

Vererbung

Dieser Typ erweitert folgende Typen:

Name/Pid	Info
StörfallIndikator	Allgemeiner Störfallindikator, wird von einer Reihe anderer Objekttypen erweitert, selbst aber nie direkt instanziert.

Von diesem Typ sind folgende Objekttypen abgeleitet:

Name/Pid	Info
FahrStreifenLangZeit	Fahrstreifen eines Messquerschnitts, der auch als Langzeitmessstelle Verwendung findet.

Attributgruppen

Folgende Attributgruppen sind bei diesem Objekttyp erlaubt:

Name/Pid	Info
FahrStreifen	Konfigurierende Eigenschaften der Objekte des Typs.
VerkehrsDatenKurzZeitIntervall	Verkehrsdaten (Kurzzeit) mit Intervallwerten (noch nicht normiert auf Stundenwer
VerkehrsDatenKurzZeitGeschwindigkeitsKlassenIntervall	Verkehrsdaten (Geschwindigkeitsklassen Kurzzeit) mit Intervallwerten (noch nicht
VerkehrsDatenKurzZeitIntervallPlausibilitätsPrüfungLogisch	Parameter für PL-Prüfung logisch bei Kurzzeitdaten Verkehr.
VerkehrsDatenKurzZeitIntervallMessWertErsetzung	Parameter für Messwertersetzung bei Kurzzeitdaten Verkehr.
VerkehrsDatenKurzZeitFs	Verkehrsdaten (Kurzzeit) mit Intervallwerten (normiert auf Stundenwerte).
VerkehrsDatenKurzZeitAnalyseFs	Parameter für Analysedaten bei Kurzzeitdaten Verkehr.

Abbildung 2-3: typ.fahrStreifen (HTML-Datenkatalog)

2.1.2.3 Attributgruppen

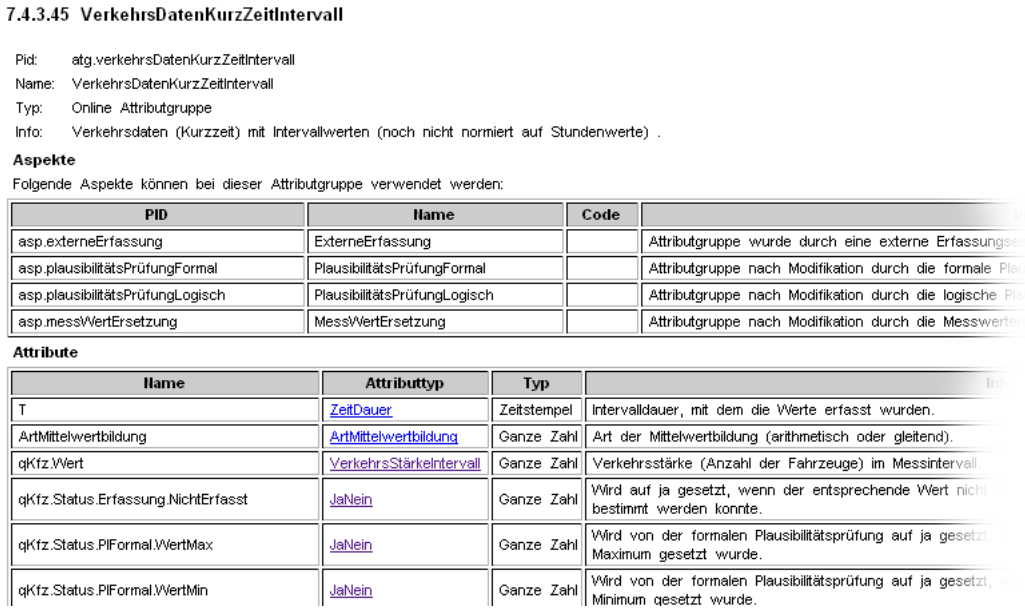
> **Wichtig**
>
> Entsprechend der Dokumentation zur Kernsoftware wird auf eine strenge Trennung von Typ (z.B. Attributtyp) und Instanz (Attribut) verzichtet. Wann Typ oder Instanz gemeint ist, erschließt sich aus dem Kontext:
>
> - Werden konkrete Daten oder Objekte verwendet, handelt es sich um *Instanzen*.
> - Andernfalls handelt es sich um *Typen*.

Attributgruppen legen die Struktur der Datensätze fest, die über den Datenverteiler transportiert werden. Da in der Verkehrstechnik oftmals bestimmte Werte in Beziehung zueinander stehen, werden diese Werte in einer Attributgruppe zusammengefasst. Die Werte werden in Attributen (siehe Abschnitt 2.1.2.4) gespeichert.

Jedem Objekttyp können beliebig viele Attributgruppen zugeordnet werden. Außerdem sind einer Attributgruppe beliebig viele Aspekte und Attribute zugeordnet — mindestens jedoch jeweils ein Aspekt und ein Attribut.

Eine Attributgruppe kann andere Attributgruppen nicht erweitern, d.h. eine Vererbung wie bei Objekttypen ist nicht möglich. Weiterhin legt eine Attributgruppe fest, ob ein Datensatz dieser Attributgruppe im Datenmodell abgelegt werden kann. So könnte z.B. der Name einer Straße als Datensatz im Datenmodell gespeichert werden. Man spricht dann von einer *konfigurierenden Attributgruppe* (siehe Abschnitt 2.1.4). Des Weiteren besteht die Möglichkeit, einen Datensatz einer Datenidentifikation in der SWE Parametrierung abzulegen. In diesem Fall spricht man von einer *parametrierenden Attributgruppe* (siehe Abschnitt 2.4).

Beispiel: Der Objekttyp `typ.fahrStreifen` besitzt die Attributgruppe `VerkehrsDatenKurzZeitIntervall`, welche diverse Verkehrsdaten zusammenfasst. Der Attributgruppe sind vier Aspekte zugeordnet, die ebenfalls im HTML-Datenkatalog aufgeführt werden.



7.4.3.45 VerkehrsDatenKurzZeitIntervall

Pid: atg.verkehrsDatenKurzZeitIntervall
Name: VerkehrsDatenKurzZeitIntervall
Typ: Online Attributgruppe
Info: Verkehrsdaten (Kurzzeit) mit Intervallwerten (noch nicht normiert auf Stundenwerte) .

Aspekte

Folgende Aspekte können bei dieser Attributgruppe verwendet werden:

PID	Name	Code	
asp.externeErfassung	ExterneErfassung		Attributgruppe wurde durch eine externe Erfassungs
asp.plausibilitätsPrüfungFormal	PlausibilitätsPrüfungFormal		Attributgruppe nach Modifikation durch die formale Pla
asp.plausibilitätsPrüfungLogisch	PlausibilitätsPrüfungLogisch		Attributgruppe nach Modifikation durch die logische Pla
asp.messWertErsetzung	MessWertErsetzung		Attributgruppe nach Modifikation durch die Messwerte

Attribute

Name	Attributtyp	Typ	
T	ZeitDauer	Zeitstempel	Intervalldauer, mit dem die Werte erfasst wurden.
ArtMittelwertbildung	ArtMittelwertbildung	Ganze Zahl	Art der Mittelwertbildung (arithmetisch oder gleitend).
qKfz.Wert	VerkehrsStärkeIntervall	Ganze Zahl	Verkehrsstärke (Anzahl der Fahrzeuge) im Messintervall.
qKfz.Status.Erfassung.NichtErfasst	JaNein	Ganze Zahl	Wird auf ja gesetzt, wenn der entsprechende Wert nicht bestimmt werden konnte.
qKfz.Status.PlFormal.WertMax	JaNein	Ganze Zahl	Wird von der formalen Plausibilitätsprüfung auf ja gesetzt, Maximum gesetzt wurde.
qKfz.Status.PlFormal.WertMin	JaNein	Ganze Zahl	Wird von der formalen Plausibilitätsprüfung auf ja gesetzt, Minimum gesetzt wurde.

Abbildung 2-4: atg.verkehrsDatenKurzZeitIntervall (HTML-Datenkatalog)

2.1.2.4 Attribute

Attribute können als Platzhalter für Werte aufgefasst werden[4]. Welche Werte in einem Attribut abgelegt werden können, legt der jeweilige Attributtyp fest. Diese Attributtypen können in zwei Gruppen unterteilt werden: Typen für *atomare* und *nicht-atomare* Attribute.

2.1.2.4.1 Atomare Attribute

Atomaren Attributen kann genau ein Wert zugeordnet werden. Dafür stehen fünf Grundtypen zur Verfügung. Jeder dieser Grundtypen hat bestimmte Eigenschaften, die bei der Definition eines Attributtyps festgelegt werden können:

Grundtyp	Eigenschaften	Beschreibung	Beispiel Attributtyp
Ganzzahl	Einheit	Einheit des Wertes kann als Zusatzinformation angegeben werden.	`att.zahl`
	Skalierungsfaktor	Skalierungsfaktor kann automatisch beim Berechnen des Attributwerts berücksichtigt werden.	
	Wertebereich	Der Wertebereich des Attributs kann festgelegt werden.	
	Zustand	Bestimmten Werten können Namen zugewiesen werden.	
Fließkommazahl	Einheit	Einheit des Wertes kann als Zusatzinformation angegeben werden.	`att.faktor`
	Genauigkeit	Die Genauigkeit (einfache oder doppelte) kann festgelegt werden.	
Text	Länge	Die maximale Länge der Zeichenkette kann beschränkt werden.	`att.text`
	Zeichenkodierung	Die Zeichenkodierung (z.B. ISO-8859-1) kann festgelegt werden	

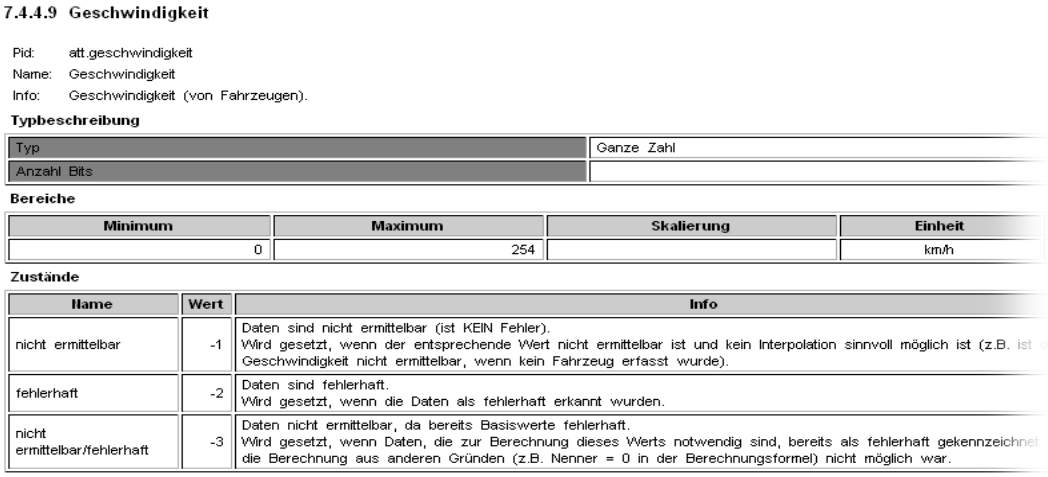
[4] Ein Attribut kann mit einer Variable verglichen werden.

Grundtyp	Eigenschaften	Beschreibung	Beispiel Attributtyp
Referenz	Typ	Es kann festgelegt werden, auf welche Objekte welchen Typs referenziert werden darf.	`att.objektReferenz`
	Verhalten	Das Verhalten der Objekte bei Änderungen am Datenmodell kann festgelegt werden, vgl. *Referenzierungsart*, Kapitel 5.1.2.4.4 in [TPUK]	
	Undefiniert	Undefinierte Referenzen zulassen oder verbieten.	
Zeitstempel	Genauigkeit	Festlegen der Genauigkeit auf Millisekunden oder Sekunden	`att.zeitstempel`
	Art	Art des Zeitstempels (relative Zeitspanne oder fester Zeitpunkt) festlegen.	

Tabelle 2-1: Verfügbare Grundtypen für Attribute

Beispiel:

- Der Attributtyp `att.geschwindigkeit` ist ein atomares Ganzzahl-Attribut mit folgenden Eigenschaften:



7.4.4.9 Geschwindigkeit

Pid: att.geschwindigkeit
Name: Geschwindigkeit
Info: Geschwindigkeit (von Fahrzeugen).

Typbeschreibung

Typ	Ganze Zahl
Anzahl Bits	

Bereiche

Minimum	Maximum	Skalierung	Einheit
0	254		km/h

Zustände

Name	Wert	Info
nicht ermittelbar	-1	Daten sind nicht ermittelbar (ist KEIN Fehler). Wird gesetzt, wenn der entsprechende Wert nicht ermittelbar ist und kein Interpolation sinnvoll möglich ist (z.B. ist d Geschwindigkeit nicht ermittelbar, wenn kein Fahrzeug erfasst wurde).
fehlerhaft	-2	Daten sind fehlerhaft. Wird gesetzt, wenn die Daten als fehlerhaft erkannt wurden.
nicht ermittelbar/fehlerhaft	-3	Daten nicht ermittelbar, da bereits Basiswerte fehlerhaft. Wird gesetzt, wenn Daten, die zur Berechnung dieses Werts notwendig sind, bereits als fehlerhaft gekennzeichnet die Berechnung aus anderen Gründen (z.B. Nenner = 0 in der Berechnungsformel) nicht möglich war.

Abbildung 2-5: Der Attributtyp att.Geschwindigkeit

Gültige Werte für Attribute dieses Typs liegen zwischen 0 und 254 (inklusive), zudem sind die Werte -1, -2 und -3 möglich. Die Einheit des Wertes ist „km/h“.

- Beim Attributtyp `att.grafikSkalierung` ist eine Skalierung eingestellt. Mit Hilfe des Java-API der Kernsoftware kann dann entweder mit dem „skalierten“ oder „unskalierten“ Attributwert gearbeitet werden.

- Referenzen werden verwendet, um auf andere Elemente des Datenmodells zu verweisen. So kann z.B. in den Eigenschaften eines Staus auf die Straße (siehe `atg.stauEigenschaften:Straße`) referenziert werden, auf der er aufgetreten ist.

2.1.2.4.2 Nicht-atomare Attribute

Neben den atomaren Attributen existieren die *nicht-atomaren* Attribute. Diese Attribute setzen sich aus atomaren und anderen nicht-atomaren Attributen zusammen, d.h. sie können mehrere Werte speichern. Nicht-atomare Attribute sind:

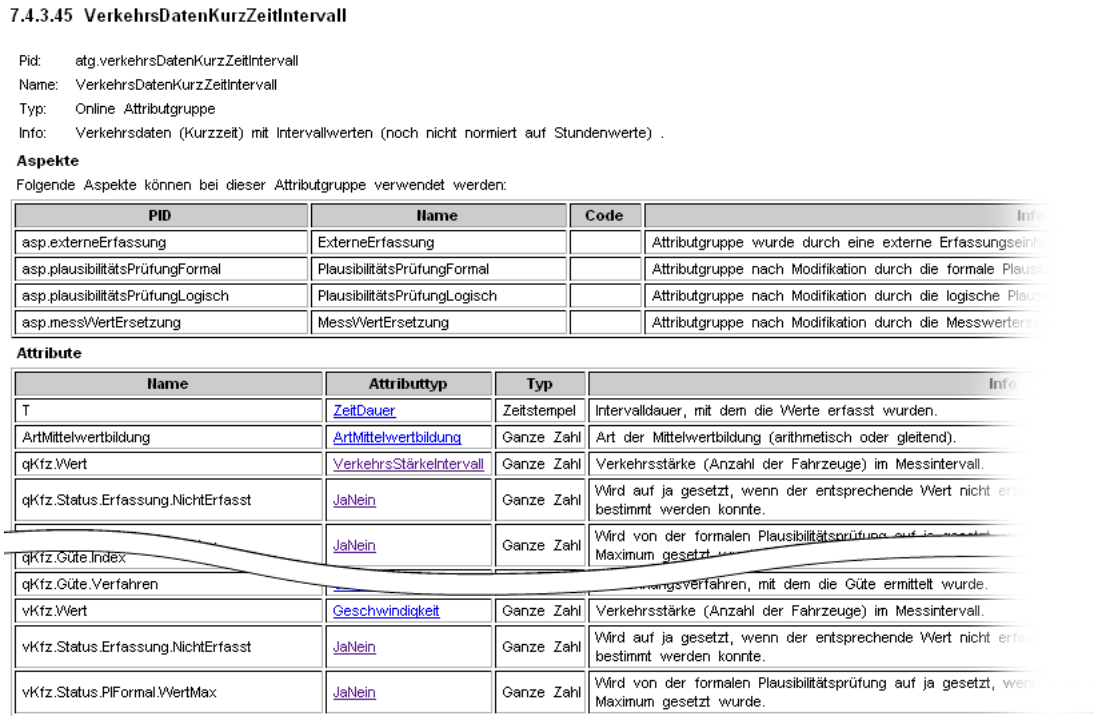
- *Felder*
  - Felder sind Reihen von Attributen des gleichen Typs. So können in einem Feld des Typs `att.geschwindigkeit` nur Werte abgelegt werden, die diesem Typ entsprechen.
  - Ein Feld kann eine variable oder feste Länge haben. Ist die Länge variabel, so kann die maximale Elementanzahl festgelegt werden.
- *Attributlisten*
  - Attributlisten sind Gruppen von beliebigen Attributen, welche von unterschiedlichem Typ sein können.

Tipp

- Innerhalb von Feldern und Attributlisten darf jeder beliebige Attributtyp verwendet werden. Beispielsweise kann ein Feld auf einer Attributliste basieren. Ebenso ist es möglich, Attributlisten zu verschachteln.
- Im HTML-Datenkatalog erkennt man Felder durch die eckigen Klammern hinter dem Attributnamen.
- Attributlisten und ihre Elemente werden im HTML-Datenkatalog durch Punkte getrennt.

Beispiel: Die Attributgruppe `atg.verkehrsDatenKurzZeitIntervall` enthält die Attributliste `qKfz` in der Werte bzgl. der Verkehrsstärke abgelegt werden. Im HTML-Datenkatalog gehören alle Attribute dieser Attributgruppe, deren Namen mit `qKfz` beginnt, zu der Attributliste `qKfz`. Bestandteile der Attributliste sind u.a. die beiden folgenden Attribute:

- `qKfz.Wert`: Das Attribut `Wert` ist ein atomares Attribut der Attributliste `qKfz`.
- `qKfz.Status.Erfassung.NichtErfasst`: Es handelt sich hierbei um eine verschachtelte Attributliste: `qKfz` enthält die Attributliste `Status`. Diese besitzt die Attributliste `Erfassung`, in welcher sich das atomare Attribut `NichtErfasst` befindet.



7.4.3.45 VerkehrsDatenKurzZeitIntervall

Pid: atg.verkehrsDatenKurzZeitIntervall
Name: VerkehrsDatenKurzZeitIntervall
Typ: Online Attributgruppe
Info: Verkehrsdaten (Kurzzeit) mit Intervallwerten (noch nicht normiert auf Stundenwerte) .

Aspekte

Folgende Aspekte können bei dieser Attributgruppe verwendet werden:

PID	Name	Code	Info
asp.externeErfassung	ExterneErfassung		Attributgruppe wurde durch eine externe Erfassungseinh
asp.plausibilitätsPrüfungFormal	PlausibilitätsPrüfungFormal		Attributgruppe nach Modifikation durch die formale Plaus
asp.plausibilitätsPrüfungLogisch	PlausibilitätsPrüfungLogisch		Attributgruppe nach Modifikation durch die logische Plaus
asp.messWertErsetzung	MessWertErsetzung		Attributgruppe nach Modifikation durch die Messwerters

Attribute

Name	Attributtyp	Typ	Info
T	ZeitDauer	Zeitstempel	Intervalldauer, mit dem die Werte erfasst wurden.
ArtMittelwertbildung	ArtMittelwertbildung	Ganze Zahl	Art der Mittelwertbildung (arithmetisch oder gleitend).
qKfz.Wert	VerkehrsStärkeIntervall	Ganze Zahl	Verkehrsstärke (Anzahl der Fahrzeuge) im Messintervall.
qKfz.Status.Erfassung.NichtErfasst	JaNein	Ganze Zahl	Wird auf ja gesetzt, wenn der entsprechende Wert nicht erf bestimmt werden konnte.
qKfz.Güte.Index	JaNein	Ganze Zahl	Wird von der formalen Plausibilitätsprüfung auf ja gesetzt, Maximum gesetzt
qKfz.Güte.Verfahren			rungsverfahren, mit dem die Güte ermittelt wurde.
vKfz.Wert	Geschwindigkeit	Ganze Zahl	Verkehrsstärke (Anzahl der Fahrzeuge) im Messintervall.
vKfz.Status.Erfassung.NichtErfasst	JaNein	Ganze Zahl	Wird auf ja gesetzt, wenn der entsprechende Wert nicht erf bestimmt werden konnte.
vKfz.Status.PlFormal.WertMax	JaNein	Ganze Zahl	Wird von der formalen Plausibilitätsprüfung auf ja gesetzt, wen Maximum gesetzt wurde.

Abbildung 2-6: Einige Attribute von atg.verkehrsDatenKurzZeitIntervall (HTML-Datenkatalog)

2.1.2.5 Aspekte

Aspekte ermöglichen die Interpretation der Werte, die in den Attributen gespeichert sind. Sie können als Bearbeitungszustände oder Sichten auf Attribute verstanden werden. Jede Attributgruppe besitzt einen oder mehrere Aspekte. Diese Aspekte dienen ausschließlich der logischen Klassifizierung von Datensätzen, sie haben keinen Einfluß auf die Struktur und Werte der Datensätze.

Beispiel: Ein Messquerschnitt sendet die Daten der Attributgruppe `atg.verkehrsDatenKurzZeitMq` im Minutentakt unter dem Aspekt `asp.agregation1Minute`. Eine Applikation, die diese Daten über 15 Minuten aggregiert, versendet die aggregierten Daten dann unter dem Aspekt `asp.agregation15Minuten`. Der Aspekt legt eben nicht die Struktur der Daten fest, sondern ermöglicht anderen Applikationen deren Interpretation.

Dabei ist zu beachten, dass dies nur eine logische Einteilung der Daten ist. Die oben genannte Aggregations-Applikation könnte die Daten auch unter dem Aspekt `asp.agregation60Minuten` versenden, obwohl die Daten alle 15, und nicht 60 Minuten aggregiert werden.

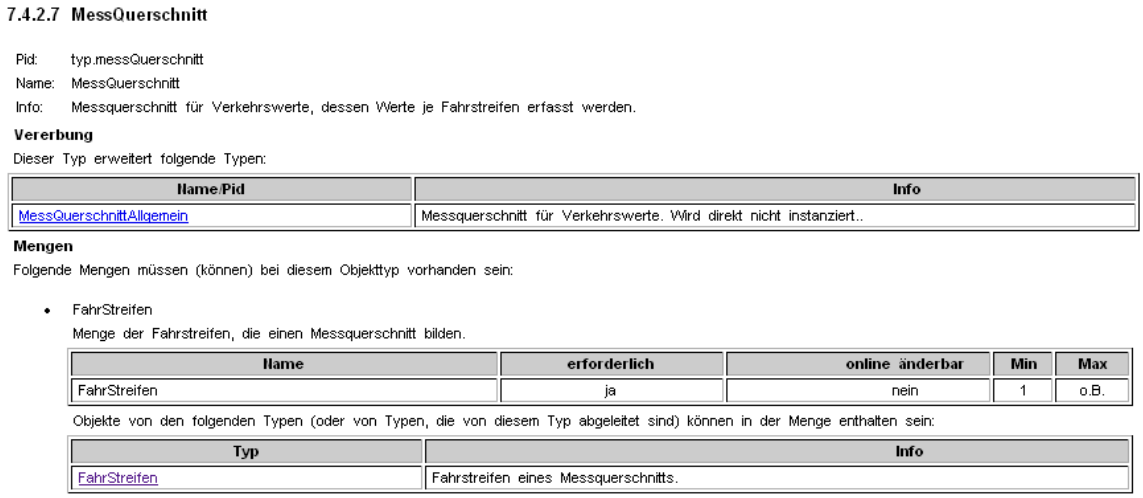
2.1.2.6 Mengen

Mengen sind Gruppen von Objekten. Dabei kann jede Menge ein Objekt genau einmal enthalten, ein Objekt kann jedoch Element mehrerer Mengen sein. Jeder Menge ist ein Mengentyp zugeordnet. Der Mengentyp legt fest, welche Objekte in der Menge abgelegt werden können. Dazu werdem dem Mengentyp Objekttypen zugeordnet. Jedes Objekt, das die zugeordneten Objekttypen direkt oder indirekt erweitert, kann in der Menge abgelegt werden. Welche Objekte zu einer Menge gehören, kann mit dem Datenmodellbrowser überprüft werden.

Außerdem ist es möglich, unterschiedliche Objekttypen anzugeben, die einander nicht erweitern, so dass Objekte unterschiedlichen Typs in der Menge abgelegt werden können. Des Weiteren können Mengen eine minimale und maximale Elementanzahl haben.

Beispiel: Objekte vom Typ `typ.messQuerschnitt` besitzen jeweils eine Menge mit Namen

`FahrStreifen`[5]. In dieser Menge werden alle Fahrstreifen (-Objekte) abgelegt, die den Messquerschnitt bilden:



7.4.2.7 MessQuerschnitt

Pid: typ.messQuerschnitt
Name: MessQuerschnitt
Info: Messquerschnitt für Verkehrswerte, dessen Werte je Fahrstreifen erfasst werden.

Vererbung

Dieser Typ erweitert folgende Typen:

Name/Pid	Info
MessQuerschnittAllgemein	Messquerschnitt für Verkehrswerte. Wird direkt nicht instanziert..

Mengen

Folgende Mengen müssen (können) bei diesem Objekttyp vorhanden sein:

- FahrStreifen

  Menge der Fahrstreifen, die einen Messquerschnitt bilden.

Name	erforderlich	online änderbar	Min	Max
FahrStreifen	ja	nein	1	o.B.

Objekte von den folgenden Typen (oder von Typen, die von diesem Typ abgeleitet sind) können in der Menge enthalten sein:

Typ	Info
FahrStreifen	Fahrstreifen eines Messquerschnitts.

Abbildung 2-7: typ.messQuerschnitt besitzt eine Menge (HTML-Datenkatalog)

- Zur Laufzeit sind keine Änderungen an der Menge möglich („online änderbar: nein“).
- Die Menge kann beliebig viele, mindestens jedoch ein Objekt enthalten („Min“, „Max“).
- Die Menge kann nur Objekte vom Typ `Fahrstreifen` (`typ.fahrStreifen`) enthalten.

2.1.2.7 Zusammenfassung

Die Elemente Objekt, Attributgruppe und Aspekt des Datenmodells bilden zusammen mit der Simulationsvariante die *Datenidentifikation*, die zum Senden und Empfangen von Daten über den Datenverteiler benötigt wird (vgl. Abschnitt 1.4.2 und Abschnitt 2.2.2.7). Dabei ist darauf zu achten, dass nur gültige Kombinationen verwendet werden: zusammen mit einem bestimmten Objekt dürfen nur diejenigen Attributgruppen angegeben werden, die dem Objekttyp des Objekts zugeordnet sind. Ebenso dürfen nur genau die Aspekte verwendet werden, die zu der gewählten Attributgruppe gehören.

Die übrigen Elemente, wie Attribute und Mengen, dienen dem Speichern von Daten: in Attributen können unterschiedliche Werte gespeichert werden, und diese können dann über den Datenverteiler transportiert oder im Datenmodell oder der SWE Parametrierung abgelegt werden. Im Gegensatz dazu enthalten Mengen immer nur Referenzen auf Objekte und werden ausschließlich im Datenmodell gespeichert.

2.1.3 Ausprägungen

> **Wichtig**
>
> Die folgenden Kapitel beziehen sich auf die Kernsoftware Stand 20.03.2006.

2.1.3.1 Darstellungsformen des Datenmodells

Das Datenmodell des Systems definiert die Struktur von und die Beziehungen zwischen Daten.

[5] Jedes Objekt erhält eine eigene Menge. D.h. obwohl die Mengen den gleichen Namen tragen, ist die Menge `FahrStreifen` von Objekt `mq.a10.0000` von der Menge `FahrStreifen` des Objekts `mq.a10.0001` unabhängig.

Die Kernsoftware verfügt über kein fest eingestelltes Datenmodell, d.h. es ist möglich, dass sich das Datenmodell mit der Zeit ändert. Neue Objekttypen können hinzukommen und nicht mehr benötigte Objekte können wieder entfernt werden. Um sowohl der Flexibilität des Datenmodells als auch der Anwenderfreundlichkeit Rechnung zu tragen, existiert das Datenmodell in drei verschiedenen physischen Ausprägungen:

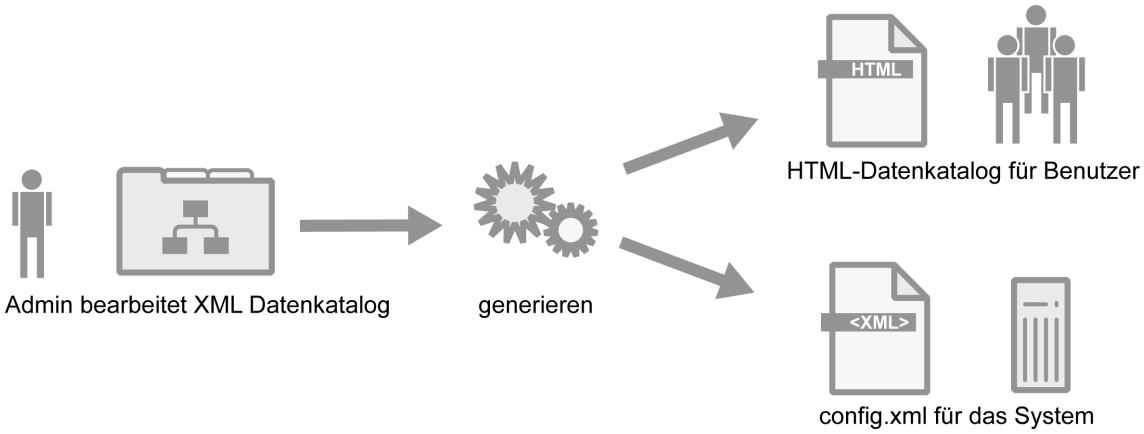


Abbildung 2-8: Ausprägungen des Datenmodells

(XML-) Datenkatalog

Der XML-Datenkatalog ist die Grundlage für alle Darstellungsformen des Datenmodells. Die übrigen Darstellungsformen können automatisch aus dem XML-Datenkatalog generiert werden. Der XML-Datenkatalog besteht aus allen XML-Dateien (*Versorgungsdateien*), die im Verzeichnis `datenkatalog` zu finden sind. Sollen Anpassungen am Datenmodell vorgenommen werden, so sind sie im XML-Datenkatalog umzusetzen. Wie bei Änderungen am XML-Datenkatalog vorgegangen werden muss, ist im Praxis-Teil des Tutorials beschrieben (Abschnitt 3.3).

HTML-Datenkatalog

Der HTML-Datenkatalog ist eine HTML-Datei, die das Datenmodell in einer gut lesbaren, übersichtlichen Darstellungsform enthält. Er kann mit einem beliebigen Web-Browser angezeigt werden und eignet sich daher besonders für Anwender, die sich einen Überblick über das Datenmodell verschaffen wollen. Allerdings enthält der HTML-Datenkatalog keine Objekte, sondern nur die Objekttypen, Attributgruppen und Attribute des Datenmodells. Um die Objekte anzuzeigen, muss man den GTM oder den Datenmodellbrowser verwenden.

Der HTML-Datenkatalog wird mit einem von der Firma *Kappich Systemberatung* entwickelten Werkzeug (das nicht frei verfügbar ist) aus dem XML-Datenkatalog erzeugt. Er gehört nicht zum Lieferumfang der Kernsoftware, sondern muss separat heruntergeladen[6] werden. Da der HTML-Datenkatalog nur zur Anzeige des Datenmodells dient, können darin keine Änderungen am Datenmodell vorgenommen werden.

config.xml

Aus dieser Datei liest die SWE Konfiguration das Datenmodell ein. Die `config.xml` wird oft als *Konfiguration* (entsprechend der SWE Konfiguration, die das Datenmodell zur Laufzeit verwaltet) bezeichnet. Sie kann durch ein mit der Kernsoftware ausgeliefertes Tool (`skripte-dosshell\DatenkatalogKonvertieren.bat` bzw. `skripte-bash/DatenkatalogKonvertieren.bash`) aus dem XML-Datenkatalog generiert werden. Änderungen am Datenmodell, die zur Laufzeit durch die SWE Konfiguration vorgenommen werden, schlagen sich nur in der `config.xml` nieder. Der XML-Datenkatalog wird von der SWE nicht verändert!

[6] http://zid.almo-traffic.de/uploads/media/DatK.zip

Warnung

Änderungen am Datenmodell sollten **nie** direkt in der `config.xml` vorgenommen werden, sondern nur im XML-Datenkatalog. Aus diesem wird dann die neue `config.xml` generiert.

Anmerkung

Änderungen, die am XML-Datenkatalog vorgenommen werden, werden erst nach Generieren der `config.xml` und Neustart der Kernsoftware berücksichtigt.

2.1.3.2 Konvertieren des Datenkatalogs

Änderungen des Datenmodells müssen im XML-Datenkatalog umgesetzt werden. Dieser XML-Datenkatalog kann dann zur `config.xml` konvertiert werden, denn nur in diesem Format kann die SWE Konfiguration das Datenmodell verarbeiten. Dies hat den Grund, dass in der `config.xml` noch zusätzliche Informationen gehalten werden: eine eindeutige ID (Identifikationsnummer)[7], die beim Konvertieren des Datenkatalogs erzeugt wird, und die momentan existierenden dynamischen Objekte.

Das Tool zum Konvertieren des Datenkatalogs (`skripte-dos-shell\DatenkatalogKonvertieren.bat` bzw. `skripte-bash/DatenkatalogKonvertieren.bash`) führt folgende Schritte beim Konvertieren des Datenkatalogs automatisch aus:

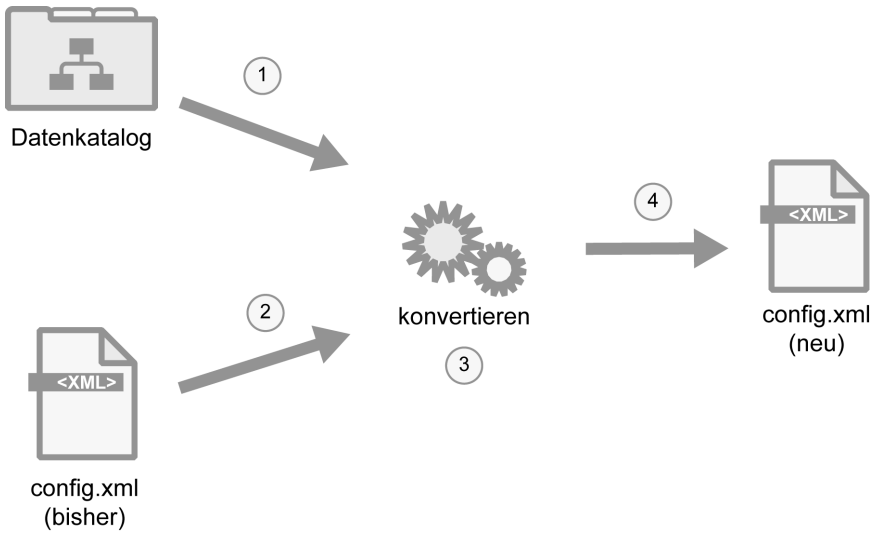


Abbildung 2-9: Konvertieren des Datemodells

1. Der XML-Datenkatalog wird eingelesen. Dabei werden fehlende Dateien bemängelt.
2. Die bisherige `config.xml` wird eingelesen, um das aktuelle Datenmodell und die bereits vergebenen IDs festzustellen.

[7] Diese ID ist mit der PID vergleichbar, bleibt aber auch über die Lebensdauer eines Elements hinaus erhalten. D.h. selbst nachdem das Element aus dem Datenmodell entfernt wurde, wird die ID nicht erneut vergeben. Die PID hingegen kann wieder vergeben werden.

3. Konvertierung

   a. Überprüfen, ob sich PID Konflikte aus dem XML-Datenkatalog und der gerade aktuellen `config.xml` ergeben.

   b. Vergabe von IDs für neu hinzugefügte Elemente. Elemente, die bereits in der `config.xml` vorhanden waren, behalten ihre IDs.

   c. Löschen von nicht mehr vorhandenen Elementen. Dabei werden deren PIDs freigegeben, die IDs hingegen nicht.

4. Schreiben des neuen `config.xml` Datenkatalogs.

Achtung

- Viele Applikationen, z.B. Archivsysteme nach [TANFARS], arbeiten mit den IDs von Objekten. Daher darf die `config.xml` auf keinen Fall manuell geändert oder vor der Konvertierung entfernt werden. Letzteres könnte dazu führen, dass sich die IDs ändern oder zur Laufzeit angelegte dynamische Objekte verloren gehen.
- Objekte behalten ihre IDs, selbst wenn ihre Eigenschaften (Attributgruppen und Mengen) geändert wurden. Die IDs von Attributgruppen können sich ändern, wenn sich die Struktur der Attributgruppen ändert.
- Die Kernsoftware sollte vor dem Konvertieren beendet werden.

2.1.3.3 Lebenszyklus von Objekten

Beim Lebenszyklus von Objekten müssen zwei Fälle unterschieden werden: der Lebenszyklus von nicht-dynamischen Objekten und der von dynamischen Objekten.

Nicht-dynamische Objekte:

Nicht-dynamische Objekte sind alle Objekte, die den Typ `typ.dynamischesObjekt` *nicht* erweitern. Diese Objekte können nur durch Anpassen des XML-Datenkatalogs erzeugt und entfernt werden, d.h. zur Laufzeit können diese Objekte nicht erzeugt oder gelöscht werden. Wird ein Objekt aus dem Datenkatalog entfernt, wird die PID wieder freigegeben. Damit die Änderungen wirksam werden, ist ein Beenden der Kernsoftware, Konvertieren des Datenkatalogs und anschließender Neustart der Kernsoftware nötig.

Dynamische Objekte:

Dynamische Objekte können zur Laufzeit angelegt und entfernt werden. Diese Änderungen werden sofort wirksam, d.h. mit den angelegten Objekten kann sofort gearbeitet werden, ohne dass ein Konvertieren des Datenkatalogs nötig ist. Wird ein dynamisches Objekt gelöscht, so tritt auch diese Änderung sofort in Kraft, d.h. das Objekt steht ab sofort nicht mehr zur Verfügung. Je nachdem welchen Objekttyp das dynamische Objekt erweitert, steht es auch nach einem Neustart der Kernsoftware zur Verfügung.

Werden Objekte zur Laufzeit angelegt oder entfernt, so wird dies durch die SWE Konfiguration in der `config.xml` vermerkt: Die jeweiligen dynamischen Objekte werden in die `config.xml` eingetragen oder daraus entfernt.

2.1.4 Im Datenmodell gespeicherte Daten

Das Datenmodell enthält neben Objekttypen, Attributgruppen, usw. auch konkrete Objekte

(siehe Abschnitt 2.1.2.1). Den Objekttypen dieser Objekte sind Attributgruppen zugeordnet. Zu jeder *konfigurierenden Attributgruppe* kann pro Objekt jeweils ein Datensatz im Datenmodell hinterlegt werden. Dieser Datensatz kann nur durch Anpassen des XML-Datenkatalogs geändert werden. Eine Änderung zur Laufzeit ist nicht möglich. Man nennt diese Datensätze *konfigurierende Daten*. Sie können z.B. vom Benutzer mit dem GTM (Abschnitt 1.5.3) eingesehen werden.

Beispiel: Für Objekte vom Typ `typ.straße` kann in der konfigurierenden Attributgruppe `Straße` der Straßentyp (Autobahn, Bundesstraße, usw.) und die Straßennummer festgelegt werden.

Die zu einer Menge zugehörigen Objekte (Abschnitt 2.1.2.6) können ebenfalls im Datenmodell festgelegt werden. Bei änderbaren Mengen können Elemente zur Laufzeit hinzugefügt oder entfernt werden, jedoch nur solange die minimale bzw. maximale Elementzahl eingehalten wird.

Neben diesen im Datenmodell gespeicherten Datensätzen existieren noch die in der SWE Parametrierung abgelegten Daten (siehe Abschnitt 2.4), und die Datensätze, die über den Datenverteiler transportiert werden:

- Die SWE Parametrierung verwaltet Systemeinstellungen. Die Datensätze werden dazu persistent gespeichert, können aber auch zur Laufzeit geändert werden (siehe Abschnitt 2.4).
- Die Datensätze, die über den Datenverteiler transportiert werden, sind die Nutzdaten des Systems. Sie werden nicht persistent gespeichert und werden vom Datenverteiler nur solange vorgehalten, bis ein anderer Datensatz der gleichen Datenidentifikation auf den Datenverteiler gelegt wird.

2.1.5 Organisationseinheiten

Wie in Abschnitt 2.1 beschrieben werden im Datenmodell der Kernsoftware verschiedene Konzepte (Metamodell, Typdefinitionen und Daten) zusammengefasst. Daher ist das Datenmodell zur besseren Wartbarkeit in verschiedene Organisationseinheiten unterteilt. Dabei ist zwischen einer rein *logischen Strukturierung* in unterschiedliche Dateien und einer *Strukturierung nach Verantwortlichkeit* zu unterscheiden:

- Die *logische Strukturierung* trennt zwischen dem Modell der Daten (Objekttypen, Attribute, usw) und den eigentlichen Daten (Objekte, konfigurierende Attributgruppen, Mengen).

  Beispiel: Objekttypen sollten in einer anderen XML-Datei des Datenkatalogs definiert werden, als die Objekte.

- Die *Strukturierung nach Verantwortlichkeit* legt fest, welche Änderungen am Datenmodell zur Laufzeit erlaubt sind, und wird durch *Konfigurationsbereiche* festgelegt.

  Beispiel: Alle Objekttypen, die von der VRZ Leverkusen benötigt werden, werden getrennt von den Objekttypen der VRZ Ludwigsburg verwaltet.

2.1.5.1 Logische Strukturierung

Das Modell der Daten wird in den *modellspezifischen Bereichen* definiert. Objekte werden in den *inhaltlichen Bereichen* instanziert.

Modellspezifische Bereiche:

Die *modellspezifischen Bereiche* legen die grundlegenden Eigenschaften des Datenmodells fest, so z.B. den Zusammenhang von Objekttypen und Attributgruppen. Zu den modellspezifischen Bereichen gehören:

Metamodell

Das *Metamodell* beschreibt den Aufbau des Datenmodells. Das Metamodell legt fest, dass es Objekttypen, Objekte, Attributgruppen, usw. gibt, und wie diese zusammenhängen. Das Metamodell ist somit die Grundlage für alle weiteren Definitionen des Datenmodells. Normalerweise besteht keine Notwendigkeit, das Metamodell wegen neuen SWE zu verändern.

Analogie: Syntax und Semantik der Programmiersprache Java.

In der Kernsoftware: Das Metamodell wird in der Datei `datenkatalog\allgemein\kb.metaModellGlobal.xml` definiert.

Systemdatenmodell

Im *Systemdatenmodell* werden alle das System betreffenden Typen definiert. Beispielsweise sind die Typen und Attributgruppen der Benutzerverwaltung in diesem Modell angesiedelt.

Analogie: Ein Java Package, welches die Klassen zur Verwaltung der Applikation enthält.

In der Kernsoftware: Das Systemdatenmodell wird in der Datei `...\kb.systemModellGlobal` festgelegt.

Fachdatenmodell

Alle Typ-Definitionen, die die fachliche Ebene des Systems betreffen, werden in diesem Modell definiert. Beispielsweise gehört die Typdefinition des Messquerschnitts (`typ.messQuerschnitt`) zum *Fachdatenmodell*.

Analogie: Ein Java Package, welches die Klassen enthält, die die Business-Logik einer Anwendung implementieren.

In der Kernsoftware: Das Fachdatenmodell umfasst mehrere (die bisher und im weiteren nicht genannten) Dateien des XML-Datenkatalogs.

Inhaltliche Bereiche:

Die in den modellspezifischen Bereichen definierten Objekttypen werden in den *inhaltlichen Bereichen* instanziert.

Systemobjekte

Alle Objekte, die das System betreffen (wie z.B. Benutzer zur Authentifizierung) werden in diesen organisatorischen Bereich aufgenommen.

Analogie: Ein Java Objekt einer Klasse zur Verwaltung der Applikation.

In der Kernsoftware: Die *Systemobjekte* werden in `...\kb.objekteSystemGlobal.xml` definiert.

Fachobjekte

Alle Objekte, die die Fachlichkeit betreffen (wie z.B. Messquerschnitte, Fahrstreifen) werden hier instanziert.

Analogie: Ein Java Objekt einer Business-Logik Klasse.

In der Kernsoftware: Die meisten bisher nicht genannten XML-Dateien des Verzeichnisses `datenkatalog\allgemein\` enthalten diese *Fachobjekte*.

Anmerkung

Die modellspezifischen und inhaltlichen Bereiche dienen ausschließlich der logischen Strukturierung.

2.1.5.2 Konfigurationsbereiche und Konfigurationsverantwortliche

Neben der rein logischen Strukturierung wird das Datenmodell durch die *Konfigurationsbereiche* gegliedert. In einem Konfigurationsbereich werden Typen und Objekte definiert und konfigurierende Daten abgelegt.

Anmerkung

- Nur die Konfigurationsbereiche haben Auswirkungen auf das Systemverhalten, die modellspezifischen und inhaltlichen Bereiche dienen nur der logischen Strukturierung.
- Die logische Strukturierung und Konfigurationsbereiche überschneiden einander: so kann ein Konfigurationsbereich sowohl modellspezifische als auch inhaltliche Bereiche enthalten. Ebenso können sich inhaltliche oder modellspezifische Bereiche über mehrere Konfigurationsbereiche erstrecken.

Jedem Konfigurationsbereich ist genau ein *Konfigurationsverantwortlicher*[8] zugeordnet, der für diesen Konfigurationsbereich zuständig ist. Zur Laufzeit sind dann nur Änderungen am Datenmodell möglich, die Konfigurationsbereiche betreffen, die demjenigen Konfigurationsverantwortlichen zugeordnet sind, mit dem die Kernsoftware gestartet wurde (siehe Abschnitt 3.1.1.2.2).

Im Auslieferungszustand der Kernsoftware werden die SWE Konfiguration und Datenverteiler mit dem Objekt `kv.system` als Konfigurationsverantwortlichem gestartet. Der XML-Datenkatalog oder der Datenmodellbrowser kann verwendet werden, um herauszufinden, zu welchem Konfigurationsbereich ein Typ oder Objekt gehört.

Neben diesen organisatorischen Aufgaben spielt der Konfigurationsverantwortliche auch bei der Kommunikation von Applikationen eine wichtige Rolle. Vielen SWE wird beim Start der Konfigurationsverantwortliche übergeben, den sie verwenden sollen. Über diesen Konfigurationsverantwortlichen werden dann bestimmte Datensätze versandt[9].

Beispiel: Wenn eine Archivanfrage an ein Archivsystem gesendet wird, wird in der Datenidentifikation das Objekt des Konfigurationsverantwortlichen verwendet. Der Grund hierfür ist, dass jede Applikation das Objekt des Konfigurationsverantwortlichen kennt. Im Gegensatz dazu ist das Objekt, das die Archivsystem-Applikation repräsentiert (Abschnitt 2.2.1), nicht jeder Applikation bekannt.

2.2 Datenverteiler

Die gesamte Kommunikation zwischen den auf der Kernsoftware beruhenden Applikationen, also auch den Kernsoftware-Komponenten selbst, wird über den Datenverteiler (DAV) abgewickelt. Die auf der Kernsoftware basierenden Applikationen dürfen nur den Datenverteiler verwenden, um Daten auszutauschen. Der Datenverteiler stellt somit die Schnittstelle zwischen allen Kernsoftware-Applikationen dar. Er ermöglicht die asynchrone Kommunikation der angeschlossenen Applikationen. Es ist möglich, mehrere Datenverteiler, die auf unterschiedlichen,

[8] Einem Konfigurationsverantortlichen können auch mehrere Bereiche zugeordnet sein.
[9] D.h. man verwendet den Konfigurationsverantwortlichen als Objekt in der Datenidentifikation.

über ein TCP/IP Netzwerk verbundenen Rechnern laufen, zu einem Datenverteilerverbund zusammenzuschalten (siehe Abschnitt 2.2.3).

Diese und die grundlegenden Eigenschaften des Datenverteilers werden in den nun folgenden Abschnitten vorgestellt.

2.2.1 Aufgaben

Der Datenverteiler übernimmt folgende Aufgaben:

- Transport der Daten.
  - Der Datenverteiler transportiert Daten nach dem publish/subscribe Verfahren: Er nimmt einen Datensatz einer Datenidentifikation entgegen, und übermittelt diesen an alle angemeldeten Empfänger.
  - Dabei wird immer der letzte Datensatz einer Datenidentifikation vorgehalten.

    Beispiel: Sendet eine Quelle je einen Datensatz um 14, 15 und 16 Uhr, und ein Empfänger meldet sich um 15:30 Uhr an, so erhält der Empfänger den Datensatz von 15 Uhr sobald er sich beim Datenverteiler anmeldet. Eine halbe Stunde später, um 16 Uhr, wird der nächste Datensatz zugestellt.
  - Die Datensätze einer Datenidentifikation treffen in genau der gleichen Reihenfolge beim Datenempfänger ein, in der sie von den Datenlieferanten auf den Datenverteiler gelegt werden.
  - Es werden immer alle Attribute einer Attributgruppe übertragen. Es ist nicht möglich, einzelne Attribute zu senden oder zu empfangen.
- Verwaltung der angemeldeten Sender und Empfänger. Dabei überprüft der Datenverteiler periodisch, ob die Kommunikationsteilnehmer noch existieren.
- Überprüfen, ob die Werte der Datensätze gültig sind, d.h. ob sie den verwendeten Attributtypen entsprechen. Anmerkung: Zum gegenwärtigen Entwicklungsstand (Kernsoftware-Release vom 20.03.2006) erlaubt der Datenverteiler jedoch noch die Übertragung von ungültigen Werten.
- Kommunikationsverbindung zu den anderen Kernsoftware-Komponenten.

  Damit eine Applikation auf die Kernsoftware-Komponenten zugreifen kann, muss sie sich zunächst mit dem Datenverteiler verbinden. Sobald eine Verbindung hergestellt wurde, erzeugt die SWE Konfiguration ein dynamisches Objekt, dass die Applikation repräsentiert. Dieses Objekt steht während der Laufzeit der Applikation zur Verfügung, und wird erst wieder entfernt, wenn die Verbindung zum Datenverteiler beendet wird. Dieses Applikationsobjekt vom Typ `typ.applikation` kann z.B. verwendet werden, um Daten zu senden, zu kennzeichnen, oder in der Parametrierung (Abschnitt 2.4) abzulegen.

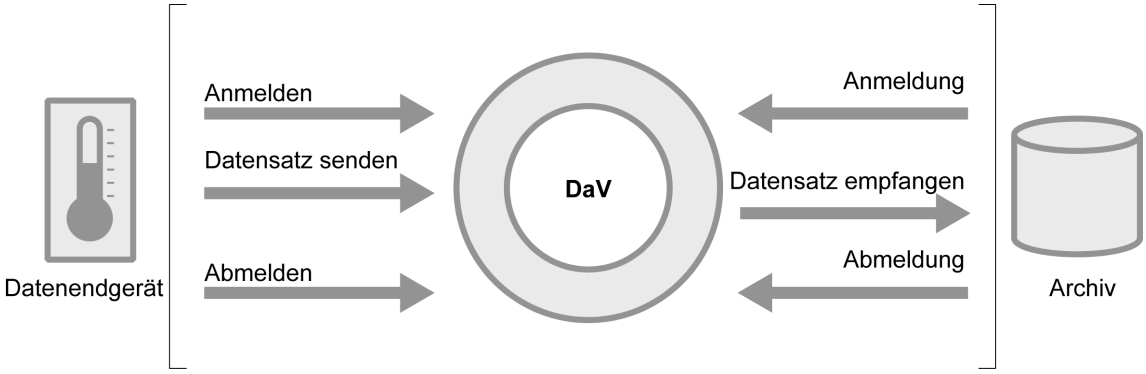


Abbildung 2-10: Der Datenverteiler ermöglicht den Transport von Daten

2.2.2 Versand von Daten

Der grundlegende Ablauf beim Senden und Empfangen von Daten und die zugrundeliegenden Konzepte werden in den folgenden Kapiteln beschrieben.

2.2.2.1 Simulationsvariante

Es existieren die Simulationsvarianten 0 bis 999. Datenendgeräte senden ihre Messdaten mit Simulationsvariante 0. Ebenso werden Datensätze, die zum Betrieb des Systems benötigt werden, mit Simulationsvariante 0 gesendet. Man nennt alle Datensätze, die mit dieser Simulationsvariante gesendet werden, Live-Daten.

Daten, die nur zu Simulationszwecken, z.B. zur Prognosenberechung, in das System eingespielt werden, haben eine Simulationsvariante zwischen 1 und 999. Die Simulationsvariante wird also verwendet, um die Zusammengehörigkeit von bestimmten Daten-Gruppen auszudrücken.

2.2.2.2 Datenidentifikation

Sollen Daten über den Datenverteiler transportiert werden, müssen Datenlieferant und Datenempfänger dies anmelden. Dazu teilen sie dem Datenverteiler das Objekt, die Attributgruppe, den Aspekt und die Simulationsvariante der Daten mit, die gesendet bzw. empfangen werden sollen. Diese Kombination von Objekt, Attributgruppe, Aspekt und Simulationsvariante nennt man *Datenidentifikation*.

Eine Datenidentifikation dient somit als Identifikation eines Kommunikationskanals. Alle Datenlieferanten und Datenempfänger, die miteinander kommunizieren möchten, müssen sich auf die gleiche Datenidentifikation anmelden.

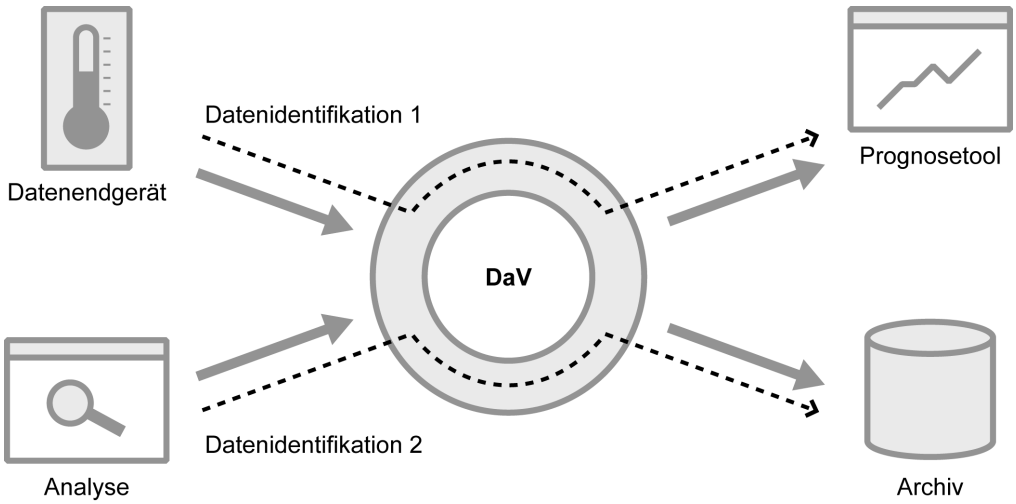


Abbildung 2-11: Die Datenidentifikation dient als Identifikation für einen Kommunikationskanal

Beispiel:

- `mq.a10.0000: VerkehrsDatenKurzZeitMq: Aggregation1Minute: 0` ist eine gültige Datenidentifikation. Sie wird vom Messquerschnitt `mq.a10.0000` verwendet, um echte (Simulationsvariante `0`) Verkehrsdaten (`VerkehrsDatenKurzZeitMq`), die über eine Minute aggregiert wurden (`Aggregation1Minute`) zu versenden. Ebenso kann diese Datenidentifikation verwendet werden, um diese Daten vom Messquerschnitt `mq.a10.0000` zu empfangen.
- `mq.a10.0000: VerkehrsDatenKurzZeitMq: Istwerte: 5` ist eine ungültige Datenidentifikation, da der Aspekt `Istwerte` nicht der Attributgruppe `VerkehrsDatenKurzZeitMq` zugeordnet ist.

2.2.2.3 Das Quelle / Senke Konzept

Bei der Anmeldung als Datenlieferant oder Datenempfänger muss eine Applikation zwei Parameter festlegen:

1. Datenidentifikation, für die die Anmeldung gilt
2. Für Datenlieferanten: Anmeldung als *Quelle* oder *Sender*

   Für Datenempfänger: Anmeldung als *Senke* oder *Empfänger*

Das publish/subscribe Konzept des Datenverteilers erlaubt es, dass sich beliebig viele Datenlieferanten und Datenempfänger für eine Datenidentifikation anmelden. Neben dem eigentlichen Transport der Datensätze muss der Datenverteiler gewissen Verwaltungsaufgaben, siehe Abschnitt 2.2.2.7, durchführen. Diese Verwaltungsaufgaben können nur von genau einem Datenverteiler übernommen werden. Deshalb gibt es genau **einen** Kommunikationsteilnehmer, der als Hauptlieferant bzw. Hauptabnehmer auftritt. Dieser Kommunikationsteilnehmer legt dann fest, welcher Datenverteiler die Verwaltungsaufgaben für diese Datenidentifikation übernimmt. Dies ist nötig, da mehrere Datenverteiler gekoppelt werden können (siehe Abschnitt 2.2.3). Den Datenverteiler, der die Verwaltungsaufgaben durchführt, bezeichnet man als *Zentraldatenverteiler* für die jeweilige Datenidentifikation. Er übernimmt sämtliche Verwaltungsaufgaben, die bei der Kommunikation über den Datenverteilerverbund anfallen.

Datenlieferanten:

Ein Datenlieferant kann sich entweder als Quelle oder als Sender beim Datenverteiler anmelden. Sowohl Quelle als auch Sender können Daten versenden. Die Verwaltungsaufgaben werden jedoch genau von demjenigen Datenverteiler übernommen, bei dem sich die Quelle angemeldet hat. Die Quelle legt somit fest, welcher Datenverteiler die Verwaltungsaufgaben durchführt.

Datenempfänger:

Analog gilt, dass sowohl Senke und Empfänger Daten empfangen, die Senke aber zusätzlich den Datenverteiler für Verwaltungsaufgaben festlegt.

> **Tipp**
>
> - Möchte man Daten senden, so meldet man sich üblicherweise als *Quelle* an.
> - Möchte man Daten empfangen, so meldet man sich üblicherweise als *Empfänger* an.
> - Neben einer Quelle kann es beliebig viele Sender und Empfänger geben. Analog ist es möglich, neben einer Senke mehrere Empfänger und Sender zu betreiben.

Beispiel: Ein Datenendgerät sendet seine Daten immer als *Quelle*, da es die Werte der Datensätze selbst festlegt, und diese Datensätze an viele verschiedene Applikationen gesendet werden sollen. Ein Archivsystem meldet sich auf diese Daten daher als *Empfänger* an. Das Archivsystem meldet sich außerdem als *Senke* für Archivanfragen an, da es dafür verantwortlich ist, diese Datensätze zu bearbeiten.

> **Wichtig**
>
> Für jede Datenidentifikation gibt es *genau eine* Quelle *oder eine* Senke! Eine Kommunikation zwischen Sendern und Empfängern ohne Quelle oder Senke ist nicht möglich.

Steht man vor der Entscheidung, ob man eine Quelle oder Senke verwenden soll, so kann nach folgenden Regeln vorgegangen werden:

- Werden die Datensätze üblicherweise von nur einer Applikation gesendet, so sollte diese als Quelle angemeldet werden.

  Beispiel: Viele Applikationen wie z.B. Archivsystem oder Prognoseberechnung müssen die Daten, die von einem Datenendgerät gesendet werden, empfangen. Daher meldet sich ein Datenendgerät als Quelle an.

- Werden Datensätze von mehreren Applikationen gesendet, aber nur von einer Applikation empfangen, so sollte letztere als Senke angemeldet werden.

  Beispiel: Mehrere Sender, etwa Unterzentralen und Verkehrsrechnerzentralen, möchten Daten gezielt an eine bestimmte Streckenbeeinflussungsanlage senden. Daher meldet sich eine Streckenbeeinflussungsanlage als Senke an.

2.2.2.4 Beispiel: Senden und Empfangen von Daten

Die folgende Abbildung zeigt eine mögliche Kommunikation von Applikationen über den Datenverteiler. Ein Datenendgerät sendet Messdaten als Quelle, da es der „Hauptverantwortliche“ für diese Daten ist. Neben dem Datenendgerät hat sich der GTM als Sender für die gleiche Datenidentifikation angemeldet, um zusätzliche Testdaten auf den Datenverteiler zu legen. Die zwei

Datenempfänger, Archiv und Prognosetool, wurden als Empfänger für die Datenidentifikation angemeldet.

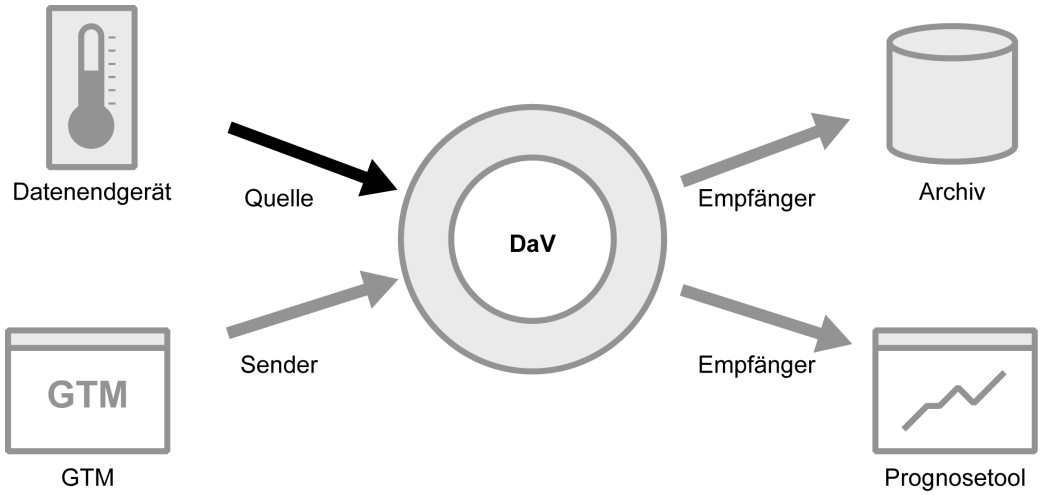


Abbildung 2-12: Eine Quelle legt Messdaten auf den Datenverteiler

Beide Datenlieferanten können unabhängig voneinander Daten senden. Es können noch weitere Sender angemeldet werden, jedoch keine weitere Quelle. Beide Empfänger erhalten jeweils alle Datensätze, die von den Datenlieferanten gesendet werden.

Die Empfänger haben keine Möglichkeit festzustellen, welcher Datenlieferant (ob Quelle oder Sender) gerade Daten sendet. Genauer: kein Kommunikationsteilnehmer kann feststellen, wieviele andere Teilnehmer existieren. Es kann nur festgestellt werden, ob es mindestens einen Datenlieferanten bzw. mindestens einen Datenempfänger (d.h. einen Empfänger oder eine Senke) gibt (vgl. folgendes Kapitel).

Das zweite Beispiel zeigt die Kommunikation von Applikationen mit dem Archivsystem. Das Archivsystem hat sich als Senke für Archivanfragen angemeldet, da es für die Beantwortung der Archivanfragen verantwortlich ist. Zusätzlich kann ein weiterer Empfänger, z.B. der GTM zu Testzwecken angemeldet werden. Ein Benutzer und das Analysetool müssen sich als Sender anmelden, um Archivanfragen zu stellen.

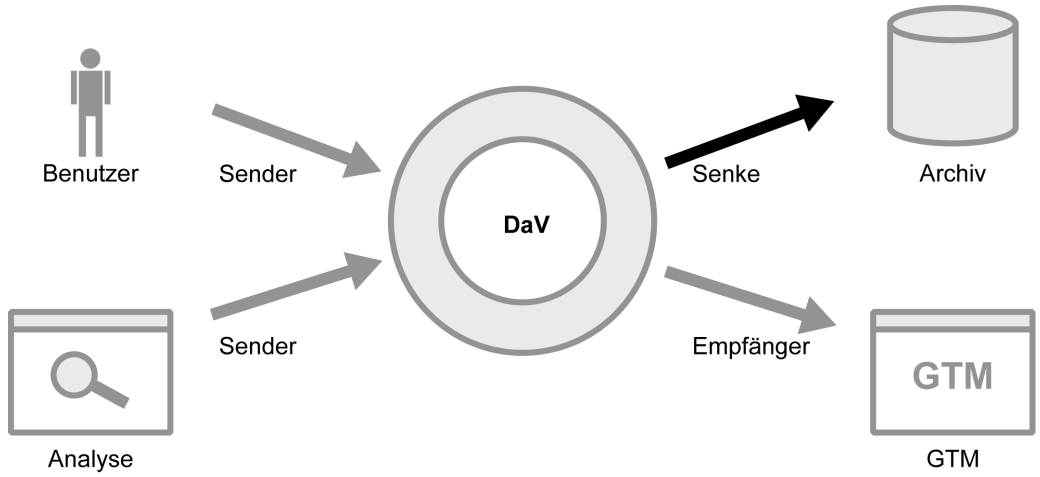


Abbildung 2-13: Das Archivsystem ist die Senke für Archivanfragen

2.2.2.5 Die Sendesteuerung

Der Datenverteiler verfügt über einen Mechanismus zur Flusskontrolle. Er informiert jeden angemeldeten Datenlieferanten, ob Daten momentan gesendet werden dürfen. Diese *Sendesteuerung* kennt vier Zustände:

1. *Positive Sendesteuerung:* Die Daten dürfen gesendet werden.

   Daten sollten erst gesendet werden, wenn der Datenverteiler dem Datenlieferanten eine positive Sendesteuerung meldet. Werden die Daten gesendet, bevor der Datenverteiler den Zustand der Sendesteuerung an die Applikation weitergibt, gehen sie u.U. verloren, ohne dass die sendende Applikation dies feststellen kann.

2. *Kein Abnehmer*: Die Daten sollen nicht gesendet werden, da kein Abnehmer existiert. Der Datenverteiler erlaubt trotzdem das Senden von Datensätzen.

   Daten sollten nur bei einer positiven Sendesteuerung gesendet werden. Eine Quelle darf die Sendesteuerung *Kein Abnehmer* jedoch ignorieren und trotzdem Daten auf den Datenverteiler legen. Der Grund hierfür ist, dass die Daten, die von einer Quelle gesendet werden, den Empfängern jederzeit zur Verfügung stehen sollen. Der Datenverteiler cached den jeweils zuletzt gesendeten Datensatz einer Datenidentifikation. Meldet sich ein Empfänger erst an, nachdem die Quelle bereits gesendet hat, kann der Datenverteiler ihm den Datensatz aus dem Cache zustellen.

   Beispiel: Eine Quelle meldet sich an und sendet einen Datensatz um 15 Uhr. Der nächste Datensatz wird um 16 Uhr gesendet. Meldet sich um 15:30 Uhr ein Empfänger an, so würde er keine Daten erhalten, falls die Sendesteuerung befolgt wurde. Ignoriert die Quelle die Sendesteuerung und legt trotz negativer Sendesteuerung einen Datensatz um 15 Uhr auf den Datenverteiler, kann dieser an den Empfänger weitergegeben werden, sobald er sich beim Datenverteiler anmeldet.

3. *Keine Rechte:* Die Daten dürfen nicht gesendet werden, da die Anwendung nicht die nötigen Zugriffsrechte besitzt. Das Senden wird vom Datenverteiler nicht zugelassen.

4. *Fehlerhafte Anmeldung:* Die Daten dürfen nicht gesendet werden, da bereits eine andere Quelle oder Senke für diese Datenidentifikation existiert. Das Senden wird vom Datenverteiler unterbunden.

Die Verwendung der Sendesteuerung in der Praxis ist in Abschnitt 3.5.3.2 zu finden.

> **Anmerkung**
>
> Neben dieser Sendesteuerung steht dem Datenverteiler noch ein weiterer Mechanismus zur Flusskontrolle zur Verfügung: Nimmt ein Empfänger die Datensätze nicht schnell genug vom Datenverteiler entgegen, so meldet der Datenverteiler den Empfänger automatisch ab.

2.2.2.6 Datenarten und Datenzeitstempel

Viele an den Datenverteiler angeschlossene Geräte sind Messgeräte, die Messdaten auf den Datenverteiler legen. Dabei vermerken die Messgeräte pro Datensatz den Zeitpunkt, zu dem die Messdaten erfasst wurden. Wird ein Gerät vom Datenverteiler getrennt, dann speichert es alle Datensätze und deren Zeitstempel lokal und sendet sie, sobald der Datenverteiler wieder zur Verfügung steht. Bei der späteren Verarbeitung der Messdaten ist es oft notwendig, zu wissen, ob die Daten sofort oder verzögert gesendet wurden. Daher markieren Datenlieferanten die Daten beim Senden als *online* oder *nachgeliefert*:

- Daten, die vom Datenlieferanten planmäßig gesendet werden, sind *online Daten*. Die Daten werden gesendet, sobald ein bestimmter Zeitpunkt erreicht wird, oder ein bestimmter, verarbeitungsabhängiger Zustand eintritt.

  Beispiel: Ein Messquerschnitt misst die Anzahl der PKW und sendet diese im Minuten- oder Stundenintervall.

- Werden Daten mit einer deutlichen Verzögerung gesendet, so sind diese als *nachgelieferte Daten* zu senden. Ebenso sind Daten mit Zeitrücksprüngen (d.h. Datensätze die einen kleineren Zeitstempel haben als der Vorgängerdatensatz) als nachgelieferte Daten zu kennzeichnen.

  Beispiel: Eine Quelle verliert die Verbindung zum Datenverteiler. Nach einer Stunde kann die Verbindung wieder hergestellt werden, und die Quelle sendet alle in der Zwischenzeit gesammelten Daten als nachgelieferte Daten.

> **Anmerkung**
>
> Die Datenart wird alleine vom Datenlieferanten festgelegt, d.h. jeder Sender bzw. jede Quelle kann frei entscheiden, was die jeweilige Datenart bedeutet und wann welche Datenart gewählt wird. Trotzdem sollte die obige Konvention eingehalten werden.

Neben den Datenarten *online* und *nachgeliefert* existieren die *nachgeforderten* Datenarten des Archivsystems. Datensätze werden vom Archivsystem als nachgefordert gekennzeichnet, wenn es diese Datensätze nicht direkt vom Datenlieferanten empfangen, sondern sie von einem anderen Archivsystem abgefragt hat. Dies ist z.B. der Fall, wenn ein Archivsystem eine Zeit lang vom Datenverteiler getrennt wird. Während dieser Zeitspanne kann das Archivsystem keine Daten empfangen. Wird die Verbindung zum Datenverteiler wiederhergestellt, kann das Archivsystem die „verpassten“ Datensätze von einem anderen Archivsystem nachfordern. Dabei sind zwei Fälle zu unterscheiden:

- Das Archivsystem kennzeichnet online Datensätze als *nachgefordert-aktuell*, falls diese nicht direkt vom Datenlieferanten empfangen wurden, sondern durch ein weiteres Archivsystem geliefert wurden.

- Analog werden nachgelieferte Datensätze vom Archivsystem als *nachgefordert-nachgeliefert* gekennzeichnet, wenn das Archivsystem diese von einem anderen Archivsystem abgefragt hat.

Weitere Informationen zu diesen Datenarten sind in Kapitel Abschnitt 3.2.1.2 und in [TANFARS] zu finden.

2.2.2.7 Datenindex

Sobald der Datenverteiler einen Datensatz eines Datenlieferanten entgegen nimmt oder selbst einen Datensatz erzeugt, wird diesem Datensatz noch der sog. *Datenindex* hinzugefügt. Der Datenindex wird von genau dem Datenverteiler vergeben, an den die jeweilige Quelle bzw. Senke angeschlossen ist.

Der Datenindex ist eine pro Datenidentifikation streng monoton steigende 64 Bit Ganzzahl mit folgenden Aufbau:

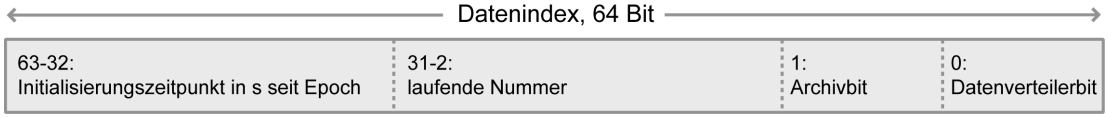


Abbildung 2-14: Aufbau des Datenindex

Die oberen 32 Bit werden durch einen Zeitstempel belegt, der anzeigt, wann sich die *Quelle* oder *Senke* beim Datenverteiler angemeldet hat[10]. Darauf folgt eine laufende Nummer, die bei jedem Sendevorgang beginnend mit dem Wert 0 um Eins inkrementiert wird. Das Archivbit wird vom Archivsystem gesetzt, wenn der Datensatz vom Archivsystem erzeugt wurde. Analog wird das Datenverteiler Bit gesetzt, wenn der Datensatz vom Datenverteiler erzeugt wurde. Dies geschieht in genau zwei Fällen:

- Der Datenverteiler erzeugt einen „keine Daten“ Datensatz, sobald sich eine Quelle anmeldet. Dieser Datensatz wird vom Datenverteiler erzeugt, wenn zwar ein Datenlieferant angemeldet wurde, aber dieser noch keine Daten gesendet hat.
- Meldet sich eine Quelle vom Datenverteiler ab, oder meldet sich ein Datenempfänger auf eine Datenidentifikation an, für die keine Quelle existiert, dann erzeugt der Datenverteiler einen „keine Quelle“ Datensatz.

2.2.2.8 Beispiel: Ablauf beim Senden und Empfangen von Daten

An einem Beispiel soll das Verhalten der Kernsoftware beim Senden und Empfangen von Daten gezeigt werden. Dieses Beispiel kann mit dem GTM nachvollzogen werden. Dazu können die in Abschnitt 1.5.2 und Abschnitt 1.5.10 beschriebenen Funktionen verwendet werden.

1. Ein Empfänger meldet sich beim Datenverteiler auf eine Datenidentifikation an, z.B. `mq.a10.0000 : atg.fundamentalDiagramm : asp.parameterIst : 0`.
2. Der Datenverteiler prüft, ob ein Datensatz für diese Datenidentifikation im Cache vorgehalten wurde. In diesem Fall wird dem Empfänger der Datensatz aus dem Cache zugestellt. Andernfalls sendet der Datenverteiler einen „keine Quelle“ Datensatz an den Empfänger.
3. Sobald sich die Quelle auf genau die gleiche Datenidentifikation anmeldet, erhält der Empfänger einen „keine Daten“ Datensatz vom Datenverteiler.
4. Die Quelle legt einen Datensatz auf den Datenverteiler.

[10] Dies kann zu Problemen führen, wenn der die Systemuhr zurückgestellt wird. In diesem Fall ist der Datenindex nach einem Neustart der Kernsoftware nicht zwingend streng monoton steigend!

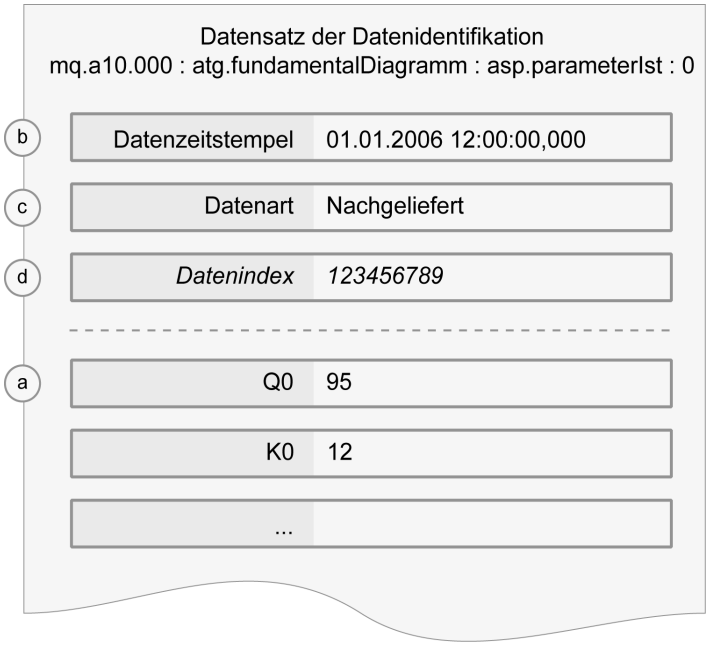


Abbildung 2-15: Zusammenstellen eines Datensatzes

a. Die Quelle stellt alle zu der angemeldeten Datenidentifikation gehörenden Daten, also alle Attribute der Attributgrupppe der Datenidentifikation, zusammen. Es können immer nur komplette Datensätze gesendet werden. Einzelne Attribute können nicht übertragen werden.

b. Die Quelle vergibt einen Zeitstempel für den Datensatz. Diesen Zeitstempel kann sie frei wählen.

c. Sie legt fest, ob es sich um *online* oder *nachgelieferte* Daten handelt. Üblicherweise senden Quellen online Daten. (siehe Abschnitt 2.2.2.6)

d. Die Quelle übergibt die in a - c erzeugten Daten dem Datenverteiler. Dieser fügt dem Datensatz den *Datenindex* (Abschnitt 2.2.2.7) hinzu.

5. Der Datensatz wird dem Empfänger durch den Datenverteiler zugestellt. Der jeweils zuletzt gesendete Datensatz wird vom Datenverteiler im Cache vorgehalten. Die Datensätze einer Datenidentifikation kommen in der Reihenfolge an, in der sie von der Quelle auf den Datenverteiler gelegt wurden.

6. Meldet sich ein weiterer Empfänger auf die gleiche Datenidentifikation an, so wird ihm sofort der im Datenverteiler-Cache vorgehaltene Datensatz zugestellt. Jeder weitere Datensatz der Quelle wird von nun an an beide Empfänger ausgeliefert.

7. Meldet sich die Quelle ab, so sendet der Datenverteiler einen „keine Quelle“ Datensatz an die Empfänger. Außerdem wird der Cache des Datenverteilers für diese Datenidentifikation geleert. Meldet sich ein weiterer Empfänger auf eben diese Datenidentifikation an, wird wie in Schritt 2 verfahren.

Tipp

Es spielt für die spätere Kommunikation keine Rolle, ob sich zuerst der Datenlieferant oder Datenempfänger anmeldet. Ebenso ist die Reihenfolge beim Abmelden irrelevant.

2.2.3 Koppeln von Datenverteilern

Einzelne Datenverteiler lassen sich zu einem Verbund zusammenschließen, so dass ein rechnerübergreifender Datenverteiler entsteht. Auf diese Weise kann die Last im System auf mehrere physikalische Rechner verteilt werden. Anwendungen, die an einem Datenverteiler aus einem Verbund angemeldet sind, können transparent auf alle im Verbund verfügbaren Resourcen zugreifen, ohne dass sie von dem Verbund wissen müssen.

Die folgende Abbildung zeigt, wie zwei Datenverteiler miteinander gekoppelt wurden. Das System arbeitet genauso, als ob alle Anwendungen an nur einen Datenverteiler angeschlossen wären. Das heißt, das Archiv kann die Daten des Datenendgeräts empfangen. Ebenso hat es Zugriff auf die SWE Parametrierung und Konfiguration.

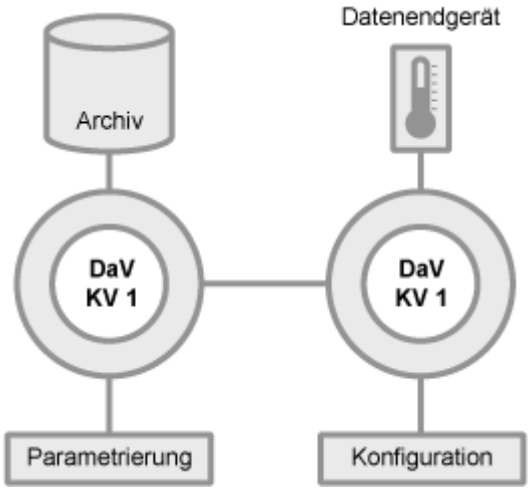


Abbildung 2-16: Einfacher Datenverteilerverbund

Es können auch Datenverteiler verbunden werden, die mit unterschiedlichen Konfigurationsverantwortlichen betrieben werden, wie die folgende Abbildung zeigt.

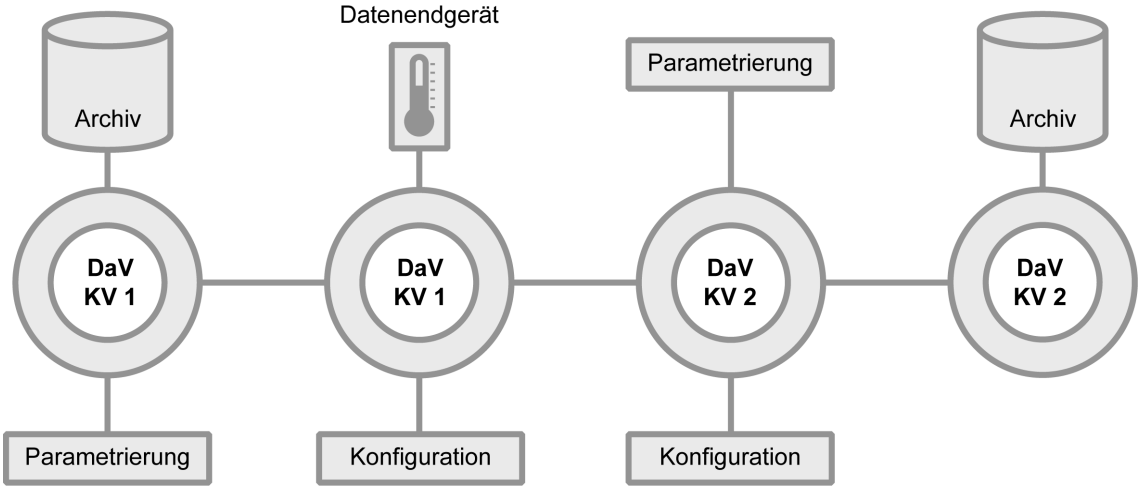


Abbildung 2-17: Datenverteilerverbund mit zwei Konfigurationsverantwortlichen

Werden Datenverteiler gekoppelt, muss dies bei der Definition der verwendeten Datenmodelle berücksichtigt werden. Jeder Datenverteiler verwendet einen eigenen `config.xml` Datenkatalog. Es muss sichergestellt werden, dass die PIDs und IDs der verwendeten Datenkataloge keine Konflike hervorrufen. Es besteht aber keine Notwendigkeit, dass alle Datenverteiler das glei-

che Datenmodell verwenden. So muss z.B. eine VRZ nicht alle Objekte kennen, die von den untergeordneten UZ verwendet werden, und umgekehrt. Zwar könnte ein „Einheitsdatenkatalog“ erzeugt werden, der von allen angeschlossenen Datenverteilern verwendet wird. Dieser wäre dann aber extrem umfangreich. So ist z.B. alleine die `config.xml` des Landes NRW bereits ca. 18 MB groß. Außerdem sind viele der Elemente des NRW-Datenkatalogs z.B. für das in Baden-Württemberg laufende Kernsystem nicht relevant.

Die Grundidee der Aufteilung des Datenmodells zeigt die folgende Abbildung:

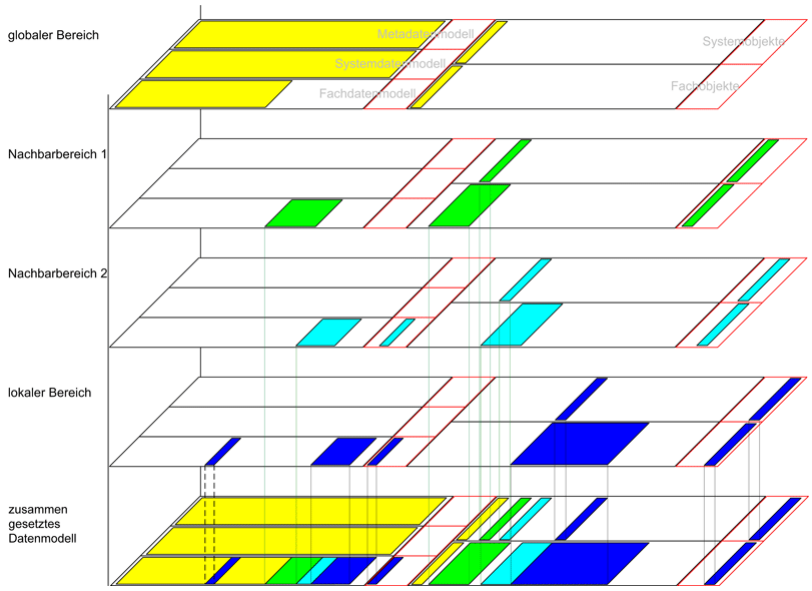


Abbildung 2-18: Hierarchisches Datenmodell[11]

Das Datenmodell, das von einem Datenverteiler verwendet wird, setzt sich aus mehreren Konfigurationsbereichen zusammen. Es wird eine Hierarchie über die Konfigurationsbereiche gebildet, anhand derer festgelegt wird, welche Definitionen in das zusammengesetzte, lokale Datenmodell übernommen werden, und welche durch andere Definitionen verschattet werden. In der obigen Abbildung wird beispielsweise fast der gesamte globale Bereich (gelb) in das lokale zusammengesetzte Datenmodell aufgenommen. Ein Teil dieser Definitionen wird jedoch durch Definitionen aus dem lokalen Bereich (blau) überschrieben.

> **Wichtig**
>
> Mit dem aktuellen Kernsoftware-Relase vom 20.03.2006 existieren keine Tools um eine Hierarchie über die Bereiche des Datenmodells zu erstellen.

2.3 Betriebsmeldungen

Betriebsmeldungen sind Textnachrichten, die über den Datenverteiler geschickt werden können. In der Regel versenden Server-Prozesse Betriebsmeldungen, wenn bestimmte Ereignisse eintreten oder ein Anwender reagieren muss. Da eine Betriebsmeldung vom Datenverteiler wie jeder andere Datensatz behandelt wird, kann sie auch archiviert oder mit dem Onlineprotokollierer angezeigt werden.

Die SWE Betriebsmeldungen ist eine einfache Implementierung einer Betriebsmeldungsverwaltung. Sie nimmt Betriebsmeldungen entgegen und publiziert diese wieder als Quelle. Außerdem kann sie die einzelnen Betriebsmeldungen in ihrer Logdatei speichern.

[11] Die Abbildung wurde aus [AFO], S. 237 leicht modifiziert übernommen.

2.4 Parametrierung

In der SWE Parametrierung können Einstellungen zentral hinterlegt werden. Dieser Vorgang wird „parametrieren“ genannt. Gespeicherte Einstellungen können zur Laufzeit geändert werden und Anwendungen können die hinterlegten Einstellungen über den Datenverteiler bei der SWE Parametrierung abfragen.

Damit eine Datenidentifikation überhaupt parametriert werden kann, muss sie eine parametrierende Attributgruppe (wird durch das Datenmodell festgelegt, siehe Abschnitt 2.1.2.3) enthalten. Zusätzlich muss für jede Datenidentifikation über die SWE Parametrierung explizit eingestellt werden, ob sie parametriert werden darf. Dazu dient die sog. *Grundparametrierung*.

Die Grundparametrierung ist ein spezieller Datensatz, für den immer Werte in der SWE Parametrierung hinterlegt werden dürfen. Er hat die Datenidentifikation `<Konfigurationsverantwortlicher>: atg.parametrierung: asp.soll: 0`. Üblicherweise ist der verwendete Konfigurationsverantwortliche `kv.system`, d.h. es muss die Datenidentifikation `kv.system: atg.parametrierung: asp.soll: 0` verwendet werden.

Alle Datenidentifikationen, die parametrierbar sein sollen, müssen in der Grundparametrierung eingetragen werden (siehe Abschnitt 3.1.2). Erst danach können sie wirklich parametriert werden.

2.5 Weiterführende Konzepte

In den vorangegangenen Abschnitten wurden alle für Benutzer und Softwareentwickler relevanten Grundlagen der Kernsoftware beschrieben. Es wurden jedoch zwei Themen ausgespart, die beim Betrieb der Kernsoftware beachtet werden müssen:

TLS-Konfiguration:

Um Daten von TLS Geräten (Streckenstationen, Datenendgeräte) empfangen zu können, muss eine TLS-Konfiguration vorgenommen werden. Dazu muss die TLS-Welt vollständig im Datenmodell abgebildet werden. Dies kann manuell (per Eingabe) oder automatisch mit dem Tool *ConfiGen*[12] der Firma *beck et al. projects*[13] geschehen. Eine manuelle TLS-Konfiguration ist nur bei sehr wenigen (<100) TLS-Geräten empfehlenswert, da für jedes TLS-Gerät ein Objekt mit konfigurierenden Daten im Datenkatalog eingetragen werden muss.

Berechtigungsmanagement:

Die Kernsoftware unterstützt ein detailiert konfigurierbares Berechtigungssystem. So kann z.B. für jeden Benutzer festgelegt werden, auf welche Datenidentifikationen er sich beim Datenverteiler anmelden kann. Standardmäßig hat ein Benutzer Zugriff auf alle Datenidentifikationen.

Das Anlegen und Verwalten dieser Berechtigungen ist jedoch zu umfangreich, um es in diesem Tutorial zu behandeln.

[12] http://configen.bea.de
[13] http://www.bea-projects.de

Kapitel 3 Kernsoftware Praxis

3.1 Systemeinstellungen

Im Schnelleinstieg, Kapitel 1, wurde schon kurz darauf eingegangen, wie die Kernsoftware zu starten ist, d.h. welche Skripte zum Starten der einzelnen Komponenten und Tools verwendet werden können. In diesem Kapitel soll nun näher erklärt werden, welche Möglichkeiten zur Einstellung die Skripte bieten.

Des Weiteren wird in diesem Kapitel darauf eingegangen, wie man mittels der Parametrierung Einstellungen am System vornehmen kann.

3.1.1 Startdateien anpassen

Die Kernsoftware enthält eine Reihe von Skripten, mit denen das System sowie die Tools gestartet werden können. Diese Skripte müssen unter Umständen noch den lokalen Gegebenheiten angepasst werden.

Alle nachfolgend erklärten Skripte finden sich im Installationsverzeichnis der Kernsoftwareinstallation (siehe Abschnitt 1.2). Die Skripte für Windows-Betriebssysteme sind im Unterverzeichnis `skripte-dosshell`, diejenigen für Linux-Betriebssysteme in `skripte-bash` abgelegt.

3.1.1.1 Linux- und Windows-Skripte

Es werden Skripte für die Betriebsysteme Windows und Linux mitgeliefert. Die Skripte unterscheiden sich nur in ihrer Syntax und Dateiendung. Der Aufbau ist jeweils gleich. Daher wird im Folgenden die Dateiendung nicht angegeben. Unter Linux ist sie `.bash`, unter Windows `.bat`.

Die Skripte sollten nur unter dem jeweiligen Betriebssystem bearbeitet werden, da sich die Textformate von Windows und Linux unterscheiden.

3.1.1.2 Die einzelnen Skripte

3.1.1.2.1 einstellungen

Dieses Skript wird von den anderen Skripten aufgerufen, um Umgebungsvariablen und globale Einstellungen zu initialisieren.

> **Wichtig**
>
> Änderungen an `einstellungen` haben Auswirkungen auf jedes der anderen Skripte, da es von allen nachfolgend beschriebenen Skripten vor der Verarbeitung des eigentlichen Skriptes ausgeführt wird.

Folgende wichtige Variablen werden in diesem Skript gesetzt:

JAVA_HOME
: Gibt den Pfad zur lokalen Java-Installation an. Die Variable muss nur angepasst werden, falls sie nicht global gesetzt oder der Pfad zur Java-Installation in der Systemvariable **PATH** nicht gesetzt ist.

benutzer
: Benutzername, unter dem sich Applikationen beim Datenverteiler standardmäßig authentifizieren.

dav1Host
: IP-Adresse oder Domainname des Standard-Datenverteilers, zu dem sich Applikationen verbinden.

dav1DavPort
Die Subadresse (bei TCP/IP der Port) des Datenverteilers für Verbindungen mit anderen Datenverteilern.

dav1AppPort
Der TCP-Port des Standard-Datenverteilers für aktive Verbindungen von Applikationen zum Datenverteiler.

dav2Host, dav2DavPort, dav2AppPort
Analog zu den Variablen **dav1Host**, **dav1DavPort**, **dav1AppPort** spezifizieren diese Variablen Kopplungs-Einstellungen für einen weiteren Datenverteiler.

passwortDatei
Gibt den vollständigen Dateinamen (inkl. Pfad) der Datei mit den Passwortinformationen des Benutzers für die Authentifizierung beim Datenverteiler an.

cp
Spezifiziert den *Classpath*, d.h. den Pfad für die Java Virtual Machine, unter der nach übersetztem Java-Code gesucht wird.

jvmArgs
Enthält weitere Argumente (durch Leerzeichen getrennt), die der Java Virtual Machine beim Starten übergeben werden. Dies sind z.B.:

- `-Xms[zahl]m`

  Gibt an, wieviel Heapspeicher von der Java Virtual Machine beim Start angefordert wird. [zahl] steht hierbei für den Wert in Megabyte.

- `-Xmx[zahl]m`

  Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. `[zahl]` steht hierbei wiederum für den Wert in Megabyte.

- `-Dfile.encoding`

  Gibt an, in welcher Kodierung die JVM Lese- und Schreiboperationen durchführt.

3.1.1.2.2 KernsoftwareSystem

Dieses Skript startet die Kernsoftware (siehe auch Abschnitt 1.3) inklusive aller wichtigen Komponenten. Dazu gehören:

- *Datenverteiler*

  Aufrufparameter, die beim Start des Datenverteilers übergeben werden, sind:

  -Xmx[zahl]m
  Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

  -rechtePruefung=[ja | nein]
  Rechteprüfung im Datenverteiler an- und abschalten. Wenn die Rechteprüfung eingeschalten ist, müssen Zugriffsrechte korrekt parametriert werden.

  -debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose vom Datenverteiler auf Standard-Error ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom Datenverteiler in das Log geschrieben werden. Siehe Abschnitt 3.6.

- *Konfiguration*

  Startet die Konfiguration mit den Aufrufparametern:

  -Xmx[zahl]m
  Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

  -datenbank=[dateiname]
  Gibt den vollständigen Dateinamen der `config.xml`-Datei, welche die Konfigurationsdaten im XML-Format enthält, an.

  -debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose von der SWE Konfiguration ausgegeben werden. Siehe

  -debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose von der SWE Konfiguration in das Log geschrieben werden. Siehe Abschnitt 3.6.

- *Parametrierung*

  Startet die Parametrierung mit den folgenden Parametern:

  -Xmx[zahl]m
  Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

  -sleep=[zahl]
  Zeit in Millisekunden, die gewartet wird, bevor die Verbindung zum Datenverteiler aufgebaut wird.

  -parameterVerzeichnis
  Verzeichnis, in dem die Parametrierung ihre Einstellungen persistent speichert.

  -debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose von der Parametrierung ausgegeben werden. Siehe Abschnitt 3.6.

  -debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose von der Parametrierung in das Log geschrieben werden. Siehe Abschnitt 3.6.

- *Betriebsmeldungen*

  Startet das Modul zur Betriebsmeldungsverwaltung. Folgende Parameter sind möglich:

  -debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
  Gibt an, welche Informationen zur Fehlerdiagnose von der Betriebsmeldungsverwaltung

ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose von der Betriebsmeldungsverwaltung in das Log geschrieben werden. Siehe Abschnitt 3.6.

3.1.1.2.3 GenericTestMonitor

Das Skript startet den GTM. Zur Verwendung des GTM sei auf Abschnitt 1.5 und Abschnitt 3.2 verwiesen. Folgende Parameter können zum Starten notwendig sein:

-Xmx[zahl]m
Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

-autologin
Ist dieser Parameter angegeben, werden die Parameter `-benutzer`,`-authentifizierung` und `-datenverteiler` ausgewertet. Andernfalls werden die Parameter ignoriert und ein Loginfenster gezeigt.

-debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom GTM ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom GTM in das Log geschrieben werden. Siehe Abschnitt 3.6.

3.1.1.2.4 DatenkatalogKonvertieren

Dieses Skript ermöglicht die Validierung und Konvertierung des XML-Datenkatalogs. Ausgabe des Skriptes ist die für den Start des Datenverteilers notwendige Datei `config.xml`. Zunächst wird die Validierung angestoßen und das Skript abgebrochen, sofern sie nicht erfolgreich ist. Als nächstes wird die Konvertierung gestartet (siehe auch Abschnitt 2.1.3.2). Folgende Parameter sind dabei notwendig, bzw. hilfreich:

-Xmx[zahl]m
Überschreibt den im Skript `einstellungen` angegebenen Wert für den maximal für die Anwendung zur Verfügung stehenden Heapspeicher.

-datenkatalog=[dateiname]
Gibt den vollständigen Dateinamen des XML-Datenkatalogs an.

-datenkatalogAnzeigen=[ja | nein]
Gibt an, ob der Datenkatalog während der Konvertierung in der Konsole angezeigt werden soll.

-ergebnis=[dateiname]
Spezifiziert den Dateinamen der zu schreibenden XML-Datei.

-ergebnisAnzeigen=[ja | nein]
Gibt an, ob die Ergebnisdatei auf der Konsole ausgegeben werden soll.

-debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]

Gibt an, welche Informationen zur Fehlerdiagnose vom Skript `DatenkatalogKonvertieren` ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom Skript `DatenkatalogKonvertieren` in das Log geschrieben werden. Siehe Abschnitt 3.6.

3.1.1.2.5 KonfigBrowserOnline

Dieses Skript startet die Online-Variante des Datenmodelbrowsers. Das Datenmodel wird dabei über den Datenverteiler aus der SWE Konfiguration geladen. Näheres dazu ist in Abschnitt 3.2.3 zu finden.

-Xmx[zahl]m
Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

-debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom `KonfigBrowserOnline` ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom `KonfigBrowserOnline` in das Log geschrieben werden. Siehe Abschnitt 3.6.

3.1.1.2.6 KonfigBrowserOffline

Diese Skript startet die Offline-Variante des Datenmodellbrowsers. Das Datenmodel wird dabei direkt aus der `config.xml` ausgelesen. Näheres dazu ist in Abschnitt 3.2.3 zu finden.

-Xmx[zahl]m
Gibt an, wieviel Heapspeicher von der Java Virtual Machine maximal angefordert wird. Wenn der Parameter hier angegeben wird, überschreibt er den Standardwert aus `einstellungen`.

-konfiguration=[dateiname]
Gibt den Pfad zu einer config.xml-Datei an, die die Konfiguration enthält, die vom Modellbrowser dargestellt werden soll.

-debugLevelStdErrText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom `KonfigBrowserOffline` ausgegeben werden. Siehe Abschnitt 3.6.

-debugLevelFileText=[ERROR | WARNING | INFO | CONFIG | FINE | FINER | FINEST | NONE]
Gibt an, welche Informationen zur Fehlerdiagnose vom `KonfigBrowserOffline` in das Log geschrieben werden. Siehe Abschnitt 3.6.

3.1.2 Parametrierung einstellen

Um Datenidentifikationen zu parametrieren, wird das Tool *Parameter editieren* des GTM verwendet. Damit kann auch die Grundparametrierung (siehe Abschnitt 2.4) eingestellt werden. Im Folgenden wird beschrieben, wie eine Datenidentifikation parametriert werden kann. Dazu wird

exemplarisch die Grundparametrierung parametriert, d.h. es wird eingestellt, welche weiteren Datenidentifikationen parametriert werden dürfen.

Beispiel: Die Datenidentfikation `kv.system: atg.archivContainer: asp.parameterSoll: 0` soll parametrierbar gemacht werden.

1. Auswählen der Datenidentfikation der Grundparametrierung im GTM Hauptfenster. Die Datenidentifikation der Grundparametrierung ist abhängig vom Konfigurationsverantwortlichen, der verwendet werden soll. Üblicherweise kann `kv.system: atg.parametrierung: asp.soll: 0` verwendet werden.

2. Starten des *Parameter editieren* Tools.

3. Eintragen des Objekts, der Attributgruppe und der Simulationsvariante der Datenidentifikation, die parametrierbar gemacht werden soll. Der Aspekt der Datenidentifikation wird nicht eingestellt. Die weiter unten folgende Abbildung zeigt die vorzunehmenden Einstellungen.

4. Durch Klicken auf den *Senden*-Button wird die Einstellung durch die SWE Parametrierung gespeichert. Die Datenidentfikation `kv.system: atg.archivContainer: asp.parameterSoll: 0` ist nun parametrierbar.

Um die Datenidentfikation `kv.system:atg.archivContainer:asp.parameterSoll:0` dann tatsächlich zu parametrieren, kann analog vorgegangen werden. In Schritt 1 wird jedoch nicht die Datenidentfikation der Grundparametrierung verwendet, sondern die der zu parametrierenden Datenidentifikation, hier `kv.system: atg.archivContainer: asp.parameterSoll: 0`.

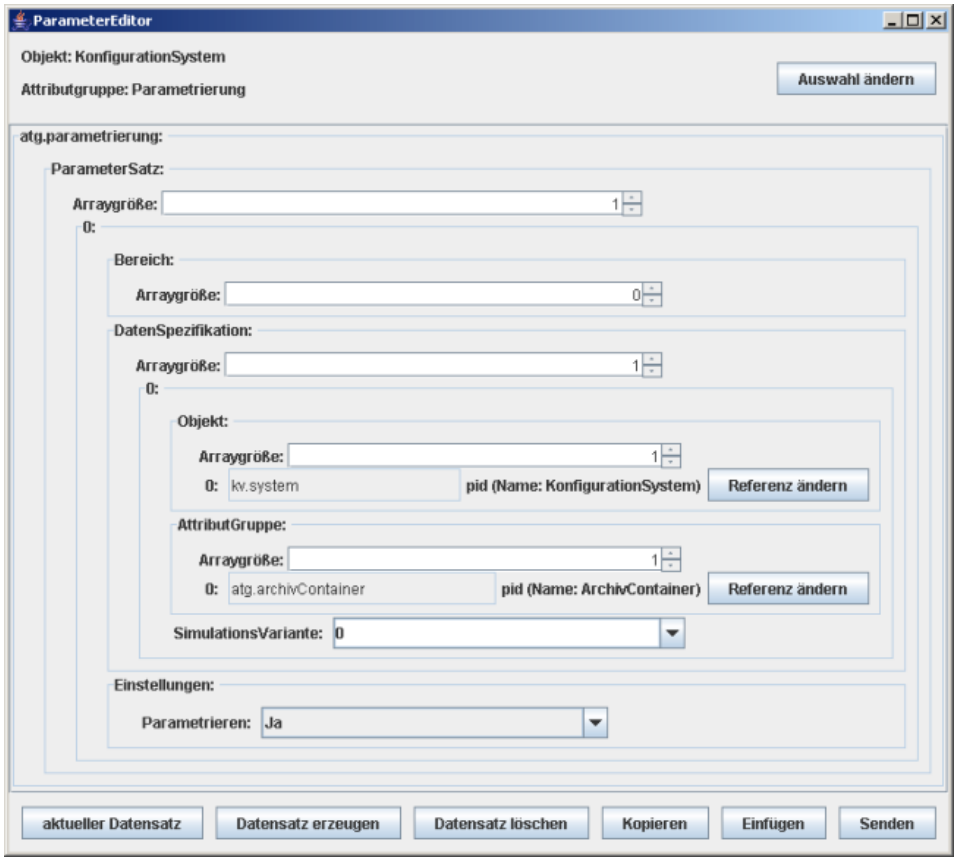


Abbildung 3-1: ParameterEditor

3.2 Kernsoftwaretools

3.2.1 GTM

Der Generische Testmonitor ist ein Werkzeug, das verschiedene nützliche Kernsoftwaretools in einer grafischen Bedienoberfläche zusammenfasst. Grundlegende Funktionen des GTM sind in Abschnitt 1.5 zu finden, detailliertere Erklärungen finden sich in den nun folgenden Abschnitten.

Wählen der Datenidentifikation(en)

Um die Tools (siehe Abbildung 3-2) starten zu können, muss zunächst mindestens eine Datenidentifikation gewählt werden. Dazu kann aus den Listen

- Objekttyp,
- Attributgruppe und
- Aspekt

jeweils ein Wert gewählt und die *Simulationsvariante* gesetzt werden. In der Liste *Objekte* ist ein Objekt oder eine Auswahl von Objekten zu selektieren. Wird mehr als ein Objekt gewählt, sind die Tools *Aktuelle Daten anzeigen* und *Aktuelle Daten senden* nicht verfügbar, da sie nur mit einem Objekt arbeiten können. Es ist jedoch möglich die Tools einzeln hintereinander mit unterschiedlichen Datenidentifikationen zu starten.

Ist die Datenidentifikation gewählt, so können die verschiedenen Kernsoftwaretools durch Anklicken des entsprechenden Buttons in der rechten Fensterhälfte des GTM gestartet werden.

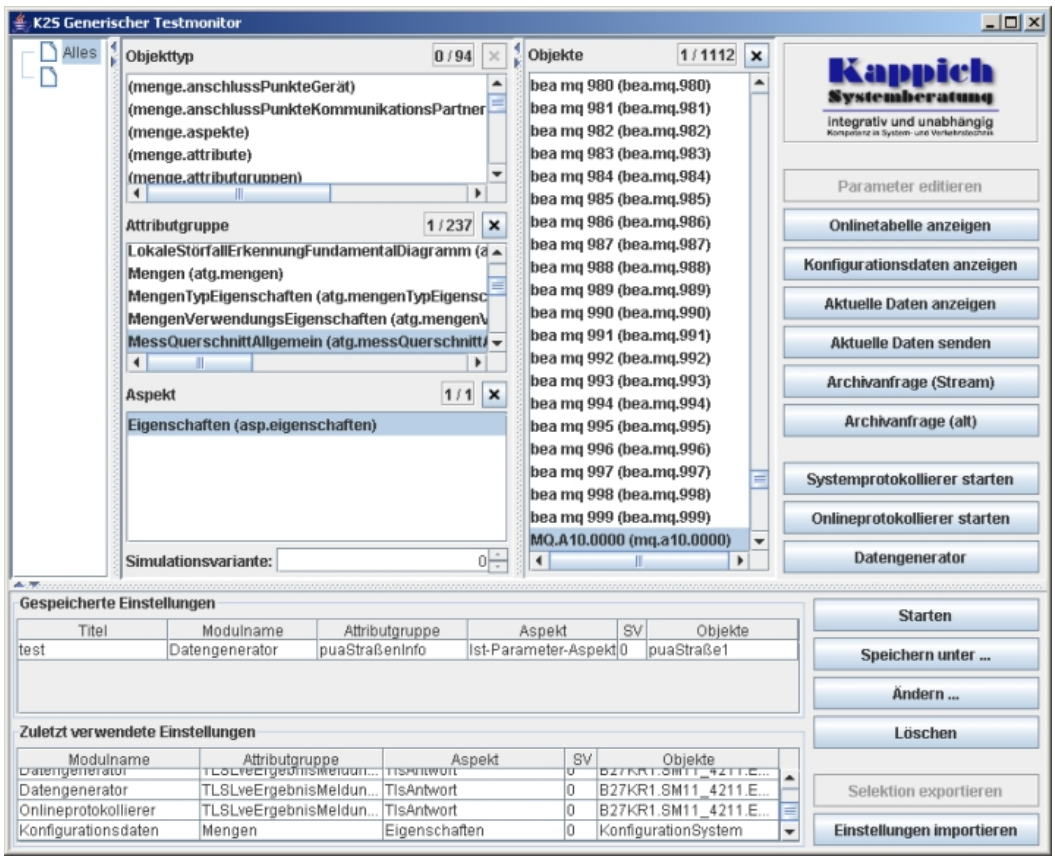


Abbildung 3-2: Generischer Testmonitor

Wenn ein Kernsoftwaretool gestartet wird, merkt sich der GTM die verwendeten Einstellungen

und nimmt sie in die Tabelle *Zuletzt verwendete Einstellungen* auf. Dazu gehören, neben der Datenidentifikation, auch die im Dialog vorgenommenen Einstellungen. Dieses nützliche Feature erlaubt ein schnelles Zugreifen auf zuletzt verwendete Tools und deren Einstellungen.

Gespeicherte und zuletzt verwendete Einstellungen

Im unteren Abschnitt des GTM-Fensters sind gespeicherte und die zuletzt verwendeten Einstellungen[1] aufgelistet. Die Tabelle *Zuletzt verwendete Einstellungen* hält dabei automatisch die letzten 20 Einstellungen vor. Um eine Einstellung länger vorzuhalten und in der Tabelle *Gespeicherte Einstellungen* abzulegen, kann der weiter unten erklärte *Speichern unter* -Button des GTM oder eines Tools verwendet werden. Die gespeicherten Einstellungen können dazu verwendet werden, häufig verwendete Einstellungen wieder abzurufen, ohne dass man alle Parameter (wie Tool, Datenidentifikation, Parameter des Tools) erneut auswählen muss.

Um eine der folgenden Funktionen für gespeicherte und zuletzt verwendete Einstellungen zu verwenden, müssen eine oder mehrere Einstellungen selektiert werden.

- *Starten*

  Diese Funktion startet das mit der Einstellung verknüpfte Tool mit den hinterlegten Parametern und Datenidentifikationen.

- *Speichern unter...*

  Die gerade selektierte Datenidentifikation wird permanent in der Tabelle *Gespeicherte Einstellungen* abgespeichert. Dazu kann der Benutzer einen Namen angeben, unter dem sie abgelegt werden soll.

- *Ändern...*

  Ein Klick auf diesen Button öffnet das mit der Einstellung verknüpfte Tool für die hinterlegten Datenidentifikationen und Parameter. Diese können dann vom Benutzer angepasst werden, bevor das Tool gestartet wird. Sollen die geänderten Parameter gespeichert werden, ist dies durch Anklicken des *Speichern unter...*-Buttons im Fenster des Tools möglich.

- *Löschen*

  Diese Funktion löscht gespeicherte oder zuletzt verwendete Einstellungen. Im Gegensatz zu den anderen Funktionen können mehrere Einstellungen gleichzeitig selektiert und gelöscht werden.

Die letzten beiden Funktionen können nur mit den gespeicherten Einstellungen verwendet werden:

- *Selektion exportieren*

  Die selektierte Einstellung aus der Tabelle *Gespeicherte Einstellungen* kann mit Hilfe dieser Funktion im XML-Format gespeichert werden. Dazu muss der Benutzer das Verzeichnis auswählen, in dem die Einstellung als XML Datei abgelegt werden soll. Als Dateiname wird der Name der Einstellung mit der Dateiendung `.xml` verwendet.

  > **Wichtig**
  >
  > Eine bereits vorhandene Datei mit gleichem Namen wird ohne Rückfrage ersetzt.

- *Einstellungen importieren*

[1] Eine *Einstellung* beinhaltet die Datenidentifikation sowie das gewählte Tool und dessen Parameter. Gespeicherte Einstellungen erhalten zusätzlich einen vom Benutzer festgelegten Titel (siehe untere Fensterhälfte in Abbildung 3-2).

Mit der Funktion *Einstellungen importieren* können im XML Format abgespeicherte Einstellungen in die Tabelle *Gespeicherte Einstellungen* importiert werden.

3.2.1.1 Onlineprotokollierer

Mit dem *Onlineprotokollierer* können Datensätze einer Datenidentifikation empfangen werden (Kurzbeschreibung siehe Abschnitt 1.5.9).

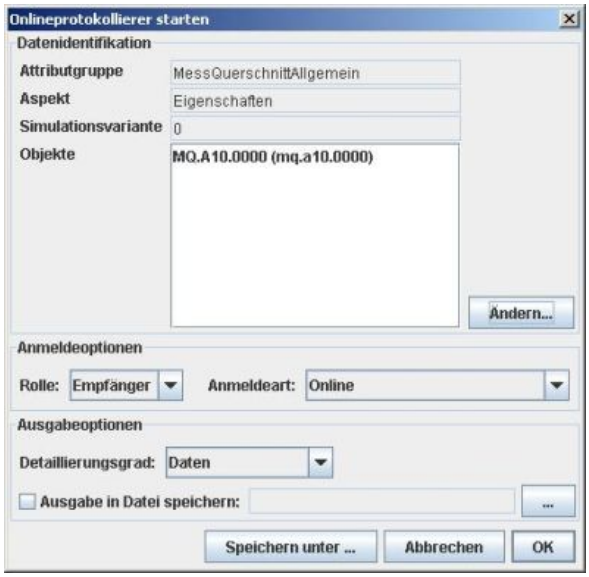


Abbildung 3-3: Onlineprotokollierer einstellen

Zum Erstellen eines Empfangsprotokolls muss der Onlineprotokollierer wie folgt eingestellt werden:

Datenidentifikation

Die Datenidentifikation, die bereits im Hauptfenster des GTM festgelegt wurde, wird angezeigt. Soll eine andere Datenidentifikation verwendet werden, so kann diese nach Anklicken des *Ändern...* Buttons eingestellt werden.

Anmeldeoptionen

Im Bereich *Anmeldeoptionen* können zwei Einstellungen vorgenommen werden:

- *Rolle*

  Hier wird festgelegt, welche Rolle der Onlineprotokollierer beim Empfangen der Daten einnimmt (genaueres zu den Bedeutungen der einzelnen Rollen siehe Abschnitt 2.2).

  - Empfänger
  - Senke

  Sowohl Empfänger als auch Senke können Daten empfangen. Eine Senke legt zusätzlich den Datenverteiler für Verwaltungsaufgaben fest. Die Rolle *Empfänger* sollte demnach gewählt werden, wenn bereits eine Senderapplikation als *Quelle* sendet und damit der Datenverteiler für die Verwaltungsaufgaben festgelegt ist. *Senke* sollte gewählt werden, falls die Senderapplikation den Datenverteiler für Verwaltungsaufgaben nicht festlegt, also in der Rolle *Sen-*

der angemeldet ist.

- *Anmeldeart*

  Durch die *Anmeldeart* wird festgelegt, welche Datenarten (siehe auch Abschnitt 2.2.2.6) empfangen werden sollen. Es stehen drei Möglichkeiten zur Auswahl:

  1. *Online*

     In diesem Fall werden nur als *online* markierte Daten protokolliert.

  2. *Nur geänderte Datensätze*

     Nur Datensätze, die sich vom davor empfangenen Datensatz unterscheiden, werden protokolliert. Sofern also der zuletzt und der neu empfangene Datensatz übereinstimmen, wird kein neuer Datensatz protokolliert.

  3. *Auch nachgelieferte Datensätze*

     Bedeutet, dass auch als *nachgeliefert* gekennzeichnete Datensätze mitprotokolliert werden.

Ausgabeoptionen

- *Detaillierungsgrad*

  Der Detaillierungsgrad der Ausgaben des Protokollierers kann festgelegt werden. Es stehen folgende Einstellungsmöglichkeiten zur Verfügung:

  1. *Keine Ausgabe*

     Es wird keine Ausgabe erzeugt.

  2. *Aktualisierung*

     Gibt jedes Mal die Anzahl der erhaltenen Datensätze aus, sobald ein Datensatz empfangen wurde.

  3. *Kopfinformationen*

     Der Header des anliegenden Datensatzes wird zusätzlich zu den Ausgaben des Levels *Aktualisierung* ausgegeben.

  4. *Daten*

     Zusätzlich zu den *Kopfinformationen* werden auch die Nutzdaten des anliegenden Datensatzes dargestellt. Dies ist die Standardeinstellung.

  5. *XML*

     Zum späteren Einspielen der empfangenen Daten in den Datenverteiler durch unterstützende Skripte kann die Ausgabe auch im XML-Format erfolgen (Abschnitt 3.2.2).

- *Ausgabe in Datei speichern*

  Hier kann eine Datei gewählt werden, in die die oben spezifizierten Details zu den empfangenen Daten geschrieben werden, sofern eine Ausgabe auf der Konsole nicht erwünscht ist.

3.2.1.2 Archivanfrage (Stream)

Bei laufendem Archivsystem (siehe [TANFARS]) können mit Hilfe des Tools *Archivanfrage (Stream)* Daten aus dem Archivsystem abgerufen werden (Kurzbeschreibung siehe

Abschnitt 1.5.6). Die Datensätze werden in einer Tabelle angezeigt. Bestimmte Datensätze, z.B. „keine Quelle“ und „keine Senke“, werden farbig hervorgehoben.

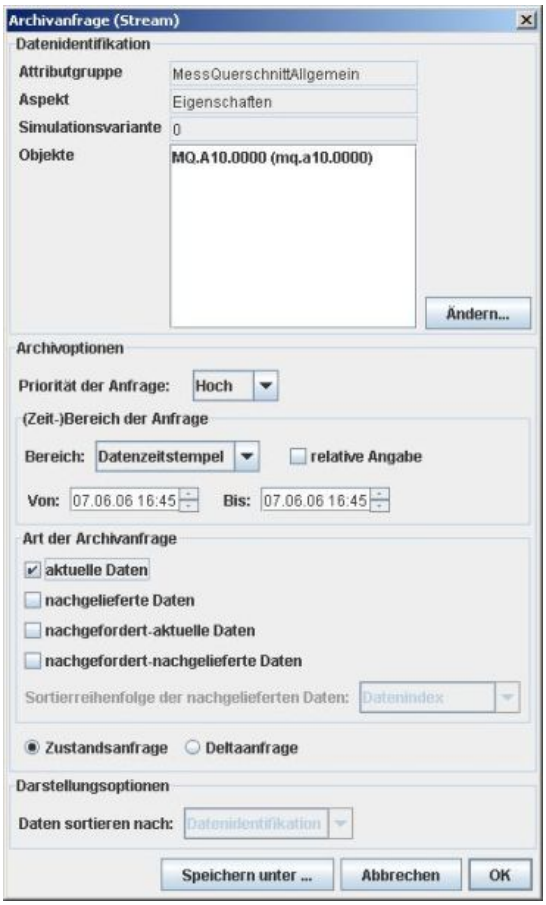


Abbildung 3-4: Einstellmöglichkeiten der Archivanfrage

Folgende Einstellungen sind dabei vorzunehmen:

Datenidentifikation

Hier kann die Datenidentifikation eingestellt werden (siehe Abschnitt 3.2.1.1).

Archivoptionen

1. *Priorität*

   Mit der Auswahlliste kann die Priorität der Archivanfrage (siehe [TANFARS]) festgelegt werden.

   Anfragen werden in eine der drei Warteschlangen *hoch*, *mittel* oder *niedrig* eingereiht, welche alle mit der gleichen Priorität abgearbeitet werden. Sinn macht die Einordnung nur, wenn von der Applikation tatsächlich ausschließlich entsprechend priorisierte Anfragen in die korrespondierenden Warteschlangen geschrieben werden. Die Warteschlange *hoch* wird also nur dann schneller abgearbeitet als die Warteschlange *mittel*, wenn sie mit weniger Anfragen gefüllt ist.

2. *Bereich der Anfrage*

   Der gewünschte Anfragebereich kann anhand des

   - Datenzeitstempels,

- Archivzeitstempels oder
- Datenindex

festgelegt werden. Der Bereich wird als Intervall oder relative Angabe spezifiziert.

Zur Intervallauswahl der Archivantwort müssen Anfangs- und Endwert des Intervalls gesetzt werden:

Abbildung 3-5: Absolute Archivanfrage

Wird das Häkchen *relative Angabe* gesetzt, muss ein Endwert und die Anzahl der Ergebnisdatensätze vor diesem spezifiziert werden:

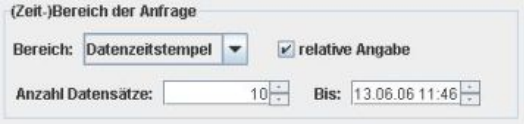


Abbildung 3-6: Relative Archivanfrage

3. *Art der Anfrage*

In *Art der Anfrage* kann eine beliebige Kombination von Datenarten für die Anfrage spezifiziert werden. Folgende Datenarten sind möglich (siehe Abschnitt 2.2.2.6):

- *Aktuelle Daten*

  Aktuelle Daten sind der Normalfall und entsprechen den als *online* gekennzeichneten Datensätzen.

- *Nachgelieferte Daten*

  Bricht zeitweise die Verbindung vom Datenendgerät zum Datenverteiler ab, können die Datensätze erst nach Wiederherstellen der Datenverteilerverbindung gesendet werden und werden deshalb als *nachgeliefert* gekennzeichnet.

- *Nachgefordert-aktuelle Daten*

  Erkennt das Archivsystem Lücken in seinen Datenbeständen, kann es Daten bei anderen Archivsystemen nachfordern um diese Lücken zu schließen. Tritt eine Lücke im Bestand der online Daten auf und können diese Datensätze von einem anderen Archivsystem auf Anforderung geliefert werden, werden sie als *nachgefordert-aktuelle* Daten gekennzeichnet und archiviert.

- *Nachgefordert-nachgelieferte Daten*

  Tritt eine Datenlücke im Bestand der *nachgelieferten* Daten auf, so wird diese mit Datensätzen der Kennzeichnung *nachgefordert-nachgeliefert* geschlossen.

Bei *nachgelieferten* Daten muss zusätzlich die Sortierung der Ergebnisdatensätze angegeben werden. Sie können nach

- Datenindex oder
- Datenzeitstempel

sortiert werden.

4. *Zustands- oder Deltaanfrage*

   Des Weiteren muss bei der Anfrage spezifiziert werden, ob es sich um eine Deltaanfrage oder eine Zustandsanfrage handelt.

   - *Zustandsanfrage*

     Bei einer Zustandsanfrage werden alle Datensätze des angegebenen Bereichs angezeigt.

   - *Deltaafrage*

     Bei der Deltaanfrage werden nur diejenigen Datensätze zurückgegeben, bei denen sich die Nutzdaten von denen des Vorgängerdatensatzes unterscheiden.

   Wichtig

   Die Deltaanfrage berücksichtigt keine als *nachgeliefert* gekennzeichneten Datensätze, d.h. Datensätze mit der Kennzeichnung *nachgeliefert* oder *nachgefordert-nachgeliefert* werden nicht ausgegeben.

Darstellungsoptionen (in der aktuellen Version nicht verwendbar)

Die streambasierte Archivanfrage unterstützt die folgenden Darstellungsoptionen:

- *Datenidentifikation*

  Die Ergebnisdatensätze werden nach Datenidentifikation sortiert angezeigt.

- *Zeitstempel*

  Die Ergebnisdatensätze werden nach Datenzeitstempel sortiert angezeigt.

3.2.1.3 Systemprotokollierer

Der *Systemprotokollierer* dient ebenfalls dazu, Anfragen an das Archivsystem zu stellen. Im Gegensatz zur streambasierten Archivanfrage (siehe voriges Kapitel) bietet der Systemprotokollierer verschiedene Möglichkeiten zur Formatierung der Ausgabe. So können die vom Archivsystem gelieferten Datensätze etwa im XML-Format ausgegeben oder in einer Datei gespeichert werden. Die streambasierte Archivanfrage erlaubt die Darstellung der Ergebnisse nur in tabellarischer Form analog zum Onlineprotokollierer (siehe Abschnitt 3.2.1.1).

Für das Tool gelten die gleichen Voraussetzungen und Parameter wie für *Archivanfrage (alt)*, das jedoch von Archivsystemen nach [TANFARS] nicht mehr unterstützt wird. Die entsprechenden Funktionen sind daher mit dem Systemprotokollierer verfügbar.

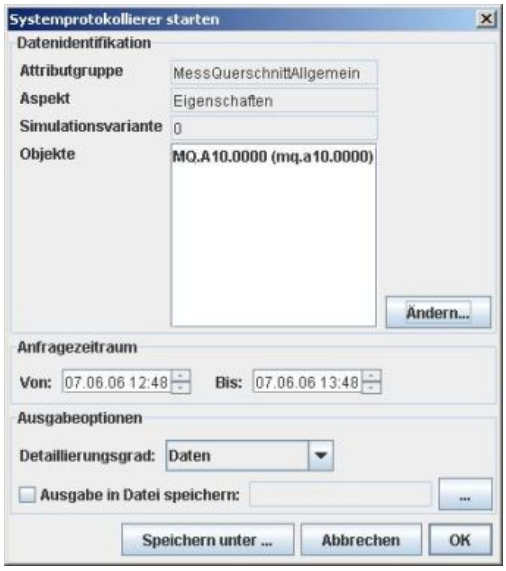


Abbildung 3-7: Einstellmöglichkeiten des Systemprotokollierers

Datenidentifikation

Hier wird die Datenidentifikation festgelegt (siehe Abschnitt 3.2.1.1).

Anfragezeitraum

Der Anfragezeitraum für die Archivanfrage muss einen Anfangs- und einen Endwert enthalten.

Ausgabeoptionen

Hier werden die Parameter für die Ausgabe der protokollierten Daten eingestellt. Die Ausgabeoptionen entsprechen denen des Onlineprotokollierers (siehe Abschnitt 3.2.1.1).

3.2.1.4 Datengenerator

Der Datengenerator kann zu einer oder mehreren Datenidentifikationen zufällige Daten erzeugen. Dies ist zum Beispiel für Tests äußerst nützlich. Die erzeugten Datensätze können über den Datenverteiler versendet und zur späteren Verwendung in einer Datei abgelegt werden, etwa um einen Test mit immer gleichen Daten durchzuführen.

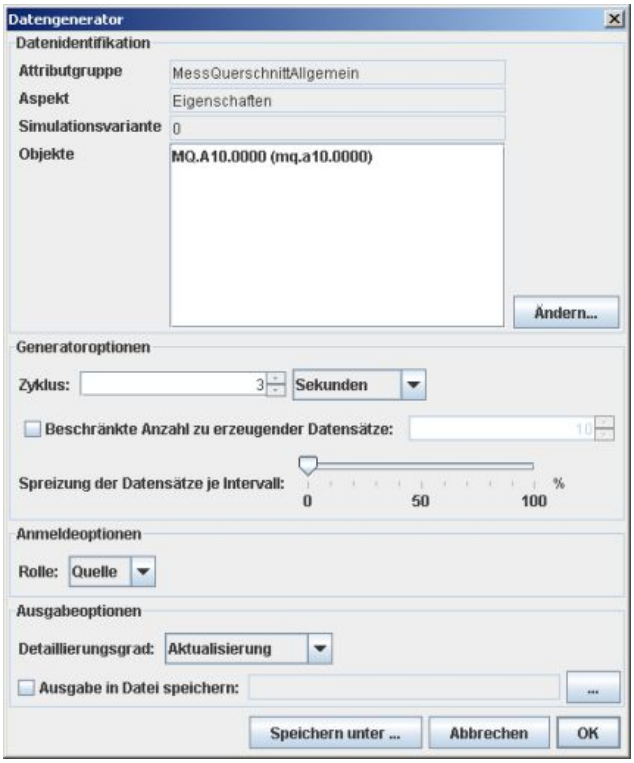


Abbildung 3-8: Einstellmöglichkeiten des Datengenerators

Um den Datengenerator starten zu können muss er entsprechend eingestellt werden:

Datenidentifikation

Hier kann die Datenidentifikation eingestellt werden (siehe Abschnitt 3.2.1.1).

Generatoroptionen

Folgende Optionen werden vom Datengenerator unterstützt:

- *Zyklus*

  Gibt an in welchen Zeitabständen Daten erzeugt werden sollen. Es sind Werte ab einer Millisekunde möglich.

- *Anzahl beschränken*

  Hier kann die Anzahl der insgesamt zu erzeugenden Datensätze festgelegt werden. Ist das Häkchen nicht gesetzt, werden solange Datensätze erzeugt, bis das Generatorfenster geschlossen wird.

- *Spreizung*

  Die Spreizung der Datensätze ist ein nützliches Instrument bei längeren Zyklen und mehreren Objekten. Ist sie auf 0% gestellt, werden gleichzeitig für alle Objekte zum Startzeitpunkt und dann erst genau zum Ablauf eines Zyklus wieder Datensätze erzeugt. Bei einer maximalen Spreizung von 100% werden die Datensätze über die gesamte Zyklusdauer **gleichmäßig verteilt** erzeugt.

Beispiel: Werden für die Messquerschnitte `mq.a10.0000`, `mq.a10.0001` und `mq.a10.0002` Datensätze mit einer Spreizung von 50% und einer Zyklusdauer von 1 Minute erzeugt, so wird

der erste Datensatz für `mq.a10.0000` zum Zeitpunkt 0, der erste Datensatz für `mq.a10.0001` nach Ablauf von 15 Sekunden und der erste Datensatz für `mq.a10.0002` nach 30 Sekunden erzeugt. Nach 30 Sekunden (50% der Zyklusdauer) wurde also für jedes Objekt ein Datensatz erzeugt. Nach Ablauf von einer Minute wiederholt sich der Vorgang.

Anmeldeoptionen

Im Bereich Anmeldeoptionen kann die Rolle des Senders festgelegt werden (genaueres zu den Bedeutungen der einzelnen Rollen siehe Abschnitt 2.2).

- Quelle
- Sender

Sowohl Quelle als auch Sender können Daten versenden. Eine Quelle legt aber zusätzlich den Datenverteiler für Verwaltungsaufgaben fest. Für eine Datenidentifikation kann eine Quelle angemeldet werden, wenn es weder eine Senke noch eine andere Quelle gibt.

Ausgabeoptionen

In diesem Bereich werden die Parameter für die Ausgabe der erzeugten Daten eingestellt (genaueres siehe Abschnitt 3.2.1.1).

3.2.2 XML aufzeichnen und per Datengenerator senden

Mit Hilfe der oben vorgestellten Kernsoftwaretools können Daten im XML-Format aufgezeichnet und zu einem späteren Zeitpunkt wieder eingespielt werden. Diese Funktion ist hilfreich, wenn ein Test mit immer gleichen Daten wiederholt werden soll: Die Daten werden einmalig aufgezeichnet und können dann beliebig oft eingespielt werden. Dabei werden die ursprünglichen Datenzeitstempel wiederverwendet. Ebenso wird das Timing beim Einspielen der Datensätze beibehalten. Das bedeutet, dass falls die Ausgangsdaten Minutenwerte eines Datenendgeräts waren, diese auch wieder im Minuten-Rhythmus eingespielt werden.

Folgende Schritte sind also nötig:

- Daten einmalig mit dem Datengenerator erzeugen oder Livedaten mitschneiden.
- Gewonnene Daten als XML-Datei abspeichern.
- Erzeugte XML-Datei anpassen.
- Skript zum Einspielen der abgespeicherten Daten über die Kommandozeile vorbereiten.
- Daten über die Kommandozeile beliebig oft einspielen.

Im Detail muss wie folgt vorgegangen werden:

1. *Initialisierung*

   Zunächst wird der GTM gestartet und die gewünschte(n) Datenidentifikation(en) für die zu erzeugenden oder aufzuzeichnenden Daten im Hauptfenster gewählt.

2. *Daten erzeugen und aufzeichnen*

   Abhängig davon, ob man zufällige Daten oder einen Livemitschnitt erzeugen möchte, müssen verschiedene Tools des GTM verwendet werden:

   - *Zufallsdaten erzeugen: Datengenerator*

Mit Hilfe des Datengenerators können Zufallsdaten erzeugt werden. Im Bereich Ausgabeoptionen des Datengenerators müssen folgende Werte eingestellt werden, damit die Daten im XML-Format in einer Datei abgelegt werden können:

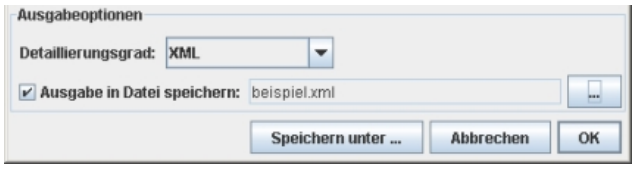


Abbildung 3-9: Einstellungen des Datengenerators für XML-Ausgabe

- *Mitprotokollieren von Daten: Onlineprotokollierer*

  Im Bereich Ausgabeoptionen des Onlineprotokollierers muss analog zum Datengenerator (siehe Abbildung 3-9) die Ausgabeoption *XML* und eine Ausgabedatei gewählt werden.

Nach dem Bestätigen mit *OK* werden Datensätze zur gewählten Datenidentifikation auf den Datenverteiler gelegt bzw. ein Mitschnitt von Datensätzen derselben erstellt und in die angegebene XML-Datei geschrieben.

3. *Datenerzeugung und Empfang abbrechen*

   Durch Schließen des Ausgabefensters des Datengenerators kann das Erzeugen der Datensätze beendet werden. Das Beenden des Livemitschnitts erfolgt analog durch Schließen des Ausgabefensters des Onlineprotokollierers.

4. *XML-Datei anpassen*

   Die in den vorherigen Schritten erzeugte XML-Datei muss nun so aufbereitet werden, dass sie mit dem Datengenerator per Kommandozeile wieder eingespielt werden kann.Ein Einspielen aus dem GTM heraus ist nicht möglich. Die Datei ist dafür mit einem Text- oder XML-Editor zu öffnen und zu editieren.

   Folgende Änderungen sind vorzunehmen:

   - Im Tag `<aufrufparameter>` innerhalb des `<kopf>`-Tags muss das Attribut `wert` angepasst werden, indem alle Parameterwerte bis auf `-objekte=...` und `-daten=...` entfernt werden.
   - Unter Umständen wird beim Beenden des Onlineprotokollierers die XML-Datei nicht korrekt abgeschlossen. Es ist möglich, dass die Schluß-Tags deshalb in der XML-Datei fehlen. In so einem Fall sind am Ende der XML-Datei die Tags `</koerper>` und `</protokoll>` einzufügen, damit die XML-Datei wohlgeformt ist und beim späteren Einspielen keine Fehler auftreten.

5. *Skript zum Einspielen erstellen*

   Eine ausführbare Datei für Windows-Betriebssysteme zum Einspielen der eben erzeugten Daten könnte wie folgt aussehen:

```
@ECHO off

REM :: Java Einstellungen ::

REM Classpath für die Java Virtual Machine unter dem nach dem
REM übersetzten Java-Code des Datengenerators gesucht wird:
SET cp=-cp ..\bibliothek\runtime.jar

REM Argumente für die JVM:
SET jvmArgs=-Dfile.encoding=ISO-8859-1 -Xms32m
```

```
REM Pfad zum java-Binary setzen:
IF "%JAVA_HOME%"=="" (SET java=java) ELSE (SET java=%JAVA_HOME%\bin\java)


REM :: Einstellungen für den Datenverteiler ::

REM Benutzereinstellungen:
SET passwortDatei=.\passwd
SET benutzer=Tester
SET authentifizierung=-benutzer=%benutzer% -authentifizierung=%passwortDatei%

REM Adresseinstellungen:
SET davHost=hauptsystem
SET davPort=8083
SET davAdresse=-datenverteiler=%davHost%:%davPort%

REM Debugeinstellungen:
SET debugDefaults=-debugFilePath=..


REM :: Einspielen der XML-Datei durchführen ::

SET javaParams=%java% %cp% %jvmArgs%
SET davParams= %davAdresse% %authentifizierung% %debugDefaults%
SET datGenClass=pat.gen.DataGenerator
SET eingabeParams=-eingabe=beispiel.xml

START /b "XML-Datei einspielen" %java% %datGenClass% %davParams% %eingabeParams%

PAUSE
```

In der vorletzten Zeile `START ...` wird der Datengenerator auf der Kommandozeile gestartet. Ein Start aus dem GTM heraus ist nicht möglich, wenn die Datensätze aus der XML-Datei eingespielt werden sollen. Die benötigten Parameter werden in den Zeilen darüber gesetzt und sollten angepasst werden:

java
Enthält Angaben für die Java Virtual Machine (Classpath, Heapspace-Angaben, usw.).

datGenClass
Enthält den Klassennamen des Datengenerators. Üblicherweise muss dieser Parameter nicht angepasst werden.

davParams
Enthält Angaben zum Benutzer (Name und Passwort) sowie zum Datenverteiler (IP-Adresse des Hosts und Port). Üblicherweise muss dieser Parameter nicht angepasst werden.

eingabeParams
Enthält den vollständigen Dateinamen der XML-Datei mit den Datensätzen, die eingespielt werden soll.

6. *Daten einspielen und kontrollieren*

Sobald das soeben erstellte Skript ausgeführt wird, werden die in der XML-Datei enthaltenen Daten eingespielt. Dabei werden die ursprünglichen Datenzeitstempel und das ursprüngliche Timing beibehalten. Der Datenindex der Datensätze ändert sich bei jedem Einspielen, da dieser ja ausschließlich vom Datenverteiler festgelegt wird.

Zur Kontrolle kann der Onlineprotokollierer gestartet werden, der auch auf die oben verwendete(n) Datenidentifikation(en) eingestellt werden muss. Falls alle Schritte korrekt durchgeführt wurden, gibt der Onlineprotokollierer die eingespielten Datensätzen wieder aus.

3.2.3 Datenmodellbrowser

Der Datenmodellbrowser zeigt das Datenmodell in einer Baumstruktur an. Im Gegensatz zum HTML-Datenkatalog zeigt der Datenmodellbrowser auch alle vorhandenen Objekte des Datenmodells. Die fett formatierten Knoten des Baumes sind Methodennamen aus der Kernsoftware API (siehe [APIBIB] und [APISYS]) und werden dort beschrieben.

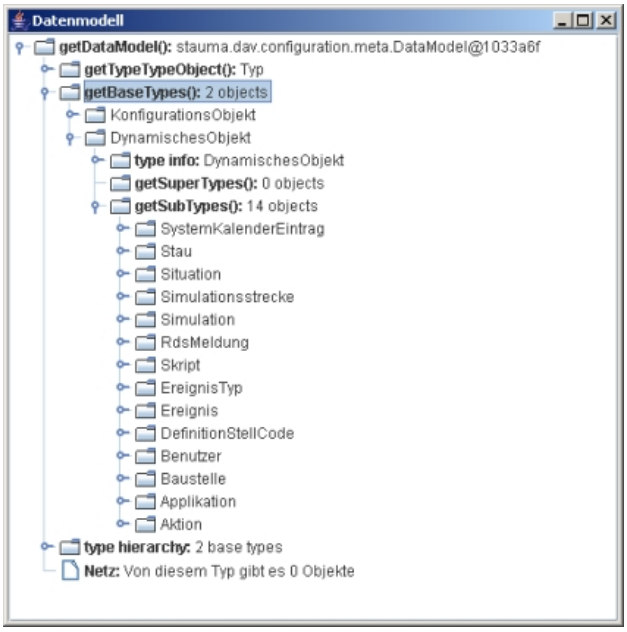


Abbildung 3-10: Datenmodellbrowser

Der Datenmodellbrowser ist in zwei Varianten verfügbar:

- *Offline-Variante*

  Die Offline-Variante des Datenmodellbrowsers liest beim Start das Datenmodell aus der Datei `config.xml` ein und stellt den Inhalt in einem Fenster in einer Baumstruktur dar. Zum Starten ist das Windows-Skript `KonfigBrowserOffline.bat` im Ordner `skripte-dosshell`, bzw. das Linux-Skript `KonfigBrowserOffline.bash` im Ordner `skripte-bash` im Installationsverzeichnis der Kernsoftware auszuführen.

- *Online-Variante*

  Die Online-Variante des Modellbrowsers bezieht Informationen zum Datenmodell von der SWE Konfiguration, d.h. vor dem Verwenden dieser Variante muss die Kernsoftware gestartet werden. Das Windows-Skript `KonfigBrowserOnline.bat` im Ordner `skripte-dosshell`, bzw. das Linux-Skript `KonfigBrownserOnline.bash` im Ordner `skripte-bash` im Installationsverzeichnis der Kernsoftware startet die Online-Variante.

3.3 Datenmodell

Dieser Abschnitt befasst sich mit der Anpassung des Datenmodells in der Praxis. Anhand eines Beispiels wird erklärt, wie ein Objekt angelegt, Mengen zugeordnet und konfigurierende Daten eingetragen werden können. Anschließend wird auf die Besonderheiten beim Erzeugen eines Konfigurationsverantwortlichen eingegangen. Alle Änderungen werden grundsätzlich am XML-Datenkatalog vorgenommen.

> **Warnung**
>
> Änderungen am Datenmodell haben Auswirkungen auf das Systemverhalten. Die Änderungen, die alle am XML-Datenkatalog vorgenommen werden, werden erst nach Konvertieren des Datenkatalogs zur `config.xml` und Neustart der Kernsoftware vom System verwendet. Abschnitt 2.1.3 zeigt, wie beim Konvertieren vorgegangen werden muss.

3.3.1 Allgemeine Vorgehensweise

Sollen Änderungen am Datenkatalog vorgenommen werden, so hat sich folgende Vorgehensweise als nützlich erwiesen:

1. Entwurf des Elements „auf dem Papier“.
2. Suchen nach einem ähnlichen Element im Datenkatalog.
3. Zieldatei identifizieren: Der XML-Datenkatalog besteht aus mehreren Versorgungsdateien, die im Verzeichnis `datenkatalog` zu finden sind. Werden Änderungen an den bereits existierenden Dateien vorgenommen, so sollte unbedingt darauf geachtet werden, dass dies in Harmonie mit der Gliederung des Datenkatalogs (siehe Abschnitt 2.1.5.1) geschieht. Alternativ kann ein neuer Konfigurationsbereich (in einer neuen XML-Datei) erzeugt werden.
4. Kopieren des bereits im Datenkatalog existierenden Elements an die neue Position. Zum Bearbeiten der XML-Dateien kann ein Texteditor verwendet werden; besser geeignet ist jedoch ein XML-Editor.
5. Durchführen der notwendigen Änderungen.
6. Herunterfahren der Kernsoftware.
7. Konvertieren des Datenkatalogs. Die Schritte 5 und 7 müssen so oft wiederholt werden, bis das Konvertieren des Datenkatalogs fehlerfrei durchgeführt werden kann.
8. Neustart der Kernsoftware.

> **Tipp**
>
> Die Fehlermeldungen, die beim Konvertieren des Datenkatalogs ausgegeben werden, sind oft nicht sehr hilfreich. Daher sollten Änderungen am Datenkatalog in möglichst kleinen Schritten durchgeführt und der Datenkatalog nach jedem Schritt konvertiert werden. So kann die Fehlerursache besser eingegrenzt werden.

3.3.2 Objekte anlegen

3.3.2.1 Allgemeine Vorgehensweise

Objekte werden im Element `objekte` des jeweiligen Konfigurationsbereiches angelegt. Dazu wird das Element `konfigurationsObjekt` dort eingefügt. Falls konfigurierende Daten für eine Attributgruppe dieses Objekts festgelegt werden sollen, werden diese dort ebenfalls eingetragen.

Es folgt eine Beschreibung des `konfigurationsObjekt` Elements, durch das sämtliche Eigenschaften eines Objekts festgelegt werden können:

Attribute		
`typ`	PID des Objekttyps, der dem Objekt zu Grunde liegen soll.	
`pid`	PID des Objekts festlegen (optional).	
`name`	Namen des Objekts festlegen (optional).	
Kind-Element		
`datensatz`	Das Element kann pro konfigurierender Attributgruppe einmal verwendet werden. In dieses Element können die konfigurierenden Daten der jeweiligen Attributgruppe des Objekts eingetragen werden.	
	Attribute	
	`pid`	PID der konfigurierenden Attributgruppe. Es können nur diejenigen konfigurierenden Attributgruppen verwendet werden, die dem Objekttyp des Objekts zugeordnet wurden.
	Kind-Elemente	
	`datum`	In dieses Element werden die konfigurierenden Daten für atomare Attribute eingetragen
	`datenliste`	In dieses Element werden die konfigurierenden Daten für nicht-atomare Attribute eingetragen.
Kind-Element		
`objektMenge`	Dieses Element kann für jede Menge, die dem zugrunde liegenden Objekttyp zugeordnet ist, einmal verwendet werden. Wenn eine Menge hier nicht eingetragen wird, dann kann diese Menge zur Laufzeit nicht verwendet werden. Der zugrunde liegende Objekttyp legt fest, ob eine Menge eingetragen werden muss (siehe Abschnitt 2.1.2.2).	
	Attribute	
	`name`	Name der Menge, die angelegt werden soll.
	Kind-Elemente	
	-	-

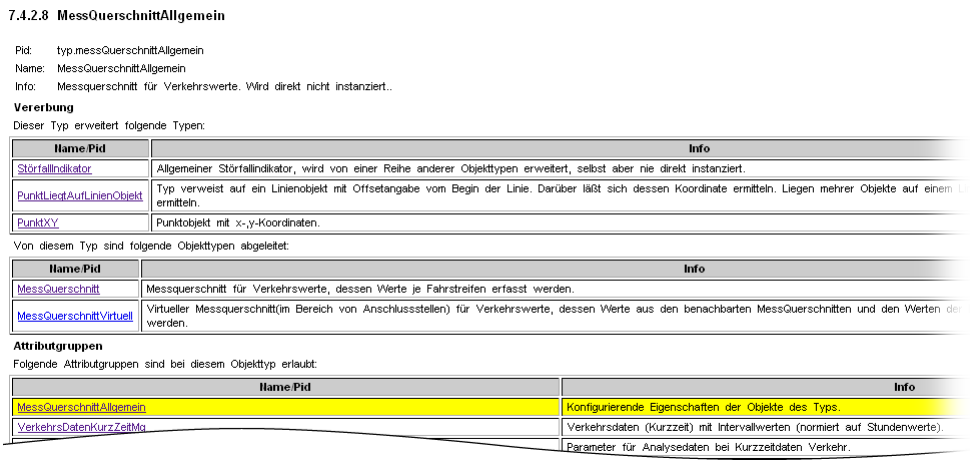
Tabelle 3-1: Element konfigurationsObjekt

> **Tipp**
>
> Das Kind-Element `info` kann verwendet werden, um Beschreibungen zu anderen Elementen, z.B. `konfigurationsObjekt`, `datum` oder `datumListe`, zu hinterlegen. Aus Gründen der Einfachheit wurde es in der obigen Beschreibung nicht erwähnt.

3.3.2.2 Beispiel

Es soll ein Messquerschnitt mit konfigurierenden Daten in das Datenmodell eingetragen werden. Im XML-Datenkatalog wird dazu ein Konfigurationsbereich mit Testobjekten, z.B. `kb.objekteTestUnterzentraleK2S_10_MessQuerschnitte` (Datei `kb.objekteTestUnterzentraleK2S_10_MessQuerschnitte.xml`) um die neue Objektdefinition erweitert. Das Objekt soll auf dem Objekttyp `typ.messQuerschnitt` basieren. Die Eigenschaften dieses Objekttyps, die beim Definieren des Objekts berücksichtigt werden müssen, können dem HTML-Datenkatalog entnommen werden:

- Der Messquerschnitt besitzt diverse konfigurierende Attributgruppen, z.B. `MessQuerschnittAllgemein` (durch den Objekttyp `MessQuerschnittAllgemein`), für die konfigurierende Daten hinterlegt werden können. Das Hinterlegen dieser Daten ist optional.



7.4.2.8 MessQuerschnittAllgemein

Pid: typ.messQuerschnittAllgemein
Name: MessQuerschnittAllgemein
Info: Messquerschnitt für Verkehrswerte. Wird direkt nicht instanziert..

Vererbung

Dieser Typ erweitert folgende Typen:

Name/Pid	Info
Störfallindikator	Allgemeiner Störfallindikator, wird von einer Reihe anderer Objekttypen erweitert, selbst aber nie direkt instanziert.
PunktLiegtAufLinienObjekt	Typ verweist auf ein Linienobjekt mit Offsetangabe vom Begin der Linie. Darüber läßt sich dessen Koordinate ermitteln. Liegen mehrer Objekte auf einem ermitteln.
PunktXY	Punktobjekt mit x-,y-Koordinaten.

Von diesem Typ sind folgende Objekttypen abgeleitet:

Name/Pid	Info
MessQuerschnitt	Messquerschnitt für Verkehrswerte, dessen Werte je Fahrstreifen erfasst werden.
MessQuerschnittVirtuell	Virtueller Messquerschnitt(im Bereich von Anschlussstellen) für Verkehrswerte, dessen Werte aus den benachbarten MessQuerschnitten und den Werten der werden.

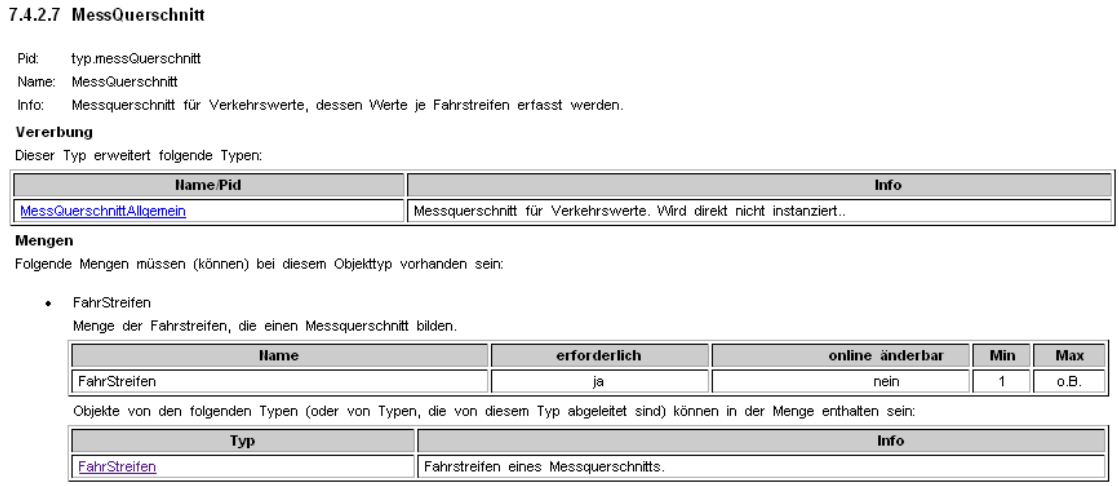
Attributgruppen

Folgende Attributgruppen sind bei diesem Objekttyp erlaubt:

Name/Pid	Info
MessQuerschnittAllgemein	Konfigurierende Eigenschaften der Objekte des Typs.
VerkehrsDatenKurzZeitMq	Verkehrsdaten (Kurzzeit) mit Intervallwerten (normiert auf Stundenwerte).
	Parameter für Analysedaten bei Kurzzeitdaten Verkehr.

Abbildung 3-11: Ein Messquerschnitt erbt die konfigurierende Attributgruppe MessQuerschnittAllgemein

- Die *erforderliche Menge* `Fahrstreifen` muss für das Objekt angelegt werden. Ansonsten kann es später zu einem Fehler beim Konvertieren des Datenkatalogs kommen — das aktuelle Kernsoftware Release vom 20.03.2006 konvertiert den Datenkatalog jedoch auch, wenn die Menge nicht angelegt wird.



7.4.2.7 MessQuerschnitt

Pid: typ.messQuerschnitt
Name: MessQuerschnitt
Info: Messquerschnitt für Verkehrswerte, dessen Werte je Fahrstreifen erfasst werden.

Vererbung

Dieser Typ erweitert folgende Typen:

Name/Pid	Info
MessQuerschnittAllgemein	Messquerschnitt für Verkehrswerte. Wird direkt nicht instanziert..

Mengen

Folgende Mengen müssen (können) bei diesem Objekttyp vorhanden sein:

- FahrStreifen
  Menge der Fahrstreifen, die einen Messquerschnitt bilden.

Name	erforderlich	online änderbar	Min	Max
FahrStreifen	ja	nein	1	o.B.

Objekte von den folgenden Typen (oder von Typen, die von diesem Typ abgeleitet sind) können in der Menge enthalten sein:

Typ	Info
FahrStreifen	Fahrstreifen eines Messquerschnitts.

Abbildung 3-12: Die erforderliche Menge Fahrstreifen

Das folgende Listing zeigt den Konfigurationsbereich, nachdem das neue Objekt (**Zeilen 7 - 17**) eingefügt wurde:

```
1: <?xml version="1.0" encoding="ISO-8859-1"?>
2: <!DOCTYPE konfigurationsBereich PUBLIC "-//K2S//DTD Dokument//DE" "K2S.dtd">
3: <konfigurationsBereich pid="kb.objekteTestUnterzentraleK2S_10_MessQuerschnitte" ...
4:   ...
5:   <objekte>
6:     ...
7:     <konfigurationsObjekt typ="typ.messQuerschnitt" pid="mq.tutorial.test"
8:         name="MQ.Tutorial.Test">
9:       <datensatz pid="atg.messQuerschnittAllgemein">
```

```
10:         <datum name="Typ" wert="HauptFahrbahn"/>
11:         <datum name="ErsatzMessQuerschnitt" wert="mq.a10.0000"/>
12:       </datensatz>
13:       <objektMenge name="FahrStreifen">
14:         <element pid="fs.mq.a10.0009.hfs"/>
15:         <element pid="fs.mq.a10.0009.2üfs"/>
16:       </objektMenge>
17:     </konfigurationsObjekt>
18:   </objekte>
19: </konfigurationsBereich>
```

In den **Zeilen 7 - 8** wird angegeben, dass ein Objekt vom Objekttyp `typ.messQuerschnitt` instanziert wird. Außerdem wird die PID und der Name des Objekts festgelegt. Die konfigurierenden Daten für die Attributgruppe `atg.messQuerschnittAllgemein` werden in den **Zeilen 9 - 12** beispielhaft eingetragen:

- In **Zeile 9** wird die PID der konfigurierenden Attributgruppe angegeben, für die Daten hinterlegt werden.
- Dem Ganzzahl-Attribut `Typ` wird der Zustand `HauptFahrbahn` zugeordnet. Das Referenz-Attribut `ErsatzMessQuerschnitt` zeigt auf das Objekt `mq.a10.0000`.

In den **Zeilen 13 - 16** werden der Menge `MessQuerschnitte` die Objekte `fs.mq.a10.0009.hfs` und `fs.mq.a10.0009.2üfs` zugeordnet.

Erst nach dem Konvertieren des Datenkatalogs und einem Neustart der Kernsoftware steht das neu angelegte Objekt dem System zur Verfügung.

3.3.3 Konfigurationsverantwortlichen erzeugen

3.3.3.1 Allgemeine Vorgehensweise

Das Anlegen eines Konfigurationsverantwortlichen entspricht dem Erzeugen eines Objekts. Dabei sind jedoch folgende Punkte zu beachten:

- Ein Konfigurationsverantwortlicher sollte im Konfigurationsbereich `kb.objekteSystemGlobal` (Versorgungsdatei `kb.objekteSystemGlobal.xml`) definiert werden (siehe Abschnitt 2.1.5.1).
- Der Objekttyp `typ.autarkeOrganisationsEinheit` sollte als Basis für den Konfigurationsverantwortlichen dienen. Dieser Typ sollte gewählt werden, da er Stand 01.08.2006 alle bisher umgesetzten Softwareeinheiten des AK VRZ unterstützt.
- Der Wert des Attributs `kodierung` der konfigurierenden Attributgruppe `atg.konfigurationsVerantwortlicherEigenschaften` muss auf einen unter allen Konfigurationsverantwortlichen eindeutigen Wert gesetzt werden. Ansonsten kann der Wert beliebig gewählt werden.

> **Anmerkung**
>
> Der Konfigurationsverantwortliche `kv.system` stellt einen Sonderfall dar: dieser Konfigurationsverantwortliche wird beim Start der Kernsoftware implizit verwendet, wenn kein anderer Konfigurationsverantwortlicher angegeben wurde. Außerdem wird die ID dieses Konfigurationsverantwortlichen bereits im XML-Datenkatalog festgelegt. Dies ist für andere Konfigurationsverantwortliche nicht vorgesehen! Ihnen wird automatisch eine ID beim Konvertieren des XML-Datenkatalogs zur `config.xml` zugeordnet.

3.3.3.2 Beispiel

Der folgende Konfigurationsverantwortliche unterstützt die Verwendung der bisher umgesetzten SWE *Archivsystem* ([TANFARS]) und *Protokolle und Auswertungen* ([TANFPUA]). Er sollte in der Datei `kb.objekteSystemGlobal.xml` eingefügt werden, in die bisher alle Konfigurationsverantwortlichen eingetragen wurden.

```
 1: <?xml version="1.0" encoding="ISO-8859-1"?>
 2: <konfigurationsBereich pid="kb.objekteSystemGlobal" name="GlobaleSystemObjekte"
 3:   verantwortlich="kv.system">
 4:   <objekte>
 5:   ...
 6:     <konfigurationsObjekt typ="typ.autarkeOrganisationsEinheit" id="" name="TutorialKV"
 7:       pid="kv.tutorial">
 8:       <datensatz pid="atg.konfigurationsVerantwortlicherEigenschaften">
 9:         <datum name="kodierung" wert="12345"/>
10:         <datenfeld name="defaultBereich"/>
11:       </datensatz>
12:       <objektMenge name="PuaSkripte"/>
13:     </konfigurationsObjekt>
14:   ...
15:   </objekte>
16: </konfigurationsBereich>
```

In **Zeile 9** wird dem Konfigurationsverantworltichen `kv.tutorial` eine unter allen Konfigurationsverantwortlichen eindeutige Kodierung zugewiesen. Der Wert wurde unter Berücksichtigung der anderen Konfigurationsverantwortlichen gewählt. Für das Feld `defaultBereich` (**Zeile 10**) der konfigurierenden Attributgruppe muss kein Wert eingetragen werden. In **Zeile 12** wird eine Menge erzeugt, die von der Softwareeinheit *Protokolle und Auswertungen* benötigt wird.

Das eben definierte Objekt kann als Vorlage für alle weiteren Konfigurationsverantwortlichen dienen — einzig die PID und der Wert der `kodierung` müssen angepasst werden.

3.4 Datenverteiler koppeln

Um zwei Datenverteiler miteinander zu verbinden (Abschnitt 2.2.3), müssen Änderungen am Datenkatalog sowie an den Start-Parametern der Kernsoftwarekomponenten vorgenommen werden.

3.4.1 Ergänzungen am Datenkatalog

Die Datenverteiler müssen im Datenmodell eingetragen werden. Dazu legt man für jeden Datenverteiler ein Konfigurationsobjekt vom Typ `typ.datenverteiler` an, mit dem der Datenverteiler beschrieben wird. Da die ID später im Aufrufparameter angegeben werden muss, empfiehlt sich, die ID im Konfigurationsobjekt fest zu vergeben. Die ID darf bis jetzt noch nicht vergeben sein.

```
1: <konfigurationsObjekt typ="typ.datenverteiler" id="10000"
2:         name="TestDatenverteiler0" pid="dav.test0">
3:     <datensatz pid="atg.datenverteilerEigenschaften">
4:         <datum name="adresse" wert="192.168.0.1"/>
5:         <datum name="subAdresse" wert="8080"/>
6:     </datensatz>
7: </konfigurationsObjekt>
```

In der konfigurierenden Attributgruppe `atg.datenverteilerEigenschaft` gibt man die Adresse (Hostname oder IP, **Zeile 4**) sowie die subAdresse (im Fall von TCP/IP den Port, typischerweise 8080, **Zeile 5**) an.

Die Verbindungen, die zwischen Datenverteilern bestehen sollen, werden mit dem Konfigurationsobjekt `typ.datenverteilerVerbindung` modelliert.

```
 1: <konfigurationsObjekt typ="typ.datenverteilerVerbindung" pid="davVerbindung.test0-1">
 2:     <datensatz pid="atg.datenverteilerTopologie">
 3:         <datum name="datenverteilerA" wert="dav.test0"/>
 4:         <datum name="datenverteilerB" wert="dav.test1"/>
 5:         <datum name="wichtung" wert="1"/>
 6:         <datum name="aktiverDatenverteiler" wert="-"/>
 7:         <datum name="ersatzverbindungsWartezeit" wert="60"/>
 8:         <datum name="benutzer1" wert="benutzer.testDatenverteiler"/>
 9:         <datum name="benutzer2" wert="benutzer.testDatenverteiler"/>
10:         <datenliste name="durchsatzPrüfung">
11:             <datum name="pufferFüllgrad" wert="75"/>
12:             <datum name="prüfIntervall" wert="180"/>
```

```
13:             <datum name="mindestDurchsatz" wert="3000"/>
14:         </datenliste>
15:     </datensatz>
16:     <objektMenge name="Ersatzverbindungen">
17:         <element pid="davVerbindung.test0-2"/>
18:     </objektMenge>
19: </konfigurationsObjekt>
```

Durch die konfigurierende Attributgruppe `atg.datenverteilerTopologie` wird nun angegeben, wie zwei Datenverteiler miteinander zu verbinden sind. Dabei wird mit `datenverteilerA` die PID des ersten Datenverteilers angegeben und unter `datenverteilerB` die des zweiten. Mit der Wichtung wird ein Wert mit dieser Verbindung assoziiert. Bei mehreren alternativen Routen durch den Datenverteilerverbund wird die Verbindung genommen, die die niedrigste Wichtung hat.

Letztendlich wird festgelegt, welcher der Datenverteiler der aktive ist. Dies kann entweder keiner sein (der Wert ist ein Strich: „-“), `A`, `B` oder `AB`. Im Fall von `AB` versuchen beide Datenverteiler aktiv die Verbindung zum anderen Datenverteiler aufzubauen, im Falle von „-“ keiner[2].

Die verbleibenden Werte der Attributgruppe sind in [DATK] beschrieben.

3.4.2 Aufrufparameter

Nachdem der Datenkatalog angepasst und konvertiert wurde, müssen noch Aufrufparameter in den Start-Skripten (siehe Abschnitt 3.1.1.2.2) ergänzt werden.

Parameter	Beschreibung
`-datenverteilerId=[zahl]`	Die nummerische ID des Datenverteilers. Der Wert kann der `config.xml`-Datei entnommen werden.
`-lokaleKonfiguration=[string]:[zahl]`	Die PID und die nummerische ID der lokalen Konfiguration, getrennt durch einen Doppelpunkt. Beispiel: `kv.system:1`

Tabelle 3-2: Parameter für den Datenverteiler

Parameter	Beschreibung
`-verantwortlich=[string]`	Die PID des Konfigurationsverantwortlichen, z.B. `kv.system`
`-konfigurationsBereich=[string]`	Die PID des Konfigurationsbereichs. Hier kann auch der Konfigurationsverantwortliche angegeben werden, um alle zugeordneten Bereiche zu wählen.

Tabelle 3-3: Parameter für die Konfiguration

3.5 Programmierung

Im diesem Abschnitt sollen beispielhaft Applikationen beschrieben werden, die die API der Kernsoftware nutzen um Daten über den Datenverteiler zu senden oder zu empfangen.

Der Abschnitt ist als Fortsetzung von Abschnitt 1.6 gedacht, der dieses Thema nur in aller Kürze behandelt. Dennoch kann auch hier nicht die Verwendung aller von der API unterstützten Funktionen beschrieben werden. Für detailliertere Informationen sei dafür auf die Dokumentation der API im Verzeichnis `dokumente` der Kernsoftwareinstallation ,[APIBIB] und [APISYS], hingewie-

[2] In letzterem Fall kommt es daher auch zu keiner Kopplung der Datenverteiler.

sen.

3.5.1 Verbindung zum Datenverteiler

Nachdem im Schnelleinstieg (Abschnitt 1.6.1 und Abschnitt 1.6.2) die Vorgehensweise zum Aufbauen und Trennen einer Verbindung zum Datenverteiler erklärt wurde, sollen in diesem Abschnitt einige weiterführende Aspekte erläutert werden.

Verbindungsparameter

Beim Aufbau der Verbindung zum Datenverteiler sind nicht nur die im Schnelleinstieg (Abschnitt 1.6.1) verwendeten Verbindungsparameter `davUser`, `davHost`, `davPassword` möglich, sondern u.a. auch folgende (teils zitiert aus [SSB]):

Parameter	Bedeutung
`-davAppPort=[zahl]`	TCP-Portnummer, unter der der Datenverteiler Verbindungen von Applikationen entgegennimmt.
`-davDavPort=[zahl]`	TCP-Portnummer, unter der der Datenverteiler Verbindungen von anderen Datenverteilern entgegennimmt.
`-benutzer=[zeichenkette]`	Benutzerkennung, unter der der Datenverteiler gestartet wird. Wenn der Datenverteiler seine Konfiguration über einen anderen Datenverteiler erhält wird diese Benutzerkennung ebenfalls beim Aufbau einer Datenverteilerapplikationsfunktionsverbindung zu dem entfernten Datenverteiler benutzt.
`-authentifizierung=[dateiname]`	Pfadangabe zu der Datei, in der die benutzerspezifischen Passwörter abgelegt ist.
`-lokaleKonfiguration=[zeichenkette]:[zahl]`	Legt den Konfigurationsverantwortlichen durch PID und ID fest.

Tabelle 3-4: Aufrufparameter für die Datenverteilerverbindung

Das folgende Listing zeigt, wie die Parameter zur Initialisierung an den Datenverteiler übergeben werden, wobei `parameter_1` bis `parameter_n` dabei für die in Tabelle 3-4 aufgeführten Parameter stehen:

```
// Argumente
String params           = new String[] {
                                  davHost,
                                  parameter_1,
                                  parameter_2,
                                  ...,
                                  parameter_n };
ArgumentList argList    = new ArgumentList(params);
ClientDavParameters cdp = new ClientDavParameters(argList);

// DAV-Verbindung
ClientDavConnection dav = new ClientDavConnection(cdp);
dav.connect();
dav.login(davUsername, davPassword);
```

Fehler

Beim Verbindungsaufbau zum Datenverteiler können die im Folgenden beschriebenen Fehler auftreten. Sie müssen von der Applikation abgefangen werden (teils zitiert aus [APIBIB]):

- *MissingParameterException*

  Ausnahme, die generiert wird, wenn notwendige Verbindungsparameter nicht spezifiziert wur-

den.

- *ConnectionException*

  Verbindungsausnahme, die generiert wird, wenn eine Verbindung nicht aufgebaut werden konnte.

- *CommunicationError*

  Ausnahme, die generiert wird, wenn bei der Kommunikation mit dem Datenverteiler Fehler aufgetreten sind.

- *InconsistentLoginException*

  Authentifizierungsausnahme, die bei einem fehlgeschlagenen Authentifizierungsversuch generiert wird.

3.5.2 Applikation zum Datenempfang

Der Abschnitt geht detailiert auf den im Schnelleinstieg (Abschnitt 1.6.3) eingeführten Empfang von Daten ein.

3.5.2.1 Anmeldung

Wie bereits im Abschnitt 1.6.3 beschrieben, muss eine Applikation zum Empfangen von Daten beim Datenverteiler als solche registriert, d.h. angemeldet werden. Grundsätzlich muss dazu die Funktion `subscribeReceiver(...)` des `ClientDavInterface` wie im folgenden Listing exemplarisch dargestellt, aufgerufen werden.

```
/** Anmeldung zum Empfang von Daten */
public void subscribe() {
    // Anmelden
    datenverteiler.subscribeReceiver(
                Empfaenger.this,
                systemobjekt,
                datenbeschreibung,
                anmeldeoption,
                rolle);

    System.out.println("Angemeldet.");
}
```

Dabei werden die Rolle und Anmeldeoptionen folgendermaßen festgelegt:

Rolle

Eine Applikation kann die Rollen *Empfänger* oder *Senke* einnehmen. Empfänger und Senke empfangen Daten, eine Senke legt zusätzlich den Datenverteiler für Verwaltungsaufgaben fest.[3]

- Empfänger

  Die Rolle *Empfänger* wird folgendermaßen gewählt:

```
// Rolle des Datenlieferanten: Empfaenger
rolle = ReceiverRole.receiver();
```

- Senke

  Um die Rolle *Senke* zu verwenden, ist folgende Methode zu verwenden:

```
// Rolle des Datenlieferanten: Senke
rolle = ReceiverRole.drain();
```

[3]Eine Senke oder das sendende Gegenstück, eine *Quelle*, bestimmt den Datenverteiler für Verwaltungsaufgaben. Falls für eine Datenidentifikation schon eine Quelle angemeldet ist, kann **keine** Senke mehr angemeldet werden, da nur ein Datenverteiler die Verwaltungsaufgaben übernehmen kann.

Ist bereits eine Senke oder Quelle für eine Datenidentifikation angemeldet, so **muss** sich eine Applikation, die Daten empfangen möchte, wie im Abschnitt 1.6.3 als Empfänger anmelden. Genaueres zum Konzept von Quelle und Senke findet sich in Abschnitt 2.2.2.3.

Anmeldeoption

Die in der Methode `subscribeReceiver(...)` des `ClientDavInterface` verwendete Anmeldeoption für eine Datenidentifikation (siehe oben) spezifiziert, welche Datensätze von der Applikation empfangen werden. Folgende Optionen sind möglich:

- *Normal*

  Mit dieser Option erfolgt die Anmeldung so, dass alle als *online* gekennzeichnete Datensätze der jeweiligen Datenidentifikation empfangen werden. Nachgelieferte Daten werden nicht bezogen.

```
// Anmeldeoption
anmeldeoption = ReceiveOptions.normal()
```

- *Delayed*

  Die Anmeldeoption `delayed()`entspricht der Option `normal()`, jedoch werden mit dieser Option auch Datensätze empfangen, die als *nachgeliefert* gekennzeichnet sind.

```
// Anmeldeoption
anmeldeoption = ReceiveOptions.delayed()
```

- *Delta*

  Im Gegensatz zur Option `normal()` bezieht sich eine Anmeldung mit der Option `delta()` nur auf geänderte Daten, d.h. nur wenn sich die Nutzdaten des aktuellen Datensatzes von den Nutzdaten des vorhergehenden unterscheiden, wird dieser empfangen. Nachgelieferte Daten werden nicht bezogen.

```
// Anmeldeoption
anmeldeoption = ReceiveOptions.delta()
```

Initialisierungsdetails

Der Methodenaufruf `getDataModel()` des `ClientDavInterface` liefert ein Java Stellvertreter-Objekt des Datenmodells, über das aus Java heraus auf die SWE Konfiguration zugegriffen werden kann.

Über dieses Java Objekt können beispielsweise Java Stellvertreter-Objekte für die Objekte, Attributgruppen und Aspekte des Datenmodells bezogen werden. Dazu wird das jeweilige Element des Datenmodells durch seine PID oder ID identifiziert. Weitere Methoden zum Zugriff auf das Datenmodell, z.B. zum Lesen von konfigurierende Datensätze einer Attributgruppe, können in [APIBIB] nachgeschlagen werden.

```
// Stellvertreterobjekt fuer ein Objekt mit PID 'objectPid'
SystemObject systemobjekt     = datenverteiler.getDataModel().getObject(objectPid);

// Stellvertreterobjekt fuer Attributgruppe mit PID 'atgPid'
AttributeGroup attributgruppe = datenverteiler.getDataModel().getAttributeGroup(atgPid);

// Stellvertreterobjekt fuer Aspekt mit PID 'aspPid'
Aspect aspekt                 = datenverteiler.getDataModel().getAspect(aspPid);
```

Existiert eine angegebene PID oder ID nicht, liefern `getObject(...)`, `getAttributeGroup(...)` oder `getAspect(...)` den Wert `null`.

3.5.2.2 Anmelden auf Änderungen der Parametrierung

Soll eine Applikation automatisch bei Änderungen an den durch die SWE Parametrierung verwalteten Daten benachrichtigt werden, muss sie sich als Empfänger auf die entsprechende

Datenidentifikation anmelden. Daten der SWE Parametrierung können also analog zu jeder anderen Datenidentifikation von der Applikation empfangen werden.

Beispiel: Um bei Änderungen an der Grundparametrierung (siehe Abschnitt 2.4) benachrichtigt zu werden, kann sich eine Applikation mit der Anmeldeoption `ReceiveOptions.delta()` auf die Datenidentifikation `kv.system:atg.parametrierung:asp.parameterIst:0` als Empfänger anmelden. Sobald die Parametrierung für diese Datenidentifikation geändert wird, empfängt die Applikation den geänderten Datensatz und kann ihn entsprechend verwerten.

3.5.3 Applikation zum Datenversand

In diesem Abschnitt wird anhand eines Beispiels eine Applikation zum Senden von Daten über den Datenverteiler umgesetzt.

Um Daten über den Datenverteiler zu versenden sind prinzipiell folgende Schritte nötig:

1. Verbindung zum Datenverteiler herstellen
2. Anmelden des Senders
3. Senden der Daten
4. Abmelden des Senders
5. Trennen der Datenverteilerverbindung

Anschließend soll auf die Verwendung der Sendesteuerung eingegangen werden.

3.5.3.1 Einfache Senderapplikation

Die in diesem Abschnitt implementierten Senderapplikation hat folgende Eigenschaften:

- Jeder Sender muss die Schnittstelle `ClientSenderInterface` implementieren. Dazu müssen die Methoden `dataRequest()` und `isRequestSupported()` implementiert werden.
- Der Sender verwendet die Klasse `DavConnection` als Verbindungsmanager. Die Klasse stellt die Funktionalität zum Herstellen und Trennen der Verbindung zum Datenverteiler bereit, siehe Abschnitt 1.6.1 und Abschnitt 1.6.2.
- Die Sendesteuerung wird verwendet, eine genauere Beschreibung erfolgt jedoch erst in Abschnitt 3.5.3.2.
- Die vollständigen Listings der in diesem Abschnitt verwendeten Klassen sind Bestandteil des Tutorials:
  - DavConnection.java (Abschnitt 1.7.1)
  - Sender.java (Abschnitt 3.5.5.1)

3.5.3.1.1 Überprüfen des Senders

Um zu überprüfen, dass die in diesem Beispiel implementierte Senderapplikation korrekt arbeitet, können die erzeugten Daten durch eine andere Applikation abgerufen werden. Das Empfangen ist auf eine der nachfolgenden Arten möglich:

- Empfängerapplikation, siehe Abschnitt 1.6.3 und Abschnitt 3.5.2
- Generischer Testmonitor mit Onlinetabelle, siehe Abschnitt 1.5.2

- Generischer Testmonitor mit Onlineprotokollierer, siehe Abschnitt 1.5.9 oder noch genauer in Abschnitt 3.2.1.1
- Generischer Testmonitor mit „Aktuelle Daten anzeigen“, siehe Abschnitt 1.5.4

3.5.3.1.2 Verbindungsaufbau und Initialisierung

Zum Aufbau der Verbindung zum Datenverteiler muss die Funktion `connect()` der Klasse `DavConnection` aufgerufen werden. Zumindest die Parameter für die Datenverteileradresse, Benutzername und Passwort müssen dabei gesetzt werden.[4]

Ist die Verbindung hergestellt, kann die Datenidentifikation analog zum Datenempfang (siehe Abschnitt 1.6.3.2) gesetzt werden.

3.5.3.1.3 Anmelden

Nun wird die Senderapplikation für die spezifizierte Datenidentifikation angemeldet. Erst nach der Anmeldung kann die Applikation Datensätze über den Datenverteiler senden.

```
public class Sender implements ClientSenderInterface {

    ...

    /** Anmeldung zum Senden von Daten */
    public void subscribe() {
        // Rolle der Applikation: Sender
        senderrolle = SenderRole.source();

        // Anmelden
        try {
            datenverteiler.subscribeSender(Sender.this, systemobjekt, datenbeschreibung,
                    senderrolle);
            System.out.println("Angemeldet.");

        } catch (OneSubscriptionPerSendData e) {
            System.out.println("Fehler: "+ e);
        }
    }

    ...

}
```

Im obigen Listing wird die Rolle *Quelle* verwendet. Soll eine Applikation als Sender und nicht als Quelle angemeldet werden, muss die entsprechende Zeile im Listing wie folgt umgeschrieben werden:

```
// Rolle der Applikation: Sender
senderrolle = SenderRole.sender();
```

Der Fehler `OneSubscriptionPerSendData` tritt auf, wenn bereits eine Sendeanmeldung für diese Datenidentifikation über die gleiche Datenverteilerverbindung durchgeführt wurde: pro Datenverteilerverbindung und Datenidentifikation darf es nur eine Sendeanmeldung geben.

3.5.3.1.4 Senden

Nachdem die Applikation angemeldet wurde, können Daten über den Datenverteiler gesendet werden. Dazu wird im Beispiel die Methode `send()` implementiert. Das im Listing vorgestellte Beispiel erzeugt die zu senden Datensätze dabei direkt in der `send()`-Methode. Eine genauere Beschreibung zur Handhabung von Datensätzen, insbesondere eine Erklärung möglicher Attribute und den Zugriff auf die Werte erfolgt in Abschnitt 3.5.4.

```
public class Sender implements ClientSenderInterface {

    ...

    /** Senden von Daten als Quelle */
    public void send() {
        // Zeitstempel
        long zeitstempel = System.currentTimeMillis();

        // Datensatz erstellen
```

[4] Weitere Parameter sind möglich (siehe Abschnitt 3.5.1), dies würde jedoch einige Anpassungen an der Klasse `DavConnection` erfordern.

```
        AttributeGroup atg = davConnection.getDataModel().getAttributeGroup(OBJECT_ATG);
        Data datensatz     = davConnection.createData(atg);

        // Den Wert des numerischen Attributs "Typ" setzen
        NumberValue nv = datensatz.getUnscaledValue("Typ");
        // wie im Datenkatalog angegeben: 2 = NebenFahrbahn
        nv.set(2);

        // Den Wert des Textattributs "ErsatzMessQuerschnitt" setzen
        TextValue tv = datensatz.getTextValue("ErsatzMessQuerschnitt");
        tv.setText("mq.a10.0001");

        // Telegramm erstellen
        ResultData datentelegramm = new ResultData(
                systemobjekt,
                datenbeschreibung,
                zeitstempel,
                datensatz);

        // Versenden
        try {
            datenverteiler.sendData(datentelegramm);
            System.out.println("Gesendet.\n\t"+datentelegramm);
        } catch (Exception e) {
            System.out.println("Fehler: " + e);
    }

    ...

}
```

Beim Versenden von Daten könnnen folgende Fehler auftreten (teils zitiert aus [APIBIB]):

- *DataNotSubscribedException*

  Ausnahme, die beim Senden von Datensätzen ohne entsprechende Sendeanmeldungen generiert wird.

- *ConfigurationException*

  Zeigt ein Problem bei der Kommunikation mit der SWE Konfiguration an.

- *SendSubscriptionNotConfirmed*

  Ausnahme, die beim Senden von Daten als einfacher Sender generiert wird, wenn noch keine positive Sendesteuerung vom Datenverteiler für die zu versendenden Daten vorliegt (siehe Abschnitt 3.5.3.2).

3.5.3.1.5 Abmelden und Trennen

Das Abmelden der Quelle kann wie folgt implementiert werden:

```
public class Sender implements ClientSenderInterface {

    ...

    /** Abmeldung vom Senden der Daten */
    public void unsubscribe() throws ConfigurationException {
        // Abmelden
        datenverteiler.unsubscribeSender(
                Sender.this,
                systemobjekt,
                datenbeschreibung);

        System.out.println("Abgemeldet.");
    }

    ...

    /** Verbindung zum Datenverteiler trennen */
    public void disconnect() {
        // Trennen
        DavConnection.disconnect(datenverteiler, false, "");

        System.out.println("Getrennt.");
    }

}
```

Beim Abmelden kann eine `ConfigurationException` auftreten, falls bei der Kommunikation mit der Konfiguration Fehler aufgetreten sind.

Zum Trennen der Datenverteilerverbindung wird die Methode `disconnect()` der Klasse `DavConnection` verwendet.

3.5.3.2 Sendesteuerung

Eine Klasse muss das Inferface `ClientSenderInterface` implementieren, um sich als Quelle oder Sender amelden zu können. Die Methoden `dataRequest(...)` und `isRequestSupported(...)` dieser Schnittstelle realisieren die Sendesteuerung:

- `isRequestSupported(SystemObject so, DataDescription dd)`

  Die Callback-Methode wird vom Datenverteiler aufgerufen um zu prüfen, ob die Anwendung die Sendesteuerung verwendet. Gibt sie `false` zurück, wird keine Sendesteuerung verwendet, gibt sie `true` zurück, wird die Sendesteuerung verwendet.

- `dataRequest(SystemObject so, DataDescription dd, byte state)`

  Die Callback-Methode wird ebenfalls vom Datenverteiler aufgerufen, falls die Sendesteuerung verwendet wird. Der Parameter `state` teilt der Applikation den Zustand der Sendesteuerung mit. Im folgenden Beispiel werden die Daten im Fall einer positiven Sendesteuerung durch Aufruf der oben beschriebenen `send()`-Methode auch sofort gesendet.

```
/**
 * Abstrakte Methode, die implementiert werden muss:
 * Übernimmt das tatsächliche Senden der Daten.
 */
public void dataRequest(SystemObject object, DataDescription dataDescription,
        byte state) {
    if (state == ClientSenderInterface.START_SENDING) {
        send();
    }
}

/**
 * Abstrakte Methode, die implementiert werden muss:
 * Zeigt, ob die Sendesteuerung erwünscht ist.
 */
public boolean isRequestSupported(SystemObject so, DataDescription dd) {
    return true;
}
```

Genaueres zum Konzept der Sendesteuerung ist im Abschnitt 2.2.2.5 nachzulesen.

3.5.4 Arbeiten mit Datensätzen

In diesem Abschnitt wird näher darauf eingegangen, wie Datensätze erzeugt werden, und wie mit ihnen gearbeitet werden kann. Anschließend wird der Zugriff auf die möglichen Grundtypen von atomaren Attributen erklärt, und danach wird im Detail erläutert, wie Felder von Attributen und Attributlisten verwendet werden.

3.5.4.1 Erzeugen von Datensätzen

Die Struktur eines Datensatzes, der über den Datenverteiler transportiert wird, wird alleine durch die Attributgruppe festgelegt. Jedem Attribut des Datensatzes muss ein Wert zugewiesen werden, bevor er auf den Datenverteiler gelegt wird, ansonsten wird beim Senden des Datensatzes eine Ausnahme erzeugt. Der folgende Code erzeugt einen leeren Datensatz für die Attributgruppe mit der PID `objectAtg`.

```
// Attributgruppe wählen
AttributeGroup atg = davConnection.getDataModel().getAttributeGroup(objectAtg);

// Datensatz für die Attributgruppe erzeugen
Data datensatz     = davConnection.createData(atg);
```

3.5.4.2 Atomare Werte

Es gibt fünf Grundtypen für atomare Attribute, die im Folgenden jeweils kurz beschrieben werden. In den folgenden Listings bezeichnet die Variable `datensatz` dabei eine Instanz eines Objekts vom Typ `Data`.

> **Wichtig**
>
> Der Entwickler einer Applikation muß selbst sicherstellen, dass nur passende Methoden auf die Attribute eines Datensatzes angewendet werden. D.h. auf Attribute, die Textwerte speichern, dürfen nur die Methoden für Textwerte, z.B. `getTextValue(...)` (s.u.), angewendet werden.

Textwerte

Das folgende Listing zeigt, wie Textwerte gesetzt werden können. `attributName` ist hier ein String, der den Namen des Attributs laut Datenmodell enthält.

```
TextValue tv = datensatz.getTextValue(attributName);

tv.setText("hallo");
```

Sollen Textwerte im Datensatz gesetzt werden, geschieht dies mit Hilfe eines `TextValue`-Objekts. Attributwerte können einen Zusatz, zum Beispiel Einheiten, enthalten. Beim lesenden Zugriff auf den Textwert kann gewählt werden, ob der Zusatz mitgeliefert werden soll:

```
// Liefert den Wert mit Zusatz
String mitZusatz  = tv.getText();

// Liefert den Wert ohne Zusatz
String ohneZusatz = tv.getValueText();
```

Ganzzahlige Werte

Das nächste Listing zeigt, wie Ganzzahl-Werte ohne Berücksichtigung der Skalierung (siehe Abschnitt 2.1.2.4.1) gesetzt werden können:

```
NumberValue nv = datensatz.getUnscaledValue(attributName);

nv.set(2);
```

Der lesende Zugriff auf das Attribut ist durch folgende Aufrufe möglich:

```
// Zahlwert als Integer
int wert  = nv.intValue();

// Zahlwert als Long
long wert = nv.longValue();
```

Soll die Skalierung verwendet werden, ist die Methode `getScaledValue(attributName)` zu verwenden.

Fließkomma-Werte

Das Listing zeigt, wie Fließkomma-Werte gesetzt werden können:

```
NumberValue nv = datensatz.getUnscaledValue(attributName);

nv.set(17.35);
```

Der lesende Zugriff auf das Attribut ist folgendermaßen möglich:

```
// Zahlwert als Float
nv.floatValue();

// Zahlwert als Double
nv.doubleValue();
```

Zeitstempel

Um ein Attribut, das Zeitstempel speichert, zu verändern kann wie folgt vorgegangen werden:

```
TimeValue tv = datensatz.getTimeValue(attribut);

// Zeitwert in Millisekunden
tv.setMillis(123456789);

// Zeitwert in Sekunden
tv.setSeconds(123456);
```

Die Funktion `setMillis(...)` setzt den Wert auf die im Datum enthaltene Zeit in Millisekunden, `setSeconds(...)` erledigt analog das Setzen des Wertes auf einen Sekundenwert.

Der lesende Zugriff auf die Werte erfolgt durch Aufruf der Methoden `getMillis()` oder `getSeconds()`:

```
// Millisekunden-Zeitwert lesen
long timeMS = tv.getMillis();

// Sekunden-Zeitwert lesen
long timeS = tv.getSeconds();
```

Referenzen

Ein Attribut kann eine Referenz auf ein Objekt des Datenmodells enthalten. Das Setzen einer Referenz geschieht über einen Aufruf der Methode `set(...)`, der ein `SystemObject`[5] übergeben werden muss:

```
ReferenceValue rv = datensatz.getReferenceValue(attribut);

rv.set( (SystemObject) so );
```

Der Wert des Referenzattributs kann als ID oder Stellvertreter-Objekt ausgelesen werden:

```
// Id des referenzierten Objekts
long         id = rv.getId();

// Stellvertreter-Objekt
SystemObject so = rv.getSystemObject();
```

- `getId()`

  Liefert die Id des referenzierten Objekts oder `0`, wenn kein Objekt referenziert wird.

- `getSystemObject()`

  Liefert ein Java Stellvertreter-Objekt für das referenzierte Objekt oder `null`, wenn kein Objekt referenziert wird.

3.5.4.3 Felder und Attributlisten

Datensätze können nicht nur atomare Attribute enthalten, sondern auch Felder von Attributen oder Attributlisten. Felder und Attributlisten können verschachtelt werden, ein Feld bzw. eine Attributliste kann als Elemente also wiederum Felder oder Attributlisten enthalten.

Felder

Eine Anzahl Attribute gleichen Typs können in einem Feld zusammengefasst werden (siehe Abschnitt 2.1.2.4.2). Auf das Feld bzw. seine Elemente kann wie folgt zugegriffen werden:

```
// Feld von Textwerten
TextArray      ta = datensatz.getTextArray(feldName);
TextValue      tv = ta.getTextValue(index);

// Feld von Zahlwerten
NumberArray    na = datensatz.getUnscaledArray(feldName);
NumberValue    nv = na.getValue(index);

// Feld von Zeitstempeln
TimeArray      ta = datensatz.getTimeArray(feldName);
TimeValue      rv = ta.getTimeValue(index);

// Feld von Referenzen
```

[5] Ein Objekt der Klasse `SystemObject` ist ein Java Stellvertreter-Objekt für ein Objekt des Datenmodells.

```
ReferenceArray ra = datensatz.getReferenceArray(feldName);
ReferenceValue rv = ra.getReferenceValue(index);

// Verschachtelte Felder und Listen
Data        field = datensatz.getItem(feldName);

// Inneres Feld
TextArray  ta = field.getTextArray(innererFeldName);
```

Im Listing kennzeichnet

- `feldName`

  den Namen des Feldes als String.

- `index`

  den Index des gewünschten Elements als Integer.

Felder bieten die Methoden `getLength()` und `setLength(...)` an um die Feldgrösse zu lesen oder zu manipulieren. Wird das Feld vergrößert, werden Elemente mit Wert `null` angehängt, wird es verkleinert, werden Einträge von hinten her gelöscht.

Attributlisten

Auch in einer Attributliste kann eine Anzahl von Attributen zusammengefasst werden. Im Unterschied zu einem Feld müssen die Elemente einer Attributliste nicht notwendigerweise vom gleichen Typ sein. Der Name der Attributliste, hier `listName`, wird verwendet, um auf die Attributliste zuzugreifen:

```
// Zugriff auf die Liste
Data list = datensatz.getItem(listenName);

// Atomare Elemente der Liste
TextValue      tv = list.getTextValue(textElementName);
NumberValue    nv = list.getValue(nummerElementName);
TimeValue      rv = list.getTimeValue(zeitElementname);
ReferenceValue rv = list.getReferenceValue(referenzElementName);

// Verschachtelte Felder
Array  innerField = list.getArray(arrayName);

// Verschachtelte Listen
Data    innerList = list.getItem(innererlistenName);
```

3.5.5 Programlistings

In diesem Abschnitt sind die in den Programmierbeispielen verwendeten Klassen aufgeführt. Zu einigen der besprochenen Klassen finden sich die vollständigen Listings bereits im Schnelleinstieg (Abschnitt 1.7), falls sie dort schon eingeführt wurden.

Die Klassen unterscheiden sich in manchen Details von den Listings, die im Abschnitt 3.5.2 und im Abschnitt 3.5.3 verwendet wurden. In diesen Fällen wurden die Klassen der Einfachheit und prägnanten Darstellung halber geändert. Es sind jedoch ohne Einschränkungen beide Möglichkeiten denkbar und in der Praxis verwendbar.

3.5.5.1 Sender.java

Das folgende Listing zeigt den vollständigen Quelltext zu der in Abschnitt 3.5.3 implementierten Senderapplikation:

```
import stauma.dav.clientside.ClientDavInterface;
import stauma.dav.clientside.ClientSenderInterface;
import stauma.dav.clientside.Data;
import stauma.dav.clientside.DataDescription;
import stauma.dav.clientside.DataNotSubscribedException;
import stauma.dav.clientside.ResultData;
import stauma.dav.clientside.SenderRole;
import stauma.dav.common.OneSubscriptionPerSendData;
import stauma.dav.common.SendSubscriptionNotConfirmed;
import stauma.dav.configuration.interfaces.Aspect;
import stauma.dav.configuration.interfaces.AttributeGroup;
import stauma.dav.configuration.interfaces.ConfigurationException;
```

```
import stauma.dav.configuration.interfaces.SystemObject;

/**
 * Diese Klasse realisiert einen einfachen Sender (als Quelle), für Tutorial :
 * Praxisteil.
 *
 * @author beck et al. projects GmbH
 */
public class Sender implements ClientSenderInterface {

    // Objekt für das Daten generiert werden sollen
    protected final String OBJECT_PID = "mq.a10.0000";
    protected final String OBJECT_ATG = "atg.messQuerschnittAllgemein";
    protected final String OBJECT_ASP = "asp.eigenschaften";
    protected final short  OBJECT_SMV = 0;

    // Handles für die Verbindung zum Datenverteiler
    protected ClientDavInterface     datenverteiler;
    protected SystemObject           systemobjekt;
    protected AttributeGroup         attributgruppe;
    protected Aspect                 aspekt;
    protected DataDescription        datenbeschreibung;
    protected short                  simulationsvariante;
    protected SenderRole             senderrolle;

    /** Verbindung zum Datenverteiler herstellen */
    public void connect() {
        // Parameter für den Datenverteiler
        String davHost     = "-datenverteiler=localhost:8083";
        String davUsername = "Tester";
        String davPassword = "geheimespasswort";

        // Verbinden
        datenverteiler = DavConnection.connect(davHost, davUsername, davPassword);

        System.out.println("Verbunden.");
    }

    /** Verbindung zum Datenverteiler trennen */
    public void disconnect() {
        // Trennen
        DavConnection.disconnect(datenverteiler, false, "");

        System.out.println("Getrennt.");
    }

    /** Handles initialisieren */
    public void init() {
        senderrolle = SenderRole.source(); // Rolle der Applikation (Quelle,
        // Sender)

        systemobjekt = datenverteiler.getDataModel().getObject(OBJECT_PID);
        attributgruppe = datenverteiler.getDataModel().getAttributeGroup(OBJECT_ATG);
        aspekt = datenverteiler.getDataModel().getAspect(OBJECT_ASP);
        simulationsvariante = OBJECT_SMV;

        datenbeschreibung = new DataDescription(attributgruppe, aspekt,
                simulationsvariante);

        System.out.println("Initialisiert.");
    }

    /** Anmeldung zum Senden von Daten */
    public void subscribe() {
        // Anmelden
        try {
            datenverteiler.subscribeSender(Sender.this, systemobjekt, datenbeschreibung,
                    senderrolle);
            System.out.println("Angemeldet.");
        } catch (OneSubscriptionPerSendData e) {
            System.out.println("Datenidentifikation ist bereits angemeldet.");
        }

    }

    /** Abmeldung vom Senden der Daten */
    public void unsubscribe() {
        // Abmelden
        datenverteiler.unsubscribeSender(Sender.this, systemobjekt, datenbeschreibung);

        System.out.println("Abgemeldet.");
    }

    /** Senden von Daten als Quelle */
    public void send() {
        // Zeitstempel
        long zeitstempel = System.currentTimeMillis();

        // Datensatz erstellen
        AttributeGroup atg = davConnection.getDataModel().getAttributeGroup(OBJECT_ATG);
        Data datensatz     = davConnection.createData(atg);
```

```
        // Den Wert des numerischen Attributs "Typ" setzen
        NumberValue nv = datensatz.getUnscaledValue("Typ");
        // wie im Datenkatalog angegeben: 2 = NebenFahrbahn
        nv.set(2);

        // Den Wert des Textattributs "ErsatzMessQuerschnitt" setzen
        TextValue tv = datensatz.getTextValue("ErsatzMessQuerschnitt");
        tv.setText("mq.a10.0001");

        // Telegramm erstellen
        ResultData datentelegramm = new ResultData(systemobjekt, datenbeschreibung,
                zeitstempel, datensatz);

        // Versenden
        try {
            datenverteiler.sendData(datentelegramm);
            System.out.println("Gesendet.\n\t" + datentelegramm);
        } catch (Exception e) {
            System.out.println("Fehler: " + e.getMessage());
        }
    }

    /**
     * Abstrakte Methode, die implementiert werden muss:
     * Übernimmt das tatsächliche Senden der Daten
     */
    public void dataRequest(SystemObject object, DataDescription dataDescription,
            byte state) {
        if (state == ClientSenderInterface.START_SENDING) {
            send();
        }
    }

    /**
     * Abstrakte Methode, die implementiert werden muss:
     * Zeigt, ob die Sendesteuerung erwünscht ist.
     */
    public boolean isRequestSupported(SystemObject so, DataDescription dd) {
        return true;
    }


    /**
     * exemplarische main()-Methode für den Sender
     */
    public static void main(String[] args) throws Exception {
        Sender sender = new Sender();
        sender.connect();
        sender.init();
        sender.subscribe();
        // Warten, bis der Sender angemeldet ist
        // und der erzeugte Datensatz gesendet wurde
        Thread.sleep(5000);
        sender.unsubscribe();
        sender.disconnect();
    }
}
```

3.5.5.2 Datenerzeuger.java

Die Klasse `Datenerzeuger` kapselt das Erzeugen von Daten, die dann z.B. mit einem Sender über den Datenverteiler versendet werden können. In den bisherigen Beispielen werden Daten direkt in der `send()`-Methode des Senders erzeugt, weshalb der Datenerzeuger dort nicht benutzt wird. Möchte man jedoch komplexere Daten generieren, kann es sinnvoll sein, die Datenerzeugung mit Hilfe einer eigenen Klasse zu erledigen. Das folgende Listing zeigt eine solche Klasse exemplarisch.

Im Listing wird in der Methode `createData()` ein Datensatz zu einem Messquerschnitt erzeugt. Dabei wird ein numerisches Attribut und ein Textattribut gesetzt. Der hier vorgestellte Datenerzeuger erzeugt in Intervallen von fünf Sekunden den immer gleichen Datensatz für den Messquerschnitt.

```
import stauma.dav.clientside.ClientDavInterface;
import stauma.dav.clientside.Data;
import stauma.dav.clientside.Data.NumberValue;
import stauma.dav.clientside.Data.TextValue;
import stauma.dav.configuration.interfaces.AttributeGroup;
import stauma.dav.configuration.interfaces.ConfigurationException;

/**
 * Diese Klasse realisiert einen einfachen Datengenerator, für Tutorial :
 * Praxisteil.
 *
 * @author beck et al. projects GmbH
```

```
 */
public class Datengenerator {

     /**
      * Erzeugt Daten zum oben angegebenen Objekt
      */
     public Data createData(ClientDavInterface davConnection)
               throws ConfigurationException {
          // Datenbeschreibung setzen (Attributgruppe)
          AttributeGroup atg = davConnection.
                                   getDataModel().
                                      getAttributeGroup(
                                         "atg.messQuerschnittAllgemein");

          Data datensatz = davConnection.createData(atg);

          // Den Wert eines numerischen Attributs setzen
          NumberValue nv = datensatz.getUnscaledValue("Typ");
          // laut Datenkatalog bedeutet: 2 = NebenFahrbahn
          nv.set(2);

          // Den Wert eines Textattributs setzen
          TextValue tv = datensatz.getTextValue("ErsatzMessQuerschnitt");
          tv.setText("mq.a10.0001");

          System.out.println("Datensatz erzeugt.");

          // Erzeugte Daten zurückliefern
          return datensatz;
     }

    /**
     * exemplarische main()-Methode für den Generator
     */
     public static void main(String[] args) throws Exception {
          // Parameter für den Datenverteiler
          String davAddress  = "-datenverteiler=localhost:8083";
          String davUsername = "Tester";
          String davPassword = "geheimespasswort";
          // Verbinden zum Datenverteiler
          ClientDavInterface datenverteiler = DavConnection.connect(
                    davAddress,
                    davUsername,
                    davPassword);

          // Daten erzeugen (Endlos-Schleife)
          Datengenerator datengenerator = new Datengenerator();
          while (true) {
               // 5 Sekunden warten
               Thread.sleep(5000);

               // Datensatz erzeugen und ausgeben
               Data datensatz = datengenerator.createData(datenverteiler);
               System.out.println("\t" + datensatz);
          }
     }
}
```

3.6 Problembehebung

Alle Komponenten der Kernsoftware erstellen Logdateien. Diese befinden sich im Ordner `debug` im Hauptverzeichnis der Installation. Die Logdateien werden dabei mit dem Namen der Applikation, sowie zwei Zahlen versehen.

Der Dateiname ist wie folgt aufgebaut:

`<Applikationsname>-<Instanznummer>-<Nummer>.log`, z.B. `Datenverteiler-0-0.log`.

Der Name gibt dabei an, zu welcher Applikation diese Logdatei gehört. Wenn mehrere Instanzen der Applikation gleichzeitig laufen, wird die Instanznummer inkrementiert, so dass jede Instanz ein eigenes Logfile hat. Wenn nur eine Instanz läuft, ist diese Nummer 0. Die zweite Zahl nummeriert die Logdateien pro Instanz durch. Die Datei mit der laufenden Nummer 0 ist die aktuelle Datei, je größer diese Nummer ist, umso älter ist die Datei.

Außerdem existiert zu jeder Datei, die gerade in Verwendung ist, eine Sperr-Datei. Diese hat den Namen der Logdatei, erweitert um `.lck`. Dieser Mechnismus verhindert, dass mehrere Instanzen einer Kernsoftwarekomponente in die selbe Logdatei schreiben. Existiert eine Sperr-Datei, wird die Instanzzahl um Eins erhöht und ein weiteres Logfile angelegt.

Wenn eine Datei eine vorgegebene Größe erreicht (einstellbar mit Aufrufparameter -

debugFileSize), wird die Logdatei abgeschlossen, die <Nummer> aller bisher existierenden Logdateien um eins erhöht, und eine neue Datei mit der <Nummer> 0 angelegt. Dieser Vorgang wird Logrotation genannt. Wenn eine bestimmte Anzahl an Logdateien von einer Anwedung existiert (die Anzahl kann über -debugFileCount eingestellt werden), wird bei der Logrotation die älteste Datei (also die Datei, mit der größten Nummer) gelöscht.

Jeder Eintrag in den Logdateien enthält eine fortlaufende Nummer, das Datum, eine Fehlerklasse, die Java-Klasse, die die Meldunge verursacht hat, sowie die Meldung selbst:

```
#000003 03.08.2006 13:34:12,293:+0200 (TID:000010) ######################
FEHLER : Datenverteiler.stauma.dav.common.TCP_IP_ServerCommunication
Fehler beim anlegen eines TCP-Server-Sockets auf Port 8083:
     java.net.BindException: Address already in use
```

In diesem Beispiel wurde die Meldung am 03.08.06 um 13:34 Uhr erstellt. Die Meldungsklasse weist auf den Fehler hin und besagt, dass der TCP-Port bereits belegt ist. Gemeldet wurde dies von der Java-Klasse Datenverteiler.stauma.dav.common.TCP_IP_ServerCommunication.

Es gibt sieben Meldungsklassen, die hierarchisch angeordnet sind.

Meldungsklasse	Bedeutung	Einstellungsklasse
FEHLER	Höchster Level. Es ist eine Situation aufgetreten, in der die Software nicht weiterarbeiten kann.	ERROR
WARNUNG	Bei Warnung handelt es sich um eine kritische Situation, die aber (noch) nicht den Betrieb aktiv stört.	WARNING
INFO	Zusätzliche Informationsausgaben	INFO
KONFIG	Konfigurationsinformationen	CONFIG
FEIN	Programmnahe Ausgaben zur Verfolgung des Programmablaufs	FINE
FEINER	Wie FINE, jedoch mehr Details	FINER
DETAIL	Niedrigster Level: Alle Logausgaben	FINEST

Tabelle 3-5: Meldungsklassen, sortiert nach Kritikalität

Dabei ist FEHLER die kritischte Meldung, wohingegen DETAIL die am wenigsten kritische Meldung ist. Die Meldungen dazwischen sind entsprechend nach ihrere Kritikalität sortiert.

Die Einstellungsklasse gibt die Werte an, die als Argument (für -debugLevelStdErrText bzw -debugLevelFileText) für die Aufrufparameter in den Start-Skripten verwendet werden können. Hierbei ist zu beachten, dass jede Klasse automatisch kritischere Klassen mit einschließt. Das heißt, wenn als Argument INFO angegeben wird, werden Meldungen der Klassen INFO, WARNUNG und FEHLER ausgegeben, nicht jedoch CONFIG und weniger kritische Meldungen.

3.6.1 Analyse der Logfiles

Die Logdateien sind eine der wichtigsten Resourcen bei der Fehlersuche. Besonderes Augenmerk sollte auf Meldungen der Klassen WARNUNG und FEHLER gelegt werden:

- Bei WARNUNG handelt es sich um eine kritische Situation, die aber (noch) nicht den Betrieb aktiv stört. Sie weist jedoch auf ein eventuell auftretendes Problem hin.
- Meldungen, die mit FEHLER klassifiziert wurden, stellen ein aktuell existierendes Problem dar, das ein Eingreifen durch den Systemadministrator erforderlich macht. In der Regel arbeitet die ensprechende Software nicht mehr oder nicht mehr richtig.

Wenn eine Anwendung nicht mehr wie erwartet läuft (oder gar nicht mehr läuft), sollte als erstes

die Logdatei der entsprechenden Anwendung durchgesehen werden, ob hier eine Meldung mit `WARNUNG` oder `FEHLER` steht. Diese Informationen sollten schon einen Hinweis darauf geben, was für ein Fehler aufgetreten ist. Ebenso von Interesse könnten die Einträge davor und danach sein, weil sich daraus ableiten lässt, was das System gerade gemacht hat.

Nachdem eine Logmeldung gefunden wurde, die auf den Fehler hinweist, sollte man auch die Logs anderer Komponenten, die mit der betroffenen Softwareeinheit zusammenarbeitet, ansehen. Dies ist in der Regel der Datenverteiler oder die Konfiguration. Anhand des Zeitstempels der ursprünglichen Meldung lässt sich herausfinden, welche Meldung zur selben Zeit in der anderen Softwareeinheit generiert wurde. Sollten die Komponenten auf unterschiedlichen Rechnern laufen, setzt dies voraus, dass die Systemzeiten identisch sind.

Wie detailliert die Meldungen in die Logdatei geschrieben werden, wurde beim Start der Anwendung festgelegt. Prinzipiell gilt, dass ein niedriges Loglevel (z.B. `FINEST`) viele Logmeldungen generiert, was dazu führt, dass das System langsamer läuft und dass wichtige Meldungen in der Flut von Logeinträgen untergehen. Andererseits ist man, wenn ein Fehler auftritt, auf die Logmeldungen angewiesen. Es hat sich bewährt, für den normalen Betrieb `INFO` zu verwenden.

3.6.2 Sonstiges

Manche Meldungen, insbesondere Fehlermeldungen von der Java Virtual Machine, stehen unter Umständen nicht in den Logdateien, sondern werden nur auf Standard-Out oder Standard-Error ausgegeben. In den meisten Fällen werden diese Meldungen im Terminal-Fenster angezeigt, von dem aus die Anwendung gestartet wurde.

Auch diese Informationen sind relevant und sollten wie Logmeldungen der Kategorie `FEHLER` behandelt werden. Mögliche Meldungen sind zum Beispiel `OutOfHeapSpace` (es wurde mehr Arbeitsspeicher benötigt als zur Verfügung stand) oder `UnsupportedClassVersionError` (die eingesetzt JVM ist zu alt und kann die Klassendatei nicht laden).

Kapitel 4 Bibliographie

[AFO]
Anwenderforderungen, Dokument SE-02.00.00.00.00-AFo-4.0[1], Stand 21.01.2006

[APIBIB]
Kernsoftware-API Bibliotheksfunktionen[2], Stand 20.03.2006

[APISYS]
Kernsoftware-API Systemfunktionen[3], Stand 20.03.2006

[DATK]
HTML-Datenkatalog, Dokument SE-02.00.00.00.00-DatK[4], Stand 24.03.2006

[KSGLOS]
Glossar, Dokument SE-02.0002-Glos-0.4[5], Stand 23.08.2000

[SSB]
Schnittstellenbeschreibung, Dokument SE-02.00.00.00.00-SSB-0.1[6], Stand 18.04.2006

[TANFARS]
Technische Anforderungen - Archivsystem, Dokument SE-02.03.00.00.00-TAnf-2.5[7], Stand 15.09.2005

[TANFPUA]
Technische Anforderungen - Protokolle und Auswertungen, Dokument SE-02.09.00.00.00-TAnf-2.0[8], Stand 19.08.2005

[TPUK]
Technische Anforderungen - Segment Parametrierung und Konfiguration (PuK), Dokument SE-02.08.01.00.00-TAnf-0.1[9], Stand 05.04.2005

[1] akvrz/SE-02.00.00.00.00-AFo-4.0%5BAnwenderforderungen%5D.pdf
[2] akvrz/API/bibilothek/index.html
[3] akvrz/API/system/index.html
[4] akvrz/DatK/start.html
[5] akvrz/SE-02.0002-Glos-0.4%20%5BGlossar%20%28global%29%5D.pdf
[6] akvrz/SSB/start.html
[7] akvrz/SE-02.03.00.00.00-TAnf-2.5%20%5BTAnf%20ArS%5D.pdf
[8] akvrz/SE-02.09.00.00.00-TAnf-2.0%20%5BTAnf%20PuA%5D.pdf
[9] akvrz/TAnfPuK/start.html