



# Rahmenwerk 2.0

- Services
- Einstellungen
- Berechtigungen
- Anmeldung
- Urlasser
- Menüs und Symbolleisten
- Willkommen-Bildschirm
- Statische Informationen
- Standard-Legende
- Simulation
- Hinweise zur Migration

# Services

---

## — Allgemeines —

---

Die Funktionalität des Rahmenwerks wird im Wesentlichen über OSGI-Services bereitgestellt, diese Ersetzen die bisher verwendeten Singletons wie DaVVerbindung, EinstellungsSpeicher, OberflächenFunktionen, ...

Der Zugriff auf die Services kann auf verschiedenen Wegen erfolgen:

---

## Service-Komponente

---

Ein Plug-in, das Rahmenwerk-Services benötigt installiert eine Komponente, die von den entsprechenden Services abhängig ist. Die Komponente wird im Rahmen der Initialisierung des OSGI-Frameworks mit den notwendigen Informationen versorgt und aktiviert, wenn alle Voraussetzungen für ihren Betrieb vorliegen.

Beispielhaft ist hier die entsprechende Komponente im Plug-in "Migrationssupport" dargestellt, über die alle grundlegenden Services des Rahmenwerks bereitgestellt werden (Datei rahmenwerkservice.xml).

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
    name="de.bsvrz.buv.rw.compatibility">
    <implementation class="de.bsvrz.buv.rw.compatibility.internal.RahmenwerkService"/>
    <reference bind="bindRahmenwerk" cardinality="1..1"
        interface="de.bsvrz.buv.rw.basislib.Rahmenwerk"
        name="Rahmenwerk" policy="static"
        unbind="unbindRahmenwerk"/>
    <reference bind="bindBerechtigungen" cardinality="1..1"
        interface="de.bsvrz.buv.rw.basislib.berechtigung.Berechtigungen"
        name="Berechtigungen"
        policy="static"
        unbind="unbindBerechtigungen"/>
    <reference bind="bindEinstellungen" cardinality="1..1"
        interface="de.bsvrz.buv.rw.basislib.einstellungen.Einstellungen"
        name="Einstellungen"
        policy="static"
        unbind="unbindEinstellungen"/>
</scr:component>
```

Der entsprechende Quellcode sieht dann folgendermaßen aus:

```
public class RahmenwerkService {

    private static RahmenwerkService service;
    private Rahmenwerk rahmenWerk;
    private Berechtigungen berechtigungen;
    private Einstellungen einstellungen;

    public static RahmenwerkService getService() {
        return RahmenwerkService.service;
    }

    protected void activate() {
        RahmenwerkService.service = this;
    }

    protected void deactivate() {
        RahmenwerkService.service = null;
    }

    protected void bindRahmenwerk(final Rahmenwerk newRahmenWerk) {
        this.rahmenWerk = newRahmenWerk;
    }

    @SuppressWarnings("unused")
    protected void unbindRahmenwerk(final Rahmenwerk oldRahmenWerk) {
        this.rahmenWerk = null;
    }

    public Rahmenwerk getRahmenWerk() {
        return rahmenWerk;
    }

    protected void bindBerechtigungen(final Berechtigungen newBerechtigungen) {
        this.berechtigungen = newBerechtigungen;
    }

    @SuppressWarnings("unused")
    protected void unbindBerechtigungen(final Berechtigungen oldBerechtigungen) {
        this.berechtigungen = null;
    }

    public Berechtigungen getBerechtigungen() {
        return berechtigungen;
    }
}
```

```

}

protected void bindEinstellungen(final Einstellungen newEinstellungen) {
    this.einstellungen = newEinstellungen;
}

@SuppressWarnings("unused")
protected void unbindEinstellungen(final Einstellungen oldEinstellungen) {
    this.einstellungen = null;
}

public Einstellungen getEinstellungen() {
    return einstellungen;
}
}

```

Die Komponente muss noch in in der MANIFEST.MF eingebunden werden:

```

....
Service-Component: OSGI-INF/rahmenwerkservice.xml
....

```

Für die Definition und Einbindung der Komponenten setehen in der Eclipse-IDE entsprechende Hilfsmittel zur Verfügung!

Das Plug-in, das die Komponente installiert, hat damit volle Kontrolle über die Verfügbarkeit der notwendigen Services.

---

## Zugriff über den Bundle-Aktivator

---

Der Aktivator eines Bundles (Plug-ins) hat Zugriff auf die Service-Registry des Rahmenwerks. Aus dieser kann der gewünschte Service ermittelt werden.

Nachstehend wird beispielhaft der verfügbare Service *Rahmenwerk* ermittelt:

```

public Rahmenwerk getRahmenwerk() {
    final IEclipseContext serviceContext = EclipseContextFactory
        .getServiceContext(getBundle().getBundleContext());
    return serviceContext.get(Rahmenwerk.class);
}

```

Zu beachten ist dabei, dass der Zugriff auf die Services auf diesem Weg erst erfolgen kann, wenn das Plug-in aktiviert wurde. Insbesondere sollten keine Klassen die über ExtensionPoints instantiiert werden auf

die Services zurückgreifen.

---

## Zugriff über Dependency Injection

---

Der eleganteste Weg wäre der Zugriff auf die Services wäre die Verwendung von Dependency Injection, die integraler Bestandteil der E4 Plattform ist.

Da jedoch auf Grund des aktuellen Status der E4 Komponenten und Entwicklungswerkzeuge, sowie im Hinblick auf die möglichst hohe Kompatibilität der bestehenden BuV-Software-Komponenten, entschieden wurde die Eclipse RCP Plattform im Kompatibilitätsmodus zu Eclipse 3 zu betreiben, stehen die entsprechenden Möglichkeiten nicht vollumfänglich zur Verfügung.

Prinzipiell ist es jedoch möglich neue Plug-ins und Komponenten als Erweiterung des vorhandenen Applikationsmodell als reine E4 modellbasierte Bestandteile zu integrieren oder alternativ die Eclipse3-Bridge aus dem E4-Tools-Projekt einzusetzen, um die Möglichkeiten des Dependency Injection-Mechanismus zu verwenden.

Entsprechende Literatur und Online-Artikel sind in großem Umfang vorhanden, deshalb wird hier nicht näher darauf eingegangen.

Nachstehend sind die grundlegenden vom Rahmenwerk bereitgestellten Services überblicksmäßig dargestellt.

---

## — Rahmenwerk —

---

Der Service stellt im Wesentlichen die Verbindung zum Datenverteilersystem zur Verfügung mit dem die Oberfläche verbunden werden soll.

Daneben werden einige Informationen zur allgemeinen Anwendungsumgebung geliefert, die optional auch im Offline-Betrieb zur Verfügung stehen.

```
public interface Rahmenwerk {

    String JOB_FAMILY = "RahmenwerkJobs";

    ArgumentList getStartParameter();
    String getBenutzerName();
    SystemObject getBenutzer();

    boolean isOnline();
    ClientDavInterface getDavVerbindung();

    void addDavVerbindungsListener(DavVerbindungsListener listener);
    void removeDavVerbindungsListener(DavVerbindungsListener listener);

    String getPassword();
    boolean usesBerechtigungenNeu();

    SystemObject getOberflaechenObject();
    RwToolBarManager getRwToolBarManager();
}
```

---

## isOnline und getDavVerbindung

---

dient zur Überprüfung und zum ermitteln der potentiell vorhandenen Datenverteilterverbindung. Der Service bietet außerdem die Möglichkeit, sich für die Benachrichtigung über den Zustandswechsel der Datenverteilterverbindung zu registrieren.

---

## add/removeDavVerbindungsListener

---

dient zur Anmeldung eines Listeners, der über den Status der Datenverteilterverbindung des Rahmenwerks informiert wird.

```

public interface DavVerbindungsListener {

    /**
     * es wurde eine Datenverteilterverbindung hergestellt.
     *
     * @param event
     *           die Daten des Ereignisses
     */
    void verbindungHergestellt(DavVerbindungsEvent event);

    /**
     * es wurde eine Datenverteilterverbindung getrennt.
     *
     * @param event
     *           die Daten des Ereignisses
     */
    void verbindungGetrennt(DavVerbindungsEvent event);

    /**
     * eine Datenverteilterverbindung soll getrennt werden. Der Listener kann
     * hier ein Veto einlegen, sollte aber versuchen einen Zustand zu erreichen,
     * der beim einem der nächsten Aufrufe kein Veto mehr erfordert.
     *
     * @param event
     *           die Daten des Ereignisses
     * @return <code>true</code>, wenn die Datenverteilterverbindung offen
     *         gehalten werden soll, <code>false</code> sonst (kein Veto)
     */
    boolean verbindungHalten(DavVerbindungsEvent event);
}

```

---

## JOB\_FAMILY

---

ist die ID für die Job-Familie, deren Beendigung abgewertet wird, bevor die Rahmenwerk-Applikation tatsächlich beendet wird. Damit können Hintergrundprozesse gegebenenfalls noch anstehende Arbeiten beenden.

Um einen Job der Familie zuzuordnen muss die Funktion *belongsTo* implementiert werden, wie nachstehend beispielhaft dargestellt ist:



```
@Override
public boolean belongsTo(final Object family) {
    return Rahmenwerk.JOB_FAMILY.equals(family) || super.belongsTo(family);
}
```

Die Blockierung des Rahmenwerks ist nur bedingt sicher, ein “forced” -Abbruch ist jederzeit möglich!

## getXXX-Funktionen

die Funktionen liefern spezielle Informationen zur Rahmenwerk-Applikation:

- die übergebenen Kommandozeilen-Argumente
- den Name des angemeldeten Benutzers im Online-Betrieb
- das Systemobjekt des angemeldeten Benutzers im Online-Betrieb
- das bei der Herstellung der Datenverteilerverbindung eingegebene Passwort
- die Art der verwendeten Berechtigungsklassen
- das Objekt an dem die Parameter für die Bedienoberfläche hinterlegt sind (im Online-Betrieb)
- den Toolbar-Manager des Hauptfensters der Rahmenwerks-Applikation

## — Einstellungen —

Der Service bietet den Zugriff auf die allgemeinen und benutzerspezifischen Einstellungen des Rahmenwerks.

Der Service wird durch eine Instanz der Schnittstelle *Einstellungen* implementiert.

```
public interface Einstellungen {

    /**
     * liefert die für die übergebene Adresse vorliegende Einstellung.
     *
     * @param adresse
     *         die Adresse
     * @return das Einstellungsobjekt
     * @throws IOException
     *         die Einstellung konnte nicht gelesen werden
     */
    Object getValue(final EinstellungsAdresse adresse) throws IOException;

    /**
     * setzt für die übergebene Adresse die angegebene Einstellung.
     */
}
```

```

*
* @param adresse
*         die Adresse
* @param einstellung
*         das Einstellungsobjekt
* @throws IOException
*         die Einstellung konnte nicht gelesen werden
*/
void setValue(final EinstellungsAdresse adresse, final Object einstellung)
    throws IOException;

/**
 * setzt für die übergebene Adresse die angegebene Einstellung mit den
 * übergebenen Urlasserinformationen.
 *
 * @param adresse
 *         die Adresse
 * @param einstellung
 *         das Einstellungsobjekt
 * @param urlasser
 *         die Urlasserinformationen
 * @throws IOException
 *         die Einstellung konnte nicht gelesen werden
 */
void setValue(final EinstellungsAdresse adresse, final Object einstellung,
    final UrlasserInfo urlasser) throws IOException;

/**
 * entfernt die unter der übergebenen Adresse hinterlegten Einstellungen.
 *
 * @param adresse
 *         die Adresse
 */
void removeValue(EinstellungsAdresse adresse);

/**
 * fügt einen Listener hinzu, der über Änderungen von Einstellungen
 * informiert wird.
 *
 * @param listener
 *         der Listener
 */
void addEinstellungsListener(final EinstellungChangeListener listener);

/**
 * fügt einen Listener hinzu, der über Änderungen von Einstellungen der

```

```

    * angegebenen Kategorie informiert wird.
    *
    * @param listener
    *           der Listener
    * @param category
    *           die Kategorie
    */
    void addEinstellungsListener(final EinstellungChangeListener listener,
                                final String category);

    /**
     * entfernt einen Listener, der über Änderungen der Einstellungen informiert
     * wurde.
     *
     * @param listener
     *           der Listener
     */
    void removeEinstellungsListener(final EinstellungChangeListener listener);

    /**
     * entfernt einen Listener, der über Änderungen der Einstellungen in der
     * angegebenen Kategorie informiert wurde.
     *
     * @param listener
     *           der Listener
     * @param category
     *           die Kategorie
     */
    void removeEinstellungsListener(final EinstellungChangeListener listener,
                                    final String category);

    /**
     * fügt einen Listener hinzu, der über die Verfügbarkeit des
     * Einstellungsspeichers informiert wird.
     *
     * @param listener
     *           der Listener
     */
    void addEinstellungsAvailabilityListener(
        final EinstellungAvailabilityListener listener);

    /**
     * entfernt einen Listener, der über die Verfügbarkeit des
     * Einstellungsspeichers informiert wird.
     *
     * @param listener

```

```

    * der Listener
    */
    void removeEinstellungsAvailabilityListener(
        final EinstellungAvailabilityListener listener);
}

```

Eine detaillierte Beschreibung des Zugriffs auf die Einstellungen erfolgt in einem gesonderten Kapitel.

---

## — Berechtigungen —

---

Der Service bietet den Zugriff auf die von den Plug-ins des Rahmenwerks definierten Berechtigungen mit den in der Regel bestimmte Funktionalitäten auf Berechtigungsklassen bezogen verfügbar gemacht oder entzogen werden können.

Der Service wird durch eine Instanz der Schnittstelle *Berechtigungen* implementiert.

```

public interface Berechtigungen {

    /**
     * Fügt eine Funktion zur Liste der Funktionen mit Berechtigungen hinzu.
     *
     * @param funktion
     * Funktion, die hinzugefügt werden soll.
     */
    void addOberflaechenFunktion(final FunktionMitBerechtigung funktion);

    /**
     * Entfernt eine Funktion aus der Liste der Funktionen mit Berechtigungen.
     *
     * @param id
     * ID der Funktion, die entfernt werden soll.
     */
    void removeOberflaechenFunktion(final String id);

    /**
     * liefert die registrierten Funktionen.
     *
     * @return die Menge der Funktionen
     */
    Collection<FunktionMitBerechtigung> getFunktionen();

    /**
     * prüft, ob der als SystemObject übergebene Benutzer, die Berechtigung für

```

```
* die angegebene Funktion besitzt.  
*  
* @param benutzer  
*           der Benutzer  
* @param funktion  
*           die Funktion  
* @return der Zustand der Berechtigung  
*/
```

```
boolean hasBerechtigung(SystemObject benutzer,  
                        FunktionMitBerechtigung funktion);
```

```
/**  
* prüft, ob der Benutzer mit der übergebenen PID, die Berechtigung für die  
* angegebene Funktion besitzt.  
*  
* @param benutzerPid  
*           die PID  
* @param funktion  
*           die Funktion  
* @return der Zustand der Berechtigung  
*/
```

```
boolean hasBerechtigung(String benutzerPid, FunktionMitBerechtigung funktion);
```

```
/**  
* liefert die Funktion mit der angegebenen Id oder <code>null</code>, wenn  
* keine solche existiert.  
*  
* @param id  
*           die ID  
* @return die Funktion oder <code>null</code>  
*/
```

```
FunktionMitBerechtigung getFunktion(String id);
```

```
/**  
* liefert eine Sammlung der verfügbaren Berechtigungsklassen (als  
* SystemObject).  
*  
* @return die Berechtigungsklassen  
*/
```

```
Collection<SystemObject> getBerechtigungsklassen();
```

```
/**  
* fügt einen Listener hinzu, der über Änderungen von Berechtigungen  
* informiert wird.  
*  
* @param listener
```

```

*           der Listener
*/
void addOberflaechenFunktionsListener(IBerechtigungListener listener);

/**
 * fügt einen Listener hinzu, der über Änderungen von Berechtigungen der
 * angegebenen Funktion informiert wird.
 *
 * @param listener
 *           der Listener
 * @param funktion
 *           die Funktion
 */
void addListenerForFunktion(IBerechtigungListener listener,
                             FunktionMitBerechtigung funktion);

/**
 * entfernt einen für Änderungen von Berechtigungen angemeldeten Listener.
 *
 * @param listener
 *           der Listener
 */
void removeListener(IBerechtigungListener listener);

```

Eine detaillierte Beschreibung des Zugriffs auf die Berechtigungen erfolgt in einem gesonderten Kapitel.

# Einstellungen

---

## — Hintergrund —

---

Das Rahmenwerk bietet die Möglichkeit, Einstellungen die allgemein oder nutzerspezifisch gelten zu speichern und wieder abzurufen.

Einstellungen können entweder lokal hinterlegt oder netzwerkweit abgespeichert werden. Zum netzwerkweiten Speichern von Einstellungen werden die Daten in den bisher verwendeten Datenverteiler-Attributgruppen “atg.benutzerEinstellungenOberflächeNetzweit” bzw. “atg.globaleEinstellungenOberflächeNetzweit” abgelegt. Die Attributgruppen sind einer Instanz vom Typ “Oberfläche” zugeordnet, der auch die durch den Typ “Autarke Organisationseinheit” erweitert wird.

Potentiell vorgesehen ist die Speicherung von Informationen für folgende Gruppierungen:

- allgemeine systemweite Einstellungen
- allgemeine lokale Einstellungen
- benutzerklassenspezifische systemweite Einstellungen
- benutzerklassenspezifische lokale Einstellungen
- benutzerspezifische systemweite Einstellungen
- benutzerspezifische lokale Einstellungen

Eine Reihenfolge der Überlagerung bzw. der Kombinationen der oben genannten Gruppierungen für eine bestimmte Einstellung ist nicht vorgesehen. Diese Entscheidung wird durch die jeweilige Anwendung getroffen und kann damit für jede Einstellung individuell anders sein.

Die aktuelle Implementierung unterstützt bisher noch nicht die Speicherung der Einstellungen für Benutzerklassen, dazu sollte zunächst die Zuordnung der Einstellungen für Benutzerklassen bzw. Benutzer an die Objekte diesen Typs verschoben werden. Ein entsprechender Änderungsantrag an die NERZ wurde vorgenommen.

Der Zugriff auf Einstellungen sollte nicht direkt über die Daten aus den Attributgruppen erfolgen, stattdessen steht ein Service zur Verfügung, mit dem die erforderlichen Informationen über das Rahmenwerk gelesen bzw. geschrieben werden können.

---

## — Der Service “Einstellungen” —

---

---

### Allgemeine Funktionalität

---

Der Service wird beim Start des Rahmenwerk initialisiert und steht damit unmittelbar den Plug-ins, die ihn verwenden wollen zur Verfügung. Der Zugriff auf die Services des Rahmenwerks ist im Kapitel

“Rahmenwerk-Services” detailliert beschrieben.

Der Service selbst wird durch folgende Schnittstelle definiert:

```
public interface Einstellungen {

    Object getValue(final EinstellungsAdresse adresse) throws IOException;

    void setValue(final EinstellungsAdresse adresse,
                final Object einstellung) throws IOException;
    void setValue(final EinstellungsAdresse adresse,
                final Object einstellung,
                final UrlasserInfo urlasser) throws IOException;

    void removeValue(EinstellungsAdresse adresse);

    void addEinstellungenListener(final EinstellungChangeListener listener);
    void addEinstellungenListener(final EinstellungChangeListener listener,
                                final String category);
    void removeEinstellungenListener(final EinstellungChangeListener listener);
    void removeEinstellungenListener(final EinstellungChangeListener listener,
                                    final String category);

    void addEinstellungenAvailabilityListener(
                                final EinstellungAvailabilityListener listener);
    void removeEinstellungenAvailabilityListener(
                                final EinstellungAvailabilityListener listener);

}
```

## getValue - Lesen einer definierten Einstellung

Die Funktion liefert die mit der angegebenen Einstellungsadresse definierte Einstellung.

Wenn keine entsprechende Einstellung vorliegt, wird der Wert *null* geliefert. Es erfolgt keine “Vererbung” der Einstellungen, d.h. wenn nach einer benutzerdefinierten Einstellung gefragt wird, wird nicht automatisch die allgemeine Einstellung für die angegebene ID geliefert, wenn keine benutzerspezifische existiert.

## setValue - Setzen einer definierten Einstellung

Die Funktion schreibt den übergebenen Einstellungswert in den Einstellungsspeicher.

Die Standardimplementierung des Rahmenwerks sieht lediglich Strings als Einstellungsobjekte vor. Sollen Objekte eines anderen Typs als Einstellungen gespeichert werden muss eine entsprechende Factory als Service im Rahmenwerk registriert werden.



## removeValue - Entfernen einer Einstellung

Die Funktion entfernt die mit der übergebenen Adresse spezifizierte Einstellung unwiederbringlich vom Einstellungsspeicher.

## add/removeEinstellungsListener - Listener für Änderungen von Einstellungen

Die Funktionen ermöglichen die Anmeldung von Listnern, die benachrichtigt werden, wenn sich Einstellungen ändern. Optional kann die Anmeldung auf spezielle Typen erfolgen, das ist aber erst dann sinnvoll, wenn Plug-ins auch Factories zum Speichern von Einstellungen, die keine Strings sind implementieren.

Ein Listener für Einstellungen implementiert folgende Schnittstelle:

```
public interface EinstellungChangeListener {

    /**
     * es wurde eine neue Einstellung angelegt.
     *
     * @param event
     *         die Daten des Ereignisses
     */
    void einstellungAngelegt(final EinstellungsEvent event);

    /**
     * eine bestehende Einstellung wurde aktualisiert.
     *
     * @param event
     *         die Daten des Ereignisses
     */
    void einstellungAktualisiert(final EinstellungsEvent event);

    /**
     * eine Einstellung wurde aus dem Einstellungsspeicher entfernt.
     *
     * @param event
     *         die Daten des Ereignisses
     */
    void einstellungEntfernt(final EinstellungsEvent event);
}
```

und wird damit benachrichtigt über:

- neue Einstellungen
- geänderte Einstellungen und
- entfernte Einstellungen

# add/removeEinstellungsAvailabilityListener - Listener für die Verfügbarkeit des Einstellungs-Services

Die Funktion erlaubt die Registrierung von Listnern die über die Verfügbarkeit der netzwerkweiten Einstellungsspeicher informieren.

```
public interface EinstellungAvailabilityListener {  
  
    /** der Einstellungsspeicher ist verfügbar. */  
    void available();  
  
    /** der Einstellungsspeicher ist nicht verfügbar. */  
    void disabled();  
}
```

Alle oben aufgeführten Einstellungsspeicher sind nur verfügbar, wenn eine Datenverteilterverbindung besteht. Im Offline-Betrieb ist nur der allgemeine lokale Einstellungsspeicher verfügbar, da eine Zuordnung zu Benutzerklassen bzw. Benutzern auf Grund der fehlenden Authentifizierung durch die Datenverteilterkonfiguration nicht möglich ist.

---

## Die Einstellungsadresse

---

Die Einstellungsadresse spezifiziert, auf welche Einstellung zugegriffen werden soll.

Momentan wird eine Einstellungsadresse über den Konstruktor:

```
public EinstellungsAdresse(final String typ, final String id,  
    final EinstellungOwnerType ownerType, final String pid,  
    final EinstellungLocation location)
```

erzeugt. Zusätzliche Convenience-Funktionen sind geplant.

Die Parameter des Konstruktors haben folgende Bedeutung:

### typ

der Objekttyp mit dem Einstellungen abgespeichert werden ist prinzipiell freigestellt und wird mit dem Parameter "typ" übergeben. Standardmäßig werden alle Parameter als "String" abgelegt. Wenn für den Parameter "typ" der Wert *null* übergeben wird, wird der Parametertyp auf "java.lang.String" gesetzt.

Für jeden Typ muss eine entsprechende Factory als OSGI-Service registriert werden, die die Serialisierung und Deserialisierung des Einstellungsobjekts implementiert. Mit dem Rahmenwerk mitgeliefert wird die Factory für den Typ "java.lang.String"!

## id

die ID unter der die Einstellung innerhalb des Einstellungsspeichers hinterlegt ist. Die Wahl der ID ist nicht weiter festgelegt. Es sollte jedoch beachtet werden, dass die Gültigkeit für das gesamte System besteht und dass eine möglichst eindeutige ID gewählt werden sollte bspw. unter Einbeziehung der ID des Plug-ins von dem eine Einstellung verwendet wird.

## ownerType

beschreibt, für wen die Einstellung gültig ist. Mit dem enum stehen folgende Werte zur Verfügung:

```
public enum EinstellungOwnerType {  
    /** Systemweite (allgemeine) Einstellung. */  
    SYSTEM,  
    /** Einstellung ist einer Benutzerklasse zugeordnet. */  
    BENUTZERKLASSE,  
    /** Einstellung ist einem Benutzer zugeordnet. */  
    BENUTZER;  
}
```

Zu beachten ist, dass die Einstellungen für Benutzerklassen (noch) nicht unterstützt werden.

## pid

ist die PID des Benutzers oder der Benutzerklasse für den die Einstellung gültig ist. für allgemeine Einstellungen ist der Wert *null* zu übergeben.

## location

ist der Ort, an dem die Einstellung gespeichert werden soll. Mit dem enum stehen folgende Werte zur Verfügung:

```
public enum EinstellungLocation {  
    /** Einstellung wird netzwerkweit (als Parameter in Datenverteiler) gespeichert. */  
    NETZWERKWEIT,  
    /** Einstellung wird lokal gespeichert. */  
    LOKAL;  
}
```

---

## — Erweiterte Funktionalität - EinstellungsFactory —

---

Als Einstellungen werden in der Standardimplementierung lediglich String-Objekte unterstützt.

Wenn Objekte anderer Typen direkt im Einstellungsspeicher abgelegt werden sollen, muss in der Einstellungsadresse der zugeordnete Typ angegeben werden (das kann der Klassenname oder jede andere beliebige ID sein) und es muss eine `EinstellungsFactory` als Service registriert werden, mit der das entsprechende Einstellungsobjekt serialisiert und deserialisiert werden kann.

Das Interface für die Factory wird wie folgt beschrieben:

```
public interface EinstellungsFactory {

    /**
     * liefert den Typ der Einstellung (i.d.R. der Klassenname der zu
     * erzeugenden Instanzen)
     *
     * @return den Name
     */
    String getTyp();

    /**
     * wandelt ein Objekt in eine String-Repräsentation um.
     *
     * @param einstellung
     *           das zu serialisierende Einstellungsobjekt
     * @return die Stringrepräsentation
     * @throws IOException
     *           die Serialisierung konnte nicht erfolgreich ausgeführt werden
     */
    String serialisiere(final Object einstellung) throws IOException;

    /**
     * wandelt einen String in das gewünschte Einstellungsobjekt um.
     *
     * @param daten
     *           die Daten des Zielobjekts als String
     * @return das erzeugte Objekt
     * @throws IOException
     *           das Objekt konnte aus dem übergebenen String nicht erzeugt
     *           werden
     */
    Object deserialisiere(final String daten) throws IOException;
}
```

Die mit der Funktion `getTyp` gelieferte ID muss dann beim Lesen und Schreiben der Einstellung dem in der *Einstellungsadresse* angegebenen Typ entsprechen.

# Berechtigungen

---

## — Hintergrund —

---

Berechtigungen werden im Rahmenwerk als abstrakte Definitionen betrachtet, denen bezogen auf eine gegebene Benutzerklasse das Attribut Freigabe oder Sperrung zugeordnet werden kann.

Die Funktionen mit Berechtigungen sind insofern abstrakt, dass sie nicht unbedingt Funktionen in Sinne von konkreten Aktionen, sondern alternativ auch andere Eigenschaften wie Sichtbarkeit, Bedienbarkeit, Filterung bestimmter Eigenschaften usw. definieren können.

Eine Oberflächenfunktion definiert sich im Rahmenwerk 2.0 durch folgende Eigenschaften: - ID der Funktion - Name der Funktion - Beschreibung der Funktion - Kategorie

Oberflächenfunktionen können dynamisch durch die zur Verfügung stehenden API-Funktionen in die Berechtigungsverwaltung des Rahmenwerks integriert werden. Auf welche Art und Weise die Funktionen konkret definiert werden ist durch das Rahmenwerk nicht festgelegt. Es kann sich gleichermaßen um Funktionsdefinitionen im Sourcecode eines entsprechenden Plug-Ins handeln, wie um Definitionen über Extension-Points oder externe Konfigurationsdateien.

**Zu bevorzugen ist jedoch die Definition von ExtensionPoints!**

Die Kategorie der Oberflächenfunktionen dient lediglich der Verbesserung der Möglichkeiten zur Repräsentation der verfügbaren Definitionen im konkreten Datenverteilersystem.

Für jede Oberflächenfunktion können Berechtigungen pro definierter Benutzerklasse vergeben werden. Damit werden die in den Technischen Anforderungen für das Segment BuV geforderten Eigenschaften (TBuV-103/104) wie folgt realisiert:

- **Wer?** - entspricht der festgelegten Benutzerklasse (eine Rechtevergabe auf Benutzerebene ist nicht vorgesehen und wird nur implizit durch die Zuordnung eines Nutzers zu einer Benutzerklasse realisiert)
- **Was?** - entspricht der Oberflächenfunktion an sich, definiert über die festgelegte ID
- **Wie?** - entspricht der Festlegung Freigabe/Sperrung
- **Worauf?** - wird nicht direkt abgebildet sondern bei Bedarf durch mehrere Oberflächenfunktionen umgesetzt, d.h. statt der Definition einer einzelnen Funktion mit Beschränkungen auf jeweils ein Bedienelement eines Plug-Ins definiert das Plug-In zwei Funktionen, die individuell mit Rechte versehen werden können.
- **Ausnahmen?** - auf Ausnahmen im eigentlichen Sinne wird verzichtet, die Zuordnungen Benutzerklasse-Funktion sind eindeutig, die Parametrierung erfolgt über geeignete Editoren, so dass auf Platzhalter in den Definitionen der Rechte verzichtet werden kann und Ausnahmen damit obsolet werden

---

## — Repräsentation im Datenverteiler —

---

Die Oberflächenberechtigungen werden im vorhandenen Parameterdatensatz eines Objekts "Oberfläche" abgelegt. In der Regel ist das die AOE, dem Rahmenwerk kann aber ein anderes Objekt zur Berechtigungsverwaltung zugewiesen werden.

Die entsprechenden Objekte und Attributgruppen sind in der aktuellen Datenverteilerkonfiguration verfügbar. Die Einträge des Parameterdatensatzes bezüglich einschränkender Objekte, Ausnahmen und Verschachtelungstiefe von Ausnahmen werden ignoriert.

Der Zugriff auf die Berechtigungen erfolgt über einen vom Rahmenwerk bereitgestelltem Service vom Typ "Berechtigungen".

---

## — Programmierschnittstelle —

---

### Service "Berechtigungen"

---

Die Schnittstelle Berechtigungen liegt in der folgenden Form vor:

```
public interface Berechtigungen {

    void addOberflaechenFunktion(final FunktionMitBerechtigung funktion);
    void removeOberflaechenFunktion(final String id);

    Collection<FunktionMitBerechtigung> getFunktionen();
    FunktionMitBerechtigung getFunktion(String id);

    boolean hasBerechtigung(SystemObject benutzer,
        FunktionMitBerechtigung funktion);
    boolean hasBerechtigung(String benutzerPid, FunktionMitBerechtigung funktion);

    Collection<SystemObject> getBerechtigungsklassen();

    void addOberflaechenFunktionsListener(IBerechtigungListener listener);
    void addListenerForFunktion(IBerechtigungListener listener,
        FunktionMitBerechtigung funktion);
    void removeListener(IBerechtigungListener listener);
}
```

Folgende Funktionsgruppen werden zur Verfügung gestellt.

#### add/removeOberflaechenFunktion

Die Funktionen dienen dazu Berechtigungsfunktionen zur internen Berechtigungsverwaltung des Rahmenwerks hinzuzufügen bzw. zu entfernen.

Diese Funktionalität besteht im Wesentlichen aus Kompatibilitätsgründen. Zu bevorzugen ist die Definition von ExtensionPoints durch die Plug-ins, die Berechtigungsfunktionen in die Rahmenwerk-Applikation einbringen wollen.

## getFunktion/en

liefert alle bzw. durch die ID festgelegte Funktionen aus der Berechtigungsverwaltung des Rahmenwerks.

## hasBerechtigung

prüft, ob die Berechtigung für eine engegebene Berechtigungsfunktion hat.

Für die Prüfung der Berechtigungen wird geprüft, ob eine der Berechtigungsklassen, der der Nutzer zugeordnet ist eine Freigabe für die übergebene Funktion besitzt.

**Die Zuordnung der Nutzer zu Berechtigungsklassen hängt davon ab, ob das neue oder alte Berechtigungskonzept verwendet wird.**

## Listener-Funktionen

melden Listener an/ab, die über Änderungen in der Berechtigungsverwaltung allgemein oder für bestimmte Berechtigungsfunktionen benachrichtigt werden sollen.

---

## FunktionMitBerechtigung

---

Instanzen dieser Klasse repräsentieren die in der Berechtigungsverwaltung des Rahmenwerks registrierten Funktionen, denen eine Berechtigung/Freigabe pro Berechtigungsklasse zugeordnet wird.

Die Instanzen können zwar aus Kompatibilitätsgründen direkt angelegt werden, sollten aber normalerweise als ExtensionPoints in Plug-ins definiert und vom Rahmenwerk intern initialisiert werden.

```
public FunktionMitBerechtigung(final String pluginId,
                                final String kategorie, final String id, final String bezeichnung,
                                final String beschreibung) {
```

Eine Berechtigungsfunktion ist durch die im Konstruktor übergebenen Attribute definiert:

- **pluginId** die ID des Plug-ins in der die Funktion angelegt wird
- **kategorie** eine optionale Kategorie, die nur zu Anzeigezwecken verwendet wird
- **id** die eindeutige ID der Funktion
- **bezeichnung** der Name der Funktion
- **beschreibung** ein optionaler Beschreibungstext für die Funktion

---

## IBerechtigungsListener

---

Die Schnittstelle für einen Listener, der über Änderungen in Berechtigungsverwaltung informiert wird.

```
public interface IBerechtigungListener {  
    void sperrung(BerechtigungEreignis e);  
    void freigabe(BerechtigungEreignis e);  
}
```

Das BerechtigungEreignis enthält die Informationen, welche Funktionen gesperrt bzw. freigegeben wurden.

```
public class BerechtigungEreignis extends EventObject {  
  
    ....  
  
    /**  
     * ermittelt, ob eine Freigabe gemeldet wird.  
     *  
     * @return der Zustand  
     */  
    public boolean isFreigabe() {  
        return freigabe;  
    }  
  
    /**  
     * liefert die betroffene Berechtigungsklasse.  
     *  
     * @return die Berechtigungsklasse  
     */  
    public SystemObject getBerechtigungsKlasse() {  
        return berechtigungsKlasse;  
    }  
  
    /**  
     * liefert die betroffenen Funktionen.  
     *  
     * @return eine Liste der Funktionen  
     */  
    public List<FunktionMitBerechtigung> getFunktionen() {  
        return Collections.unmodifiableList(funktionen);  
    }  
}
```



---

# — Definition einer Berechtigungsfunktion mittels ExtensionPoint —

---

Die Berechtigungsfunktionen werden innerhalb des Plug-ins, das eine solche Funktion bereitstellen möchte durch einen ExtensionPoint vom Typ “de.bsvrz.buv.rw.rw.funktionmitberechtigung” repräsentiert.

Eine Extension “function” wird durch die Attribute:

- **id** die eindeutige ID der Funktion
- **bezeichnung** die Bezeichnung der Funktion
- **kategorie** die optionale Kategorie
- **beschreibung** die optionale Beschreibung

definiert.

Eine auf diese Weise angelegte Berechtigungsfunktion wird vom Rahmenwerk automatisch initialisiert und der Berechtigungsverwaltung zugeordnet.

# Anmeldung beim Datenverteiler

---

## — Hintergrund —

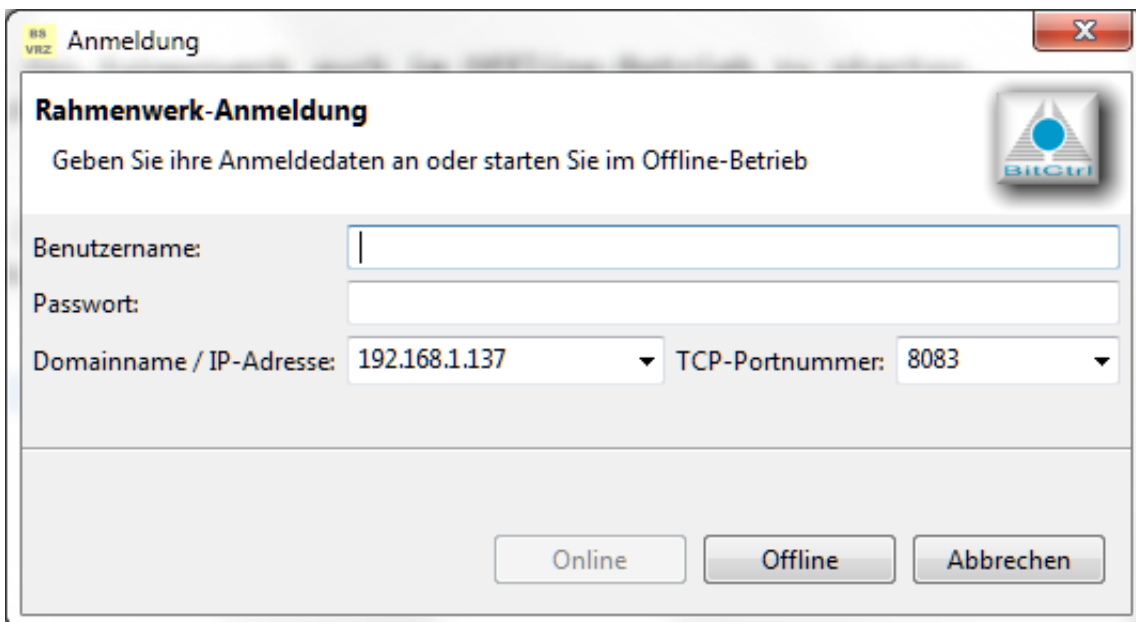
---

Eine wesentliche Aufgabe des Rahmenwerks besteht in der Bereitstellung der Verbindung zu einem Datenverteilersystem.

Die Herstellung der Verbindung erfolgt durch einen Login-Dialog, der beim Start der Rahmenwerksapplikation angezeigt wird und die Möglichkeit bietet, sich mit entsprechend gültigen Anmeldedaten mit einem System zu verbinden.

Optional besteht die Möglichkeit das Rahmenwerk auch im Offline-Betrieb zu starten. Dabei steht für die Plug-ins in der Regel nur eine eingeschränkte Funktionalität zur Verfügung.

Aus dem Offline-Betrieb kann die Datenverteilerverbindung durch Ausführen des Kommandos "Anmelden" hergestellt werden. Es öffnet sich der gleiche Login-Dialog wie beim Starten des Rahmenwerks.



*Login-Dialog*

---

## — Konfiguration des Dialogs —

---

Der Dialog kann in gewissen Grenzen an die Erfordernisse der konkreten Anwendung angepasst werden. Die Anpassung erfolgt über Kommandozeilen-Parameter, die den Rahmenwerk beim Start mitgegeben werden.

Folgende Optionen sind vorgesehen:

```
-loginLogo=<Pfad zu einer Imagedatei>
```

Die Option bestimmt das Logo, dass im Logindialog angezeigt wird. Das Image wird ohne Rahmen eingepasst, d.h. dieser sollte Bestandteil der verwendeten Grafik sein!

```
-loginDialogTitle=<Text für den Dialogtitel>
```

Die Option bestimmt den Titel des Dialogfensters. Der Standwert ist "Anmeldung".

```
-loginTitel=<Text für den Titel>
```

Die Option bestimmt den Titel im Kopfbereich des Dialogs. Der Standwert ist "Rahmenwerk-Anmeldung".

```
-loginMessage=<Text für den Meldungstext>
```

Die Option bestimmt den Text im Kopfbereich des Dialogs. Der Standwert ist "Geben Sie ihre Anmeldedaten an oder starten Sie im Offline-Betrieb"

```
-disableHost
```

Mit dieser Option wird die Anzeige der Eingabefelder für den Host und den Port für die Datenverteilterverbindung unterdrückt. Stattdessen werden die Angaben aus dem Standard-Datenverteiler-Startargument *-datenverteiler* übernommen. Falls dieses fehlt, werden die Standardwerte *localhost* und *8083* verwendet. Die Option bestimmt den Text im Kopfbereich des Dialogs. Der Standwert ist "Geben Sie ihre Anmeldedaten an oder starten Sie im Offline-Betrieb"

Der Dialog wird immer an der Position geöffnet, an der er zuletzt geschlossen wurde. Insbesondere auf einem Multi-Monitor-System muss daher die korrekte Position einmalig durch eine Anmeldung oder einen Start des Rahmenwerks im Offline-Betrieb festgelegt werden.

---

## — Verwendung der Datenverteilterverbindung —

---

Die durch die Anmeldung hergestellte Datenverteilterverbindung wird allen Komponenten des Rahmenwerks über den Service vom Typ *Rahmenwerk* zur Verfügung gestellt.

Der Service bietet auch die Möglichkeit, Änderungen des Verbindungsstatus, d.h. Abmeldungen und erneute Anmeldungen per Listener zu beobachten.

Detaillierte Informationen dazu finden sich im Kapitel "Services".

# Urlasserdialog

---

## — Hintergrund —

---

Der Urlasserdialog dient dazu ausgewählte Aktionen mit Anmeldedaten eines Nutzers zu verifizieren. Der Nutzer der über den Urlasserdialog authentifiziert wird, muss nicht der aktuell angemeldete Nutzer sein. Gegebenenfalls wird eine neue Datenverteilterverbindung für den abweichenden Nutzer erzeugt, über die dann die mit dem Urlasserdialog verknüpften Operationen ausgeführt werden, damit das Berechtigungskonzept des Datenvertailers an dieser Stelle greifen kann.

---

## — Programmierschnittstelle —

---

### Konstruktor

---

Der Urlasserdialog steht als JFace-Dialog-Klasse in der Basis-Bibliothek des Rahmenwerks zur Verfügung.

```
java public UrlasserDatenDialog(final Shell parent, final UrlasserDatenSender datenSender) {  
super(parent); this.datenSender = datenSender; }
```

Der Dialog wird mit der *open*-Funktion eines Dialogs geöffnet und liefert die gängigen Werte *Window.OK* oder *Window.CANCEL* zurück, je nachdem wie der Dialog geschlossen wurde.

Die eingegebenen Urlasserinformationen können anschließend bei Bedarf mit der Funktion

```
java public UrlasserInfo getUrlasserInfo();
```

bestimmt werden. Die potentiell erzeugte neue Urlasserverbindung steht nach dem Abschluss des Dialogs nicht mehr zur Verfügung. Alle Operationen, die über diese Verbindung ausgeführt werden sollen, müssen von der im Konstruktor übergebenen Instanz des *UrlasserDatenSender* übernommen werden.

---

## UrlasserDatenSender

---

Eine Instanz dieser Klasse übernimmt die mit einem Urlasserdialog potentiell erzeugte oder die aktuelle Datenverteilterverbindung und führt die mit dem Urlasserdialog verbundenen Operationen in Bezug zum Datenverteiler aus. Die Ausführung der Operationen blockiert den Dialog, d.h. er wird erst nach Abschluß aller Operationen des Datensenders geschlossen und die Verbindung wird abgebaut.

Die übergebene Klasse muss eine Instanz der folgende Schnittstelle sein:

```
""java public interface UrlasserDatenSender {
```

```

/**
 * Callback-Funktion wird aufgerufen, wenn ein Urlasser über den
 * Urlasserdialog identifiziert wurde und optional eine entsprechende
 * Datenverteilterverbindung erstellt wurde.
 *
 * Nach der Ausführung der gewünschten Operationen wird die Verbindung
 * automatisch wieder geschlossen.
 *
 * @param verbindung
 *           die Verbindung zum Datenverteiler
 * @param urlasser
 *           die Urlasserinformationen
 */
void execute(ClientDavInterface verbindung, UrlasserInfo urlasser);

```

```

}

```

---

## DefaultDatenSender

---

Das Rahmenwerk stellt eine Standard-Implementierung des *UrlasserDatenSender* zur Verfügung, mit dem das bisherige Verhalten des Urlasserdialogs (versenden einer Menge von *ResultData* - Instanzen) nachgebildet wurde.

```
java public DefaultDatenSender(final Collection<ResultData> resultDatas);
```

Der Konstruktor übernimmt die Daten, die bei erfolgreicher Authentifizierung über den Urlasserdialog an den Datenverteiler versendet werden.

# Menüs und Toolbars

---

## — Hintergrund —

---

Das Hauptmenü und die Toolbars (Haupt-Toolbar und Statuszeile) sind innerhalb des Rahmenwerks editierbar und können damit für die individuellen Erfordernisse einer konkreten Datenverteiler-Anwendung angepasst werden.

Menüs und Toolbars werden generell gleich behandelt, deshalb steht für die Erstellung der gleiche Editor zur Verfügung, lediglich das Ziel der angezeigten Funktionselemente unterscheidet sich. Es stehen auch die gleichen Ressourcen zur Verfügung. Praktisch sind aber nicht alle Elemente sinnvoll gleichermaßen in Menü, Toolbar und Symbolleiste verwendbar. Eine softwareseitige Unterstützung wird hierbei noch nicht geboten, es wird auf die Intuition und die Intelligenz des Menübearbeiters gesetzt.

Die erstellten Strukturen für Menüs (im weiteren Sinne) werden als Einstellungen im Datenverteiler hinterlegt. Damit steht die Möglichkeit zur Verfügung ein Menü individuell für einen bestimmten Nutzer oder individuell für einen lokalen Arbeitsplatz zu definieren.

Aktiviert wird das in der folgenden Suchreihenfolge zuersetz gefundene Menü:

- lokales nutzerspezifisches Menü
- netzwerkweites nutzerspezifisches Menü
- netzwerkweites allgemeines Menü
- lokales allgemeines Menü

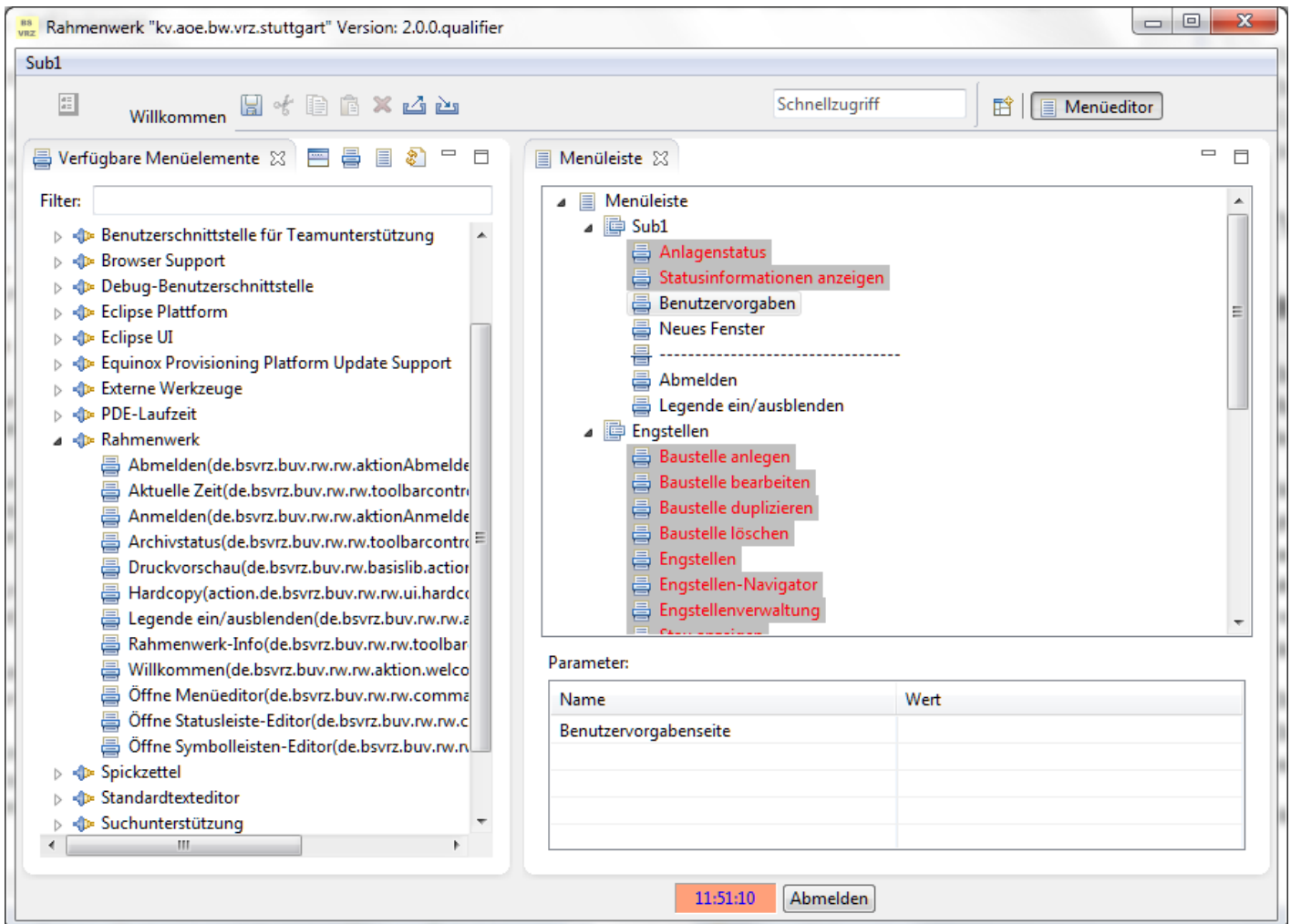
Da im Offline-Betrieb nur der lokale allgemeine Einstellungsspeicher zur Verfügung steht, wird hier immer diese Menüdefinition verwendet!

---

## — Menüeditor —

---

Der Menüeditor steht als Perspektive zur Verfügung, die aus einer Ansicht mit den für Menüs verfügbaren Elementen und dem Bereich für Editoren besteht.



*Menü-Editor-Perspektive*

Ein Editor wird über die jeweilige Schaltfläche (Menü, Toolbar, Statusleiste) in der Ansicht der verfügbaren Elemente geöffnet. Dabei erfolgt die Abfrage welches Menü aus den Einstellungen geladen werden soll.

Wird beim Betätigen der Schaltflächen zum Öffnen eines Editors gleichzeitig die CTRL-Taste betätigt, wird ein leerer Editor geöffnet.

Die gewünschten Elemente können wie gewohnt per Drag & Drop in das Menü eingefügt und dort angeordnet werden. Untermenüs werden per Kontextmenü-Funktion im Editor erzeugt.

Eine im Menüeditor befindliche Struktur kann per Kontextmenü sofort aktiviert werden. Ansonsten erfolgt die Aktivierung automatisch beim Speichern des Menüs (nach den oben genannten Kriterien).

## — Ressourcen für Menüelemente —

### Allgemein

Alle verfügbaren Menü-Elemente werden von Plug-ins über ExtensionPoints zur Verfügung gestellt.

Mögliche Quellen sind:

- Actions aus ExtensionPoints vom Typ *ActionSet* (sollten nicht mehr verwendet werden, da dieser ExtensionPoint seit Eclipse Version 2.1 auf deprecated gesetzt wurde)
- ExtensionPoints vom Typ *Command*
- ExtensionPoints vom Typ *ToolBarControl*

---

## ToolBarControl

---

sind sind spezielle Contribution-Items, die weder durch Commands noch Actions repräsentiert werden können, beispielsweise Zeitanzeige, Statusanzeigen für Elemente des Systems, Informationsanzeigen, ....

Sie müssen die Schnittstelle:

```
``java public interface RwMenuControl extends IContributionItem {
```



```

/**
 * übergibt der Instanz die im Menüeditor eingestellten Parameter für das
 * Element.
 *
 * @param parameter
 *         die Parameter als Zuordnungsliste (ID, Wert)
 */
void setParameter(Map<String, String> parameter);

/**
 * ermittelt, ob das Element im Hauptmenü des Rahmenwerks verwendet werden
 * kann.
 *
 * @return den Status
 */
boolean useInMenue();

/**
 * ermittelt, ob das Element in der Symbolleiste des Rahmenwerks verwendet
 * werden kann.
 *
 * @return den Status
 */
boolean useInToolbar();

/**
 * ermittelt, ob das Element in der Statusleiste des Rahmenwerks verwendet
 * werden kann.
 *
 * @return den Status
 */
boolean useInStatusLine();

}

```

implementieren und als *control* in einem ExtensionPoint vom Typ *de.bsvrz.buv.rw.rw.toolbarcontrol* in das System integriert werden.

# Startbildschirm

---

## — Hintergrund —

---

Der Startbildschirm des Rahmenwerks wird durch eine Intro-Seite ersetzt, die die bisherige App-Perspektive ersetzt.

Eine Standardperspektive wird beim Neustart des Rahmenwerks nicht gesetzt.

Die Introseite repräsentiert lediglich ein Browserwidget, das mit beliebigem Inhalt gefüllt werden kann.

Standardmäßig wird die Seite nur einmalig beim Start angezeigt, sofern das nicht anders festgelegt wurde, oder wenn beim Starten der Workbench keine Perspektive aktiviert wurde.

---

## — Konfiguration der Startansicht —

---

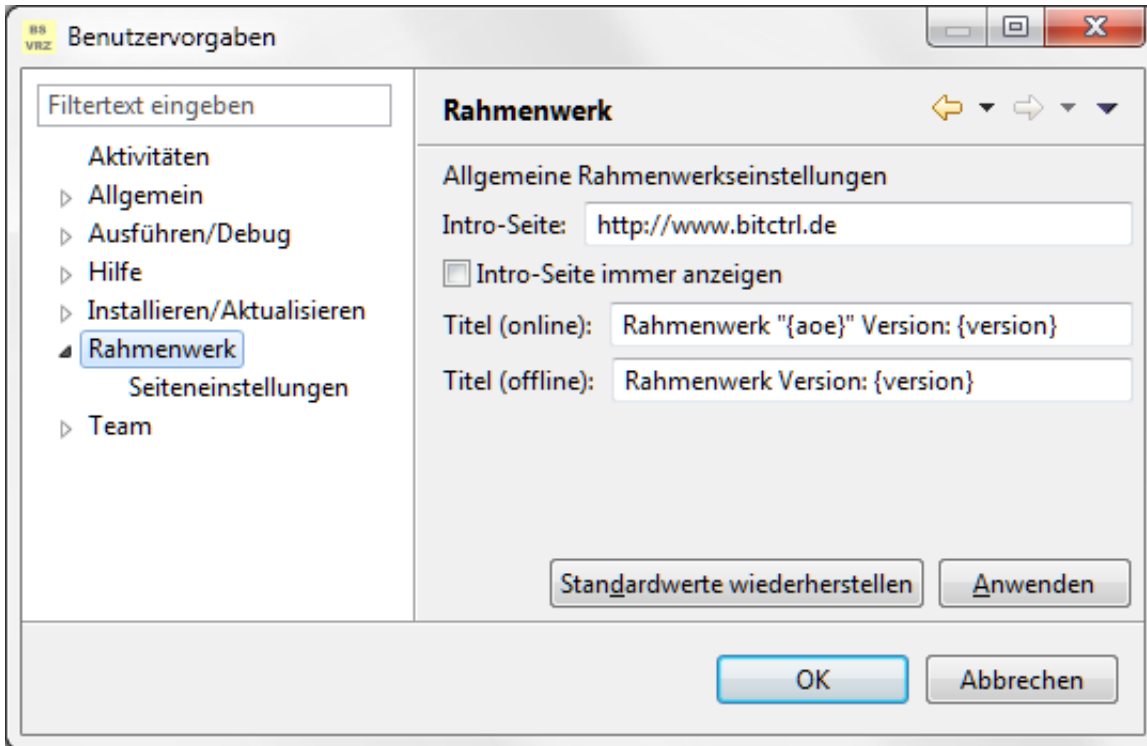
Der Inhalt der Startansicht kann über einen Kommandozeilenparameter oder über die lokale Einstellungen des Rahmenwerks vorgenommen nachdem dieses aktiviert wurde.

Als Startoption steht der Parameter:

```
bash -introUrl=<URL des anzuzeigenden Inhalts>
```

zur Verfügung. Dieser hat Vorrang gegenüber anderen Einstellungen.

Innerhalb des Rahmenwerks kann über lokale Benutzereinstellungen das Verhalten des Startbildschirms definiert werden.



*Grundeinstellungen des Rahmenwerks*

Einstellbar sind: - die anzuzeigende URL innerhalb der Willkommenseite - ob die Willkommenseite bei jedem Start des Rahmenwerks angezeigt werden soll, womit beispielsweise aktuelle Nachrichten über einen projektinternen Webserver publiziert werden könnten.

# Standardinformationen

---

## — Hintergrund —

---

Das Rahmenwerk bietet Funktionen, um statische Standardinformationen zu präsentieren. Diese werden beispielsweise im Titel des Workbenchfensters eingeblendet und können über entsprechende Controls in der Statuszeile des Hauptfensters eingeblendet werden.

Die Ausgabe der Informationen erfolgt über einen Formatstring, in dem die gewünschten statischen Informationen per Platzhalter eingebaut sind. Die Platzhalter werden durch vordefinierte Kürzel, die in geschweifte Klammern eingebettet sind gebildet.

Folgende Platzhalter stehen zur Verfügung:

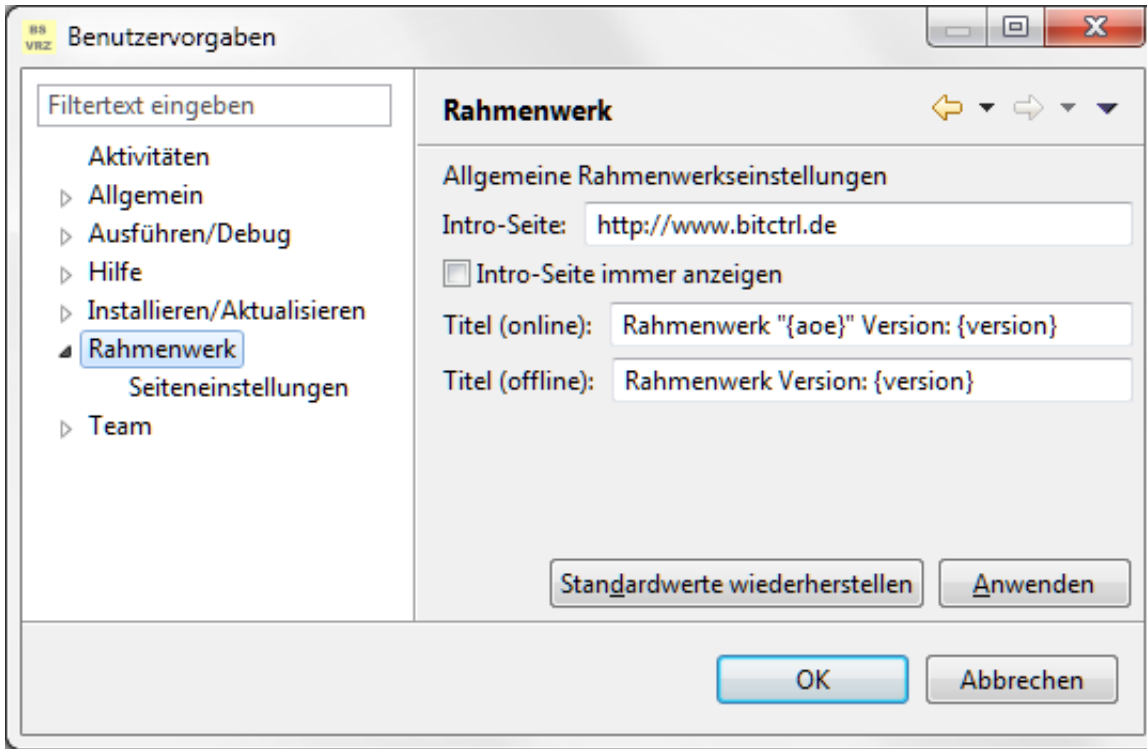
- **{user}** für den Name des angemeldeten Benutzers
- **{userid}** für die PID des angemeldeten Benutzers
- **{host}** für den Hostname aus der Datenverteileranmeldung
- **{port}** für den Port aus der Datenverteileranmeldung
- **{simvar}** für die eingestellte Standardsimulationsvariante
- **{aoe}** für den Name der verwendeten AOE
- **{aoepid}** für die PID der verwendeten AOE
- **{version}** für die Versionsnummer der Rahmenwerksoftware

---

## — Informationen im Titel des Hauptfensters der Rahmenwerkanwendung

---

Die im Titel des Rahmenwerk-Hauptfenster angezeigten Informationen werden als lokale Einstellungen abgelegt.



*Grundeinstellungen des Rahmenwerks*

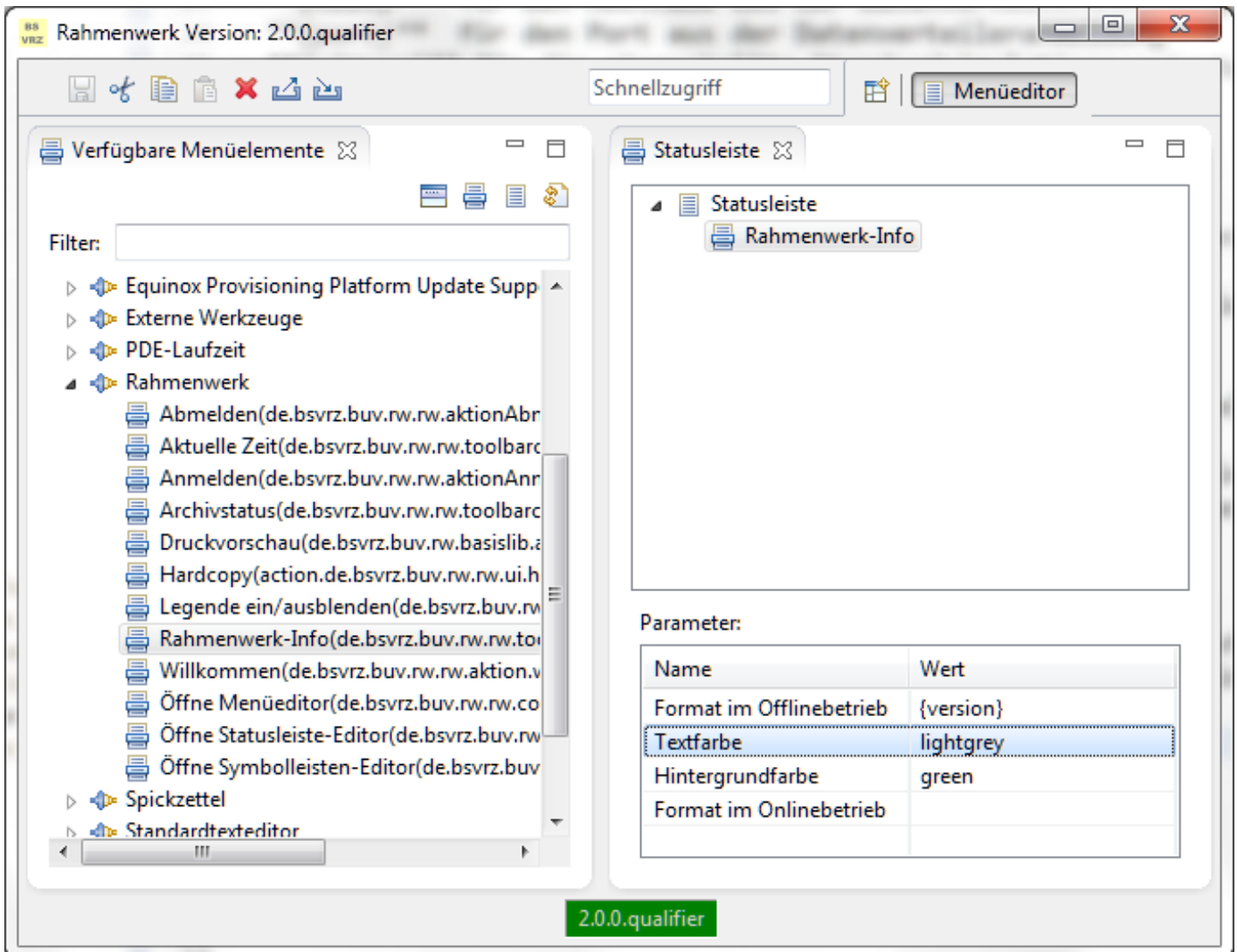
Einstellbar ist ein Titelformat für den Offline-Betrieb und ein Titel für den Online-Betrieb. Die Standwerte sind im Bild oben dargestellt.

---

## — Informationen in der Statusleiste des Hauptfensters —

---

Für die Anzeige der statischen Informationen in der Statusleiste wird ein Control bereitgestellt, das über den Menüeditor in die Statusleiste eingebaut werden kann.



*Menüeditor - Informationen Statusleiste*

Das Bild zeigt das entsprechende Element im Menüeditor des Rahmenwerks.

Das auszugebende Format wird über die Parameter “Format im Offlinebetrieb” und “Format im Onlinebetrieb” definiert.

## — Programmierschnittstelle —

Die allgemeinen statischen Informationen können auch in eigenen Anwendungen eingesetzt werden.

Das Enum *RwInfoDef* bietet eine statische Funktion, mit ein Formatstring mit den oben genannten Platzhaltern in einen mit Informationen befüllten String ausgegeben werden kann.

```
```java public enum RwInfoDef { .....
```

```

/**
 * liefert den formatierten Informationsstring, die enthaltenen Platzhalter
 * werden entsprechend ersetzt.
 *
 * @param formatStr
 *         der Formatstring
 * @return der resultierende String
 */
public static String format(final String formatStr) {
    return VERSION.berechneString(formatStr);
}

```

```

.....

```

```

} ""

```

# Legende

---

## — Hintergrund —

---

Die Ansicht "Legende" aus dem ursprünglichen Rahmenwerk wurde durch eine Legende ersetzt die als freischwebendes Legenden-Fenster an jeden beliebigen *WorkbenchPart* gekoppelt werden kann.

Die Legende wird über ein vom Rahmenwerk bereitgestelltes Kommando durch den Nutzer ein- bzw. ausgeblendet.

Damit einem *WorkbenchPart* eine Legende zugeordnet und diese angezeigt werden kann, muss dieser die Schnittstelle *ITreeLegende* oder *ICustomLegende* implementieren oder einen Adapter für eine dieser Schnittstellen liefern.

Der Inhalt der Legende wird durch die konkrete Implementierung bestimmt.

Liefert der *WorkbenchPart* einen Adapter für die Schnittstelle *IUpdateTimeProvider*, wird im Legendenfenster eine Aktualisierungszeit eingeblendet, die über die Schnittstelle per Polling abgerufen wird.

---

## — Schnittstelle ILegende —

---

Die Schnittstelle beschreibt eine für einen WorkbenchPart verfügbare Legende. Sie wird durch die konkreteren Schnittstellen *ITreeLegende* oder *ICustomLegende* erweitert.

```
``java public interface ILegende {
```

```
/**
 * Die Ecke des Workbench Parts in der das Toolfenster ausgerichtet werden
 * soll.
 */
static enum Corner {
    /** Oben links. */
    TopLeft("Oben links"),

    /** Oben rechts. */
    TopRight("Oben rechts"),

    /** Unten links. */
    BottomLeft("Unten links"),
```



```

    /** Unten rechts. */
    BottomRight("Unten rechts");

    private final String bezeichnung;

    private Corner(final String bezeichnung) {
        this.bezeichnung = bezeichnung;
    }

    public String getBezeichnung() {
        return bezeichnung;
    }
}

/**
 * Gibt ein Control zurück, an dem das {@link LegendeWindow} ausgerichtet
 * wird. Verändert das Control seine Position, dann folgt die Legende dem
 * Control.
 *
 * <p>
 * Das Control, welches hier zurückgegeben wird, ist nicht das
 * Control der Legende, sondern ein Control im, Workbench Part für das die
 * Legende angezeigt wird.
 *
 * @return das Control oder <code>null</code>, wenn die Legende nicht folgen
 *         soll.
 */
Control getControl();

/**
 * liefert die Ecke des Workbenchparts an dem die anzuzeigende Legende
 * standardßig ausgerichtet werden soll.
 *
 * @return die Ecke
 */
Corner getDefaultCorner();

/**
 * liefert einen Titel zur Anzeige im Legendenfenster (optional).
 *
 * @return den Titel oder <code>null</code>
 */
String getTitel();

}

```

---

## — Schnittstelle ICustomLegende —

---

ist die Schnittstelle für eine nicht näher bestimmte nutzerdefinierte Legende. Der Inhalt der Legende wird über die Funktion *createControl* erzeugt. Es wird lediglich das übergeordnete Composite, das befüllt werden soll übergeben.

```
```java public interface ICustomLegende extends ILegende {
```

```
    /**
     * erzeugt das anzuzeigende Control.
     *
     * @param parent
     *         der umgebende Container, in dem das Control eingebettet werden
     *         soll
     */
    void createControl(Composite parent);
```

```
} ````
```

---

## — Schnittstelle ITreeLegende —

---

ist die Schnittstelle für eine Legende, die Elemente in einer Baumansicht darstellt. Der Inhalt der Legende wird über die Funktion *getBausteine* erzeugt, welche die Wurzelemente der Baumdarstellung liefern.

```
```java public interface ITreeLegende extends ILegende {
```

```
    /**
     * Gibt die Bestandteile der Legende zurück.
     *
     * @return die Liste der Bausteine
     */
    List<ILegendeBaustein> getBausteine();
```

```
}
```

```
public interface ILegendeBaustein {
```

```

/**
 * Gibt die untergeordneten Legendenbausteine zurück. Gibt es keine, wird
 * eine leere Liste zurückgegeben.
 *
 * @return die Liste der untergeordneten Bausteine
 */
List<ILegendeBaustein> getBausteine();

/**
 * Gibt das kleine Bild des zu beschreibenden Objekts zurück.
 *
 * @return das Image für die Darstellung des Elements oder null
 */
Image getIcon();

/**
 * Gibt eine kurze Objektbeschreibung zurück.
 *
 * @return die Beschreibung des Elements zur Darstellung im Legendenbaum
 */
String getText();

}

```

Die Bausteine selbst können wieder untergeordnete Bausteine enthalten, womit sich letztendlich eine Baumdarstellung ergibt.

Ein Baustein liefert ein anzuzeigendes Label und ein optionales Icon.

# Simulation

Die Unterstützung von Simulationsvarianten innerhalb des Rahmenwerks ist momentan aus verschiedenen Gründen nur als Proof of Concepts zu betrachten.

- der Umgang der Plug-ins des Rahmenwerks und die Behandlung unvollständig parametrierter bzw. nicht konfigurierter Simulationen ist momentan nicht vollständig geklärt
- lokale Einstellungen des Rahmenwerks werden in einem zentralen Verzeichnis abgelegt, d.h. sie gelten auch für die Simulation und könnten im Rahmen der Simulation mit Auswirkungen auf den “Normalbetrieb” verändert werden
- das Eclipse 4 Framework bietet zwar die Möglichkeit per Theming die Optik der Workbench zu ändern. Die zur Verfügung stehende Funktionalität ist aber derzeit teilweise noch im Incubation-Status und die Tools und die Dokumentation für das Theming noch relativ unvollständig.

Das Rahmenwerk kann über die Standard-Datenverteilerparameter (*-simVariante*) mit einer Standardsimulationsvariante ausgeführt werden.

Folgende Möglichkeiten in Bezug auf die Simulationsvarianten bietet das Rahmenwerk derzeit:

---

## — Anpassung des Layouts —

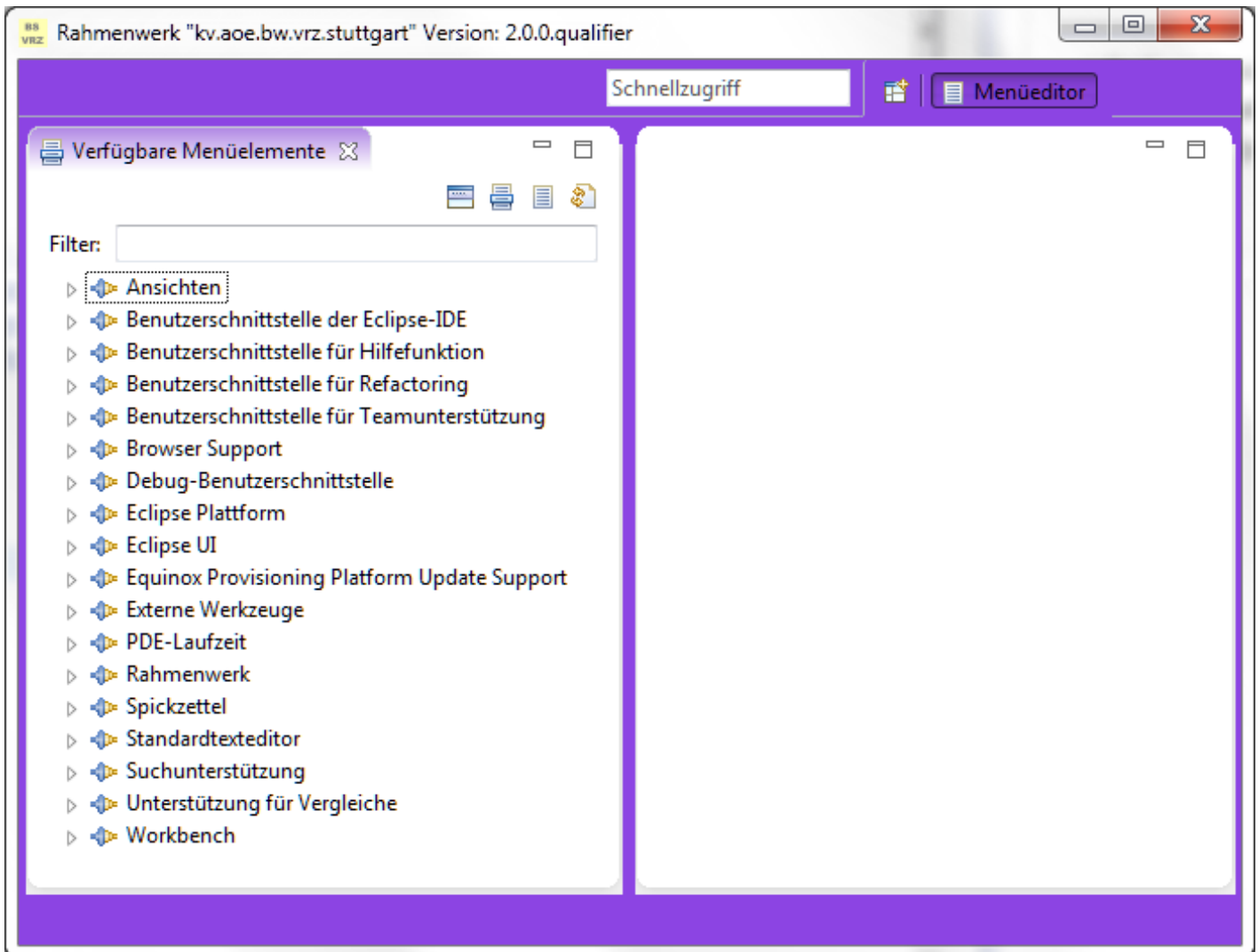
---

Das Rahmenwerk stellt zwei Themes zur Verfügung, Standard- und Simulationsdarstellung. Die Themes sind als ExtensionPoints definiert und per CSS-Datei anpassbar.

Das Rahmenwerk-Plug-in selbst wird nicht als jar-Archiv sondern in extrahierter Form installiert, um den einfachen Zugriff auf die CSS-Dateien zu ermöglichen.

Die CSS-Dateien “defaultstyle.css” und “simulation.css” sind im Unterverzeichnis *css* des Rahmenwerk-Plug-ins zu finden.

Das Theme “Simulation” wird aktiviert, wenn eine Simulationsvariante beim Start des Rahmenwerks übergeben wurde und eine Datenverteilerverbindung besteht.



*Rahmenwerk mit Thele "Simulation"*

Momentan wird die Hintergrundfarbe des Hauptfensters manipuliert.

---

## — Anzeige der Simulationsvariante als statische Information —

---

Die eingestellte Simulationsvariante kann optional als statische Information im Titel der Anwendung oder in der Statuszeile eingeblendet werden.

# Migrations-Hinweise

Die Migration eines bestehenden Plug-ins erfordert im wesentlichen folgende Maßnahmen:

- Anpassen der Abhängigkeiten der Rahmenwerks-Plug-ins (Versionierung)
- Ersetzen der Singletons für Einstellungen, Berechtigungen und Datenverteilterverbindung durch die entsprechenden Services des Rahmenwerks
- Umstellung von Actions in ActionsSets auf Commands
- Definition von Berechtigungen per ExtensionPoint

---

## — Migrationssupport —

---

Im ersten Schritt kann in einem bestehenden Plug-in die Abhängigkeit zum Plug-in *de.bsvrz.buv.rw.migrationsupport* ergänzt werden.

Damit sollte der bestehende Code in den meisten Fällen kompilierbar sein.

Folgende bekannte zusätzliche Eingriffe sind zusätzlich potentiell erforderlich:

- das zum Drucken verwendete Projekt *Paperclips* wurde von SourceForge zu einem Eclipse Nebula Projekt übernommen. Dabei wurde der Name und die Packages von *net.sf.\** auf *org.eclipse.\** geändert und die Version von 1.x auf 2.x angehoben. Die Imports in betroffenen Paketen müssen aktualisiert werden.
- Das Plugin *org.junit4* gibt es nicht mehr. Es gibt nur noch *org.junit*. Das aber in Version 4.x. Abhängigkeiten zu diesem Plug-in müssen entsprechend angepasst werden.
- die Schnittstelle *ILegende* hat zwei neue Methoden, die eventuell in den betroffenen Klassen nachgeführt werden müssen.

Der **Migrationssupport** ist kein offiziell unterstütztes Plug-in für den Produktiveinsatz sondern dient lediglich in dem Sinne der Erleichterung der Portierung, das der Code im Wesentlichen kompilierbar bleibt und mit hoher Wahrscheinlichkeit auch funktioniert.

Der Entwickler hat dann die Möglichkeit schrittweise die veralteten Komponenten und Funktionen durch die neuen zu ersetzen.

**Eine Portierung ohne Migrationssupport ist in jedem Fall angeraten!**

---

## — Anpassungen ohne Migrationssupport —

---

- die API des Urlasserdialogs hat sich geändert (neuer Klassenname *UrlasserDatenDialog*).
- die API der Rahmenwerkseinstellungen hat sich geändert
- die API für das Drucken hat sich geändert (Rahmenwerk-API für die Nutzung von *Paperclips*).

- der Extension Point `org.eclipse.ui.actionSets` ist deprecated und muss durch `org.eclipse.ui.commands` ersetzt werden.
- der Extension Point `org.eclipse.ui.viewActions` ist deprecated und muss durch `org.eclipse.ui.menus` ersetzt werden.

---

## Urlasserdialog

---

Der Urlasserdialog wird jetzt wie ein normaler Dialog verwendet und das Ergebnis nach dem Schließen an diesem abgefragt. Es gibt keine Methode mehr die den Dialog öffnet und das Ergebnis zurück gibt.

Stattdessen wird dem Urlasserdialog im Konstruktor eine *UrlasserDatenSender* übergeben, der die entsprechenden Aufgaben in der Kommunikation mit dem Datenverteiler übernimmt.

*Zum Thema "Urlasserdialog" gibt es ein gesondertes Kapitel!*

---

## Drucken

---

Der Name der Schnittstelle *IDruckvorschau* ist jetzt *RwPrintable* und liegt in einem anderem Package. Die Schnittstellen beiden Interfaces sind identisch, d.h. es müssen lediglich die Namen korrigiert und Imports angepasst werden.