Easy E-Government-Application

Implementation study of an open source E-Government-Application Case Study

Investing in the future by working together for a sustainable and competitive region

Implementation study of an open source E-Government-Application

1 Content

2 Executive Summary

Like many other universities, the University of Applied Sciences in Wilhelmshaven, is a public institution and as such is subjected to the relevant legal norms. The following communication scenarios between the students and the university occur regularly and must be supported:

- Legally binding disclosure of general information, e.g. on test conditions and course conditions
- Legally binding disclosure of information concerning individual study situations, e.g. performance certificates for exchange students
- Processing individual applications submitted by students, for example recognition of exams from other universities
- Compliance with the legal requirements for data protection

Therefore we are building an e-government system called ”Easy Egov”. The system can be, as primary task, publish legally binding forms and receive and archive the responses of these filled forms.

A system for capturing legally binding forms is composed of several components and all together will make the e-government application. These include as components:

- a complete form server with versioning,
- a document management system
- a complete user management sytem with an PKI-implementation (the Public Key Infrastrucre will provide the legally authentification of users),
- a workflow system and
- a backup system

Therefore must be respected several laws, regulations and technical guidelines. In addition the system must be sustainable. That means, for the full retention period of the documents (up to 30 years) the application must be run or it must be possible to transfer the functionality to another system.

It is therefore a main task in the development to store the data in simple structures as XML documents. All data will be encrypted to prevent the unauthorized scanning and reading. Only in the back up can be stored the data unencrypted, in this case the administrator is responsible for the security of the data.

The documents should be available in different transmission-media (HTML, PDF, XML forms) and -methods (web, email). For the initial implementation a web-portal with HTML-forms is developed. For this a module (an extension) for the CMS DRUPAL is written.
This has the advantage that some features of DRUAPL, such as the user management, can be integrated. Also the theming (DRUPAL-wording for displaying of the data), the HTML code including the CSS-design, will be done by the DRUPAL system.

Involved is Prof. Dr. C. Wunck and Dipl.-Winf. (FH) U. Bachmann at UAS Wilhelmshaven.

Date: 2011-04-27

3 Problem Statement

The application should store and manage forms and form-responses in a legal binding form. Also an official notification, because of the read receipt, should be represented in a form, so the system can also be used for official communications.

Since „easy E-Government“ deals with data for administrative acts, the system requirements are special, and laid down in laws, administrative regulations and technical guidelines (see Case Study “Feasibility Study of an Open Source E-Government-Application”).

3.1 Legal Requirements

The most important legal requirements are:

- Form data management with the possibility of unique user identification and authentication.
- Form and form data management with the ability to store data up 30 years and in a reasonable time to restore. The period is limited by record retention periods for the qualified electronic signatures.
- Form data management with the possibility data changes to demonstrate and to compare two documents together.
- Forms management with the option to rebuild forms at any time with the submitted form data.
- Data security, particularly the unauthorized reading and scanning data needs to be prevented on system side.
- Data reduction and data economy. This concept refers to the fact that only absolutely necessary data collected and this data are only kept as long as needed (german federal data protection act - BDSG §3).

3.2 Requirements oft the developers

Further requirements of the developers are:

- Implementation of the application in an open-source CMS (Content management system).
- To create a modular and expandable application.
- The application should be multilingual. As the default languages are English and German provided.
- The opportunity the application to implement in any other system, not only CMS’s.
- The opportunity to use different transmission media (HTML, PDF, XML form) and method (web, email) to.

3.3 Special Problems

Some of the points from the results in section 3.1 and 3.2 throwing particular problems.

So today can not foresee, we speak about a period up to 30 years, with which data formats will be worked in future. We also cannot forecast with which data management systems will be used in future.

Another problem is the today common timestamp formats as integer. The integer number will run out soon and should not used for the application.

Today's common qualified electronic signatures (at least in Germany) could be replaced by other methods. In other European countries are different signature options used but should be usable.

From this requirements and problems leads the following planned implementation.

4 The implementation

4.1 The Content Management System (CMS) Drupal

As CMS for the prototype will be used DRUPAL in the version 6.x (see http://www.drupal.org). Drupal offers the advantage that it is a highly secure open-source CMS. A security team handles all noticed vulnerabilities.

Drupal is a free and open source content management system (CMS) and Content Management framework (CMF) written in PHP and distributed under the GNU General Public License. Drupal runs on any computing platform that supports both a web server capable of running PHP (including Apache, IIS, Lighttpd, and nginx) and a database (such as MySQL, MariaDB, PostgreSQL, SQLite, or Microsoft SQL Server) to store content and settings. Drupal has founded as open source project in 2001. It is now used for many hundreds of thousands of Drupal sites of all kinds all over the world, for example for the reader's comments on the website of the weekly newspaper "Die Zeit" or since 24 October 2009, the website of the White House (http://www.whitehouse.gov) in Washington.

The Drupal core is the stock element of Drupal. It provides the basic functionality, and offer modules that provide additional functions and the system can be added if needed. There are currently (as of 30 January 2011) a total number of over 7,500 modules hosted on drupal.org, 5,000 of which are marked as compatible to Drupal 6 and over 1,000 for compatibility with Drupal 7.x. These modules offer a wide range of simple, common website features, to complex via graphical user configurable tools, to extensions to the already extensive programming interfaces.

The modular design of Drupal allows a multi-purpose usage. The list ranges from "one-person Web sites" such as personal weblogs to online communities with thousands of members.

For the most Drupal-Modules are language-libraries available. For new modules it is easy to create a new language-library.
Also available for Drupal is a number of final modules are available that can be integrated into the application, such as a LDAP module. A complete module list is available at http://drupal.org/project/modules.

Another great advantage of Drupal is the easy way to create custom modules. These modules are based on PHP 5.x and databases (e.g. MySQL 5.x) and can use the Drupal-HOOK-system (Drupal-wording for interfaces) for the programming. A description of the Drupal-functions and -interfaces is available under http://api.drupal.org/api/drupal/6.

Drupal utilizes the MVC - Model-View-Controller - architectural pattern. The design pattern divides a software system in three different areas.

- Model: data - that is organized by the system
- Controller: control component that receives requests, calls the business intelligence and prepare the relevant data to display the View
- View: Components of a system responsible for displaying the data (theming).

This means, the custom module just takes care of the model- and controller-developing and the design remains in the hand of the active Drupal-theme.

4.1.1 Drupal Module „egov"

For e-government application is a Drupal-module "egov" created. This module uses the hooks in Drupal and implements the application with some separate interfaces. The application is available as a PHP include files and can be completely transferred to other systems (e.g. TYPO 3). Only the CMS-integration needs to be rewritten.

The "Easy E-Government-Application" can be connected only under a SSL connection. Other connections will be intercepted and redirected to a SSL connection. In this way we ensure that the connection will not be intercepted and read out.
The administrator of the application is responsible to provide a faultless SSL certificate.

The CMS has to extract some paths from the URL and route them to the "egov"-module. The "egov"-module will check the connection (SSL), check if the requested function or form is available, the user has the authorization for the request and finally execute the function or form.

The user management is handled by CMS. The identity will be verified after the forms returned by, in the form, specified in the procedure. Therefore the application will provide some authentication-mechanism.

The module-package will be implemented, in the file system, as follows:

	Type	Description
egov	Directory	Drupal-Module, contains the whole application.
egov.info	Script	Drupal-specific script to register the module in Drupal.
egov.install	Script	Drupal specific script to install the module. Via this script will be create the database-tables and –contents, create default-values and system variables. Also can be done updates for further versions via this script.
egov.module	Script	Drupal specific script with the Hook-implementation and customized functions.
egov	Directory	Easy-E-Gouvernement-directory with all scripts of the application.
basic	Directory	
class.eg_init.php	Script	The central entry script for the application. In this script will be done all preliminary tests; as secureness-, rights- or availably-tests.
datasafe.inc	Script	An, per session, unique data-container. All

		scripts of the application can use it and communicate via this script.
…	Script	
services	Directory	Container for all services and utility classes.
Class_db.inc	Script	Database class with the implementation of all functions from the database-interface.
db_interface.inc	Script	Database-Interface.
xmlHandler.inc	Script	The class will handle all XML-activities. It is designed to use the SimpleXML-function from the PHP Framework. It is not necessary to get an instance from this class. Normally it is sufficient to include the class and call the functions as "Paamayim Nekudotayim" (Scope Resolution Operator) xmlHandler::getObjectFromString($xmlString)
…	Script	
form_handler	Directory	
class_formDesigner	Script	Governs the design and handling to create and edit forms, versioning, and rights handling in the creating and processing of forms.
class_formHandler	Script	Controls the access to forms, the preprocessing, delivery and receipt of responses, validation, filling and signing the forms.
…		
…	Directory	
backup	Directory	Directory to store backups and archives. The data are not stored in plain text.

4.1.2 The rights concept of Drupal

Drupal uses a role concept to set the permissions. Each user will be at least assigned to one role. A non-registered user automatically has the role "guest", a registered user the role "authenticated user". The administrator can define any new roles and assign them to the user. With the using of LDAP registration can be included the organization-roles in the Drupal system.

4.1.3 The rights concept of the „egov“-Module

The "egov" module also uses the role-based permission concept of Drupal and will enlarge it by a few points.

In either form it will be possible to define access roles for the rights “view”, “handle” and “edit”. “view” means the form-filling access; which includes the call, fill out, returning and restoring submitted forms. “handle” is the access to the submitted form-data, mostly for employees and with the “edit”-access it is possible to edit the form.

A user, regardless of his current role or the current distribution of rights in the form, always has access to by himself submitted forms and can restore them.

In addition of the role access it will be possible to define IP-based positive and negative lists. These lists will have always priority against the roles.
Positive lists will be defined in the forms and allow visibility restrictions on sub-networks, such as the own company network (e.g. 139.13.*.*). User from outside the defined sub-net will be refused.
The negative lists are managed in the administration area and allow for rejection of individual IP addresses or subnets. In this way, an administrator can exclude robots, spiders and hackers.

4.1.4 The function- and hook-implementation of the „egov“-module

The application requires very few functions and hooks (interfaces) in the Drupal module. The most important implementation is the menu-hook. The menu-hook defines the required paths and redirects the request to defined functions. It is possible to define wildcards for the paths used.

```
function egov_menu()
{
            $items['egov/basis_administration'] = array( …
            $items['egov/basis_administration/global_settings'] = array( …
            $items['egov/basis_administration/backup_settings'] = array( …
            $items['egov/%'] = array( …
            $items['eg/%'] = array( …
            $items['eg'] = array( …
```

The path “../egov/ basis_administration/*” will be contain the administration area.
The path “../egov/*” leading in an especially protected administration area of the application.
The path “.. / eg / *” directs the user into the application. The final resolution of the path is in the application; Class eg_init:: check_path (). This makes it possible forms with name, identifier or an alias to call and extend the paths as desired;

- https://myApplication.de/eg/form1
- https://myApplication.de/eg/60c0b53095f81a7bf551b30c93fd20dd
- https://myApplication.de/eg/60c0b53095f81a7bf551b30c93fd20dd/submited
- https://myApplication.de/eg/60c0b53095f81a7bf551b30c93fd20dd/submited/page/2

The path length is limited to 250 characters by the application. This also provides an implementation on a MS SharePoint server.

Further is a “perm”-hook (permission) is implemented. This implementation allows to defining permissions for the application directly in Drupal.

4.2 Use Cases

The main functions of the application "Easy E-Government" are creating forms and filling out, receiving of form data and their signing.

These functions and some other features are illustrated in the following use cases schematically and simplified.

4.2.1 Design Forms

Only individual or groups, authorized by the administration, have the rights to create forms. The access will assure by the user management of the CMS and will be set on role permissions.

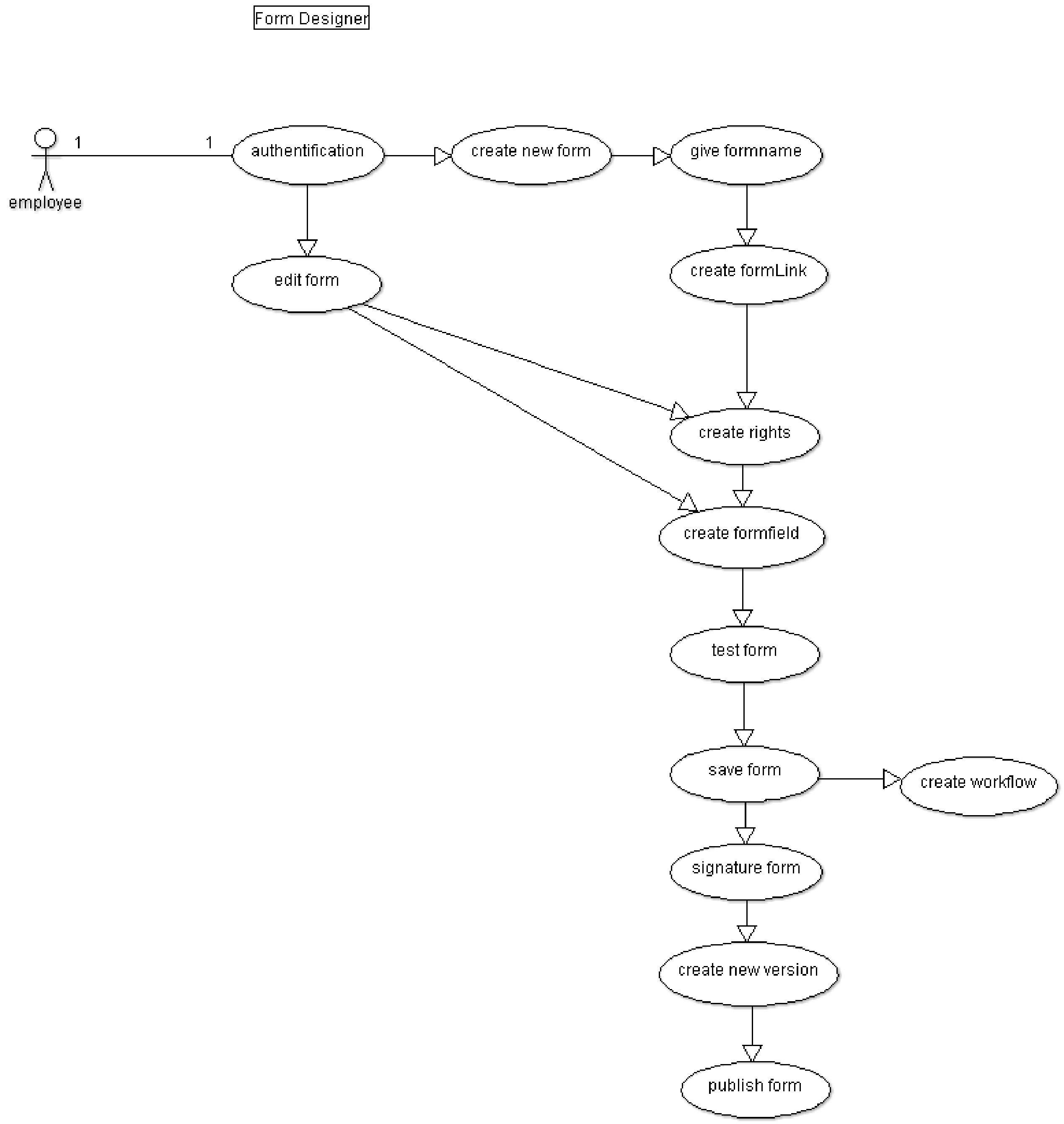


Figure 1 - Create forms

An authorized user has the choice to edit a form or create new one. If a new form created must be assigned a form-name as the first and the system creates a unique link under which the form is reached. It is also possible to generate URL aliases, for example "speaking links " as “../eg/myForm1”.

With the first step, the form is created in principle, but until the conclusion of the form creation the form is inactivated, i.e. only the owner has access.

Next, restrictions on access are set to the form. The access rights may relate to persons, roles or/and IP- /subnet-addresses.

Then any form of components and workflows can be added.

At the end, the form can be signed. So the user has the assurance that their data get only to authorize persons.

Before the activation of a form a version number will be generated. A version number must be created always new if a form is created or at least a request for this form received.

4.2.2 Use Forms

Access to a form can be divided into two areas - before filling out a form and after filling.

In the forefront of the form delivery, the application will made all security queries – is it a SSL-Connection, exists the form, is the user authorized for the requested access, is the form activated and published and so on.

If errors are found, the user is informed by a message about it.

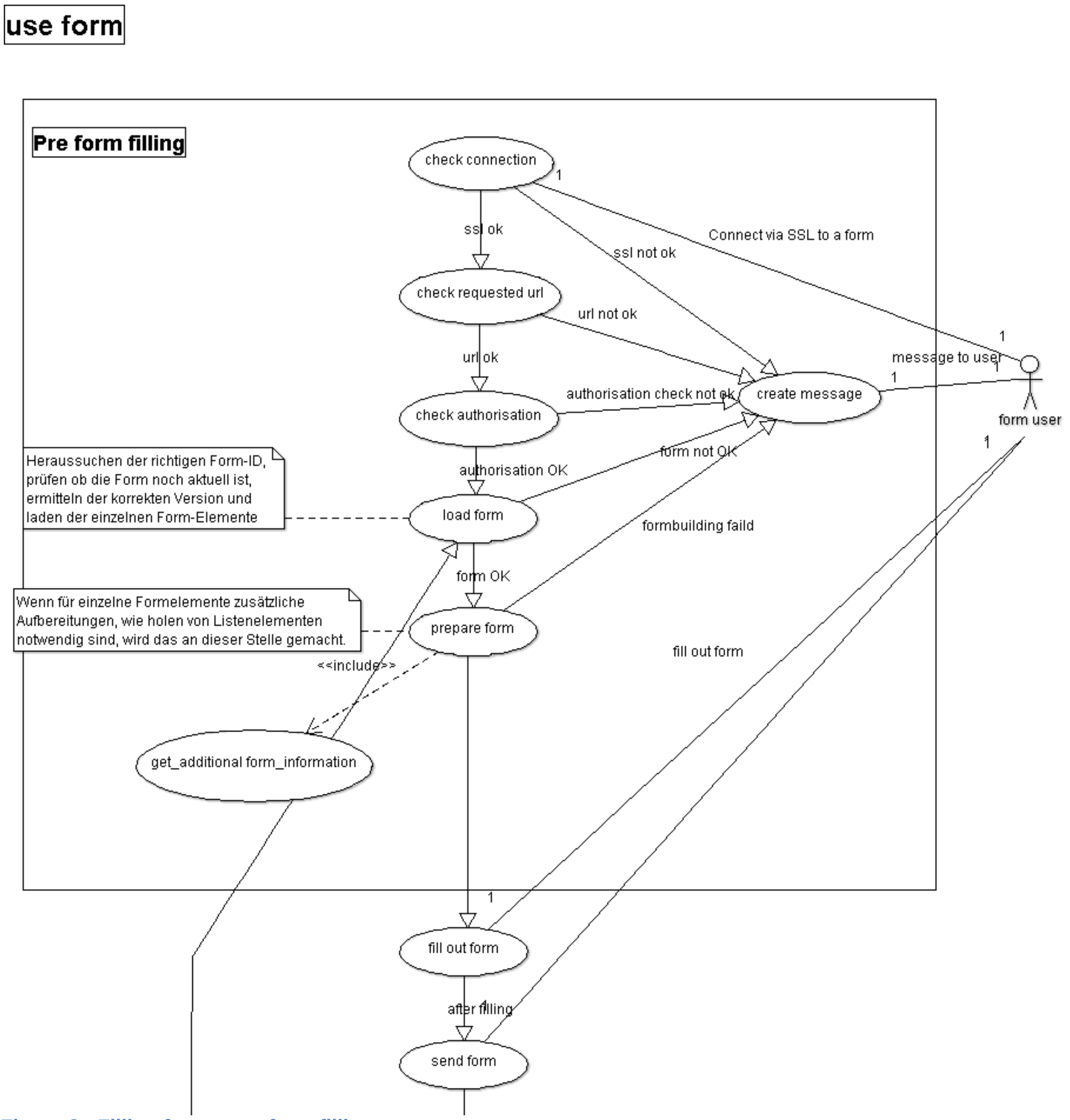
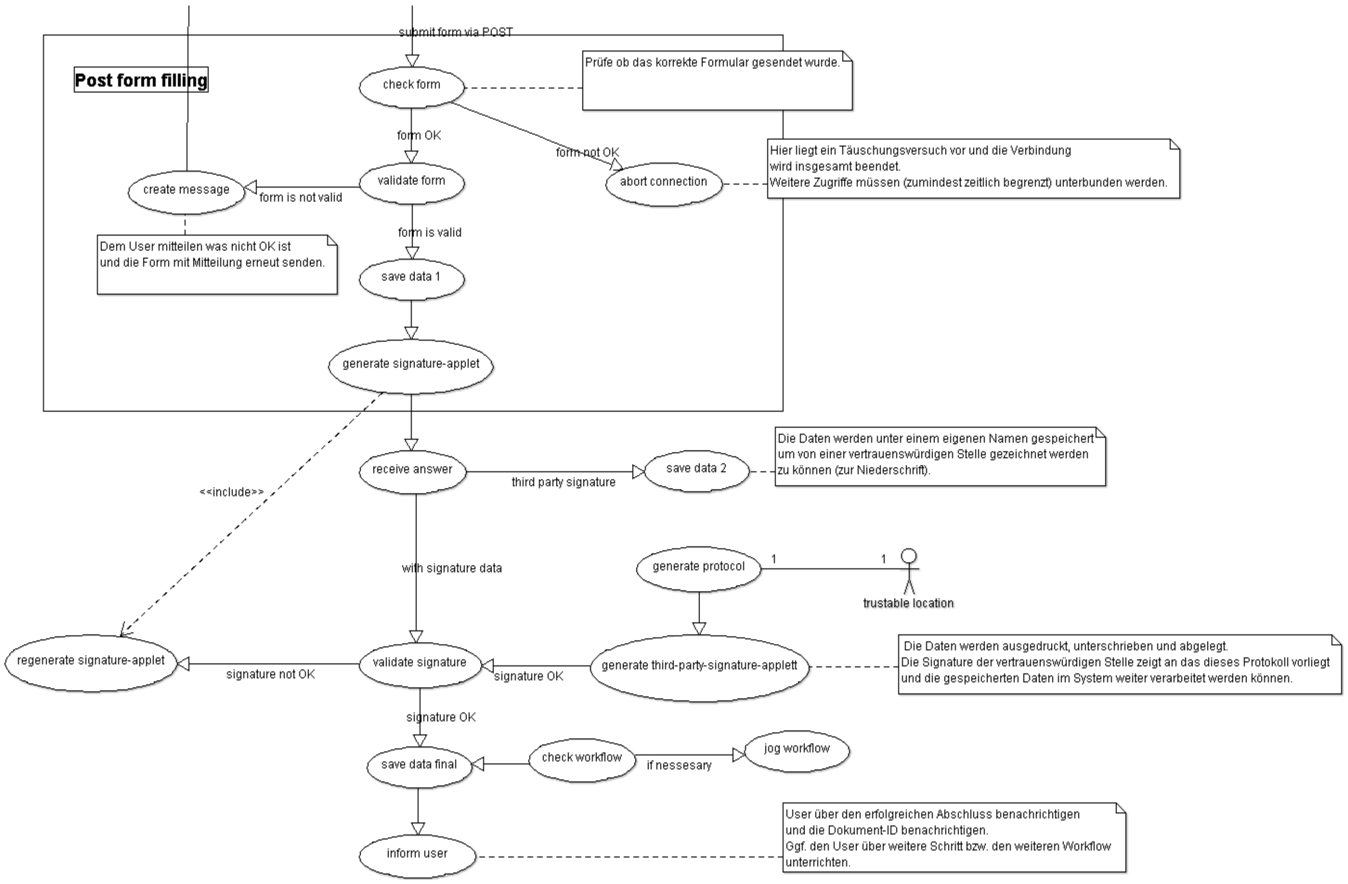

use form
Pre form filling
check connection
ssl ok
ssl not ok
Connect via SSL to a form
check requested url
url not ok
url ok
check authorisation
authorisation check not ok
create message
message to user
form user
1
1
1
1
Herausuchen der richtigen Form-ID, prüfen ob die Form noch aktuell ist, ermitteln der korrekten Version und laden der einzelnen Form-Elemente
authorisation OK
form not OK
load form
formbuilding faild
form OK
Wenn für einzelne Formelemente zusätzliche Aufbereitungen, wie holen von Listenelementen notwendig sind, wird das an dieser Stelle gemacht.
prepare form
<<include>>
fill out form
get_additional form_information
1
fill out form
after filling
send form

Figure 2 - Filling forms, pre form filling

In front of every transmission the form will be added with a “transmitted_id”. If the form is sent back the “transmitted_id” must exists in the system. Otherwise the form is not accepted. This prevents a multiple submitting of forms (e. g. F5 key or downloaded PDF).


Post form filling
submit form via POST
check form
Prüfe ob das korrekte Formular gesendet wurde.
form OK
form not OK
abort connection
Hier liegt ein Täuschungsversuch vor und die Verbindung wird insgesamt beendet. Weitere Zugriffe müssen (zumindest zeitlich begrenzt) unterbunden werden.
validate form
form is not valid
create message
Dem User mitteilen was nicht OK ist und die Form mit Mitteilung erneut senden.
form is valid
save data 1
generate signature-applet
<<include>>
receive answer
third party signature
save data 2
Die Daten werden unter einem eigenen Namen gespeichert um von einer vertrauenswürdigen Stelle gezeichnet werden zu können (zur Niederschrift).
with signature data
generate protocol
1
1
trustable location
regenerate signature-applet
signature not OK
validate signature
signature OK
generate third-party-signature-applett
Die Daten werden ausgedruckt, unterschrieben und abgelegt. Die Signature der vertrauenswürdigen Stelle zeigt an das dieses Protokoll vorliegt und die gespeicherten Daten im System weiter verarbeitet werden können.
signature OK
save data final
check workflow
if nessesary
jog workflow
inform user
User über den erfolgreichen Abschluss benachrichtigen und die Dokument-ID benachrichtigen. Ggf. den User über weitere Schritt bzw. den weiteren Workflow unterrichten.
Figure 3 - Filling formes, post form filling

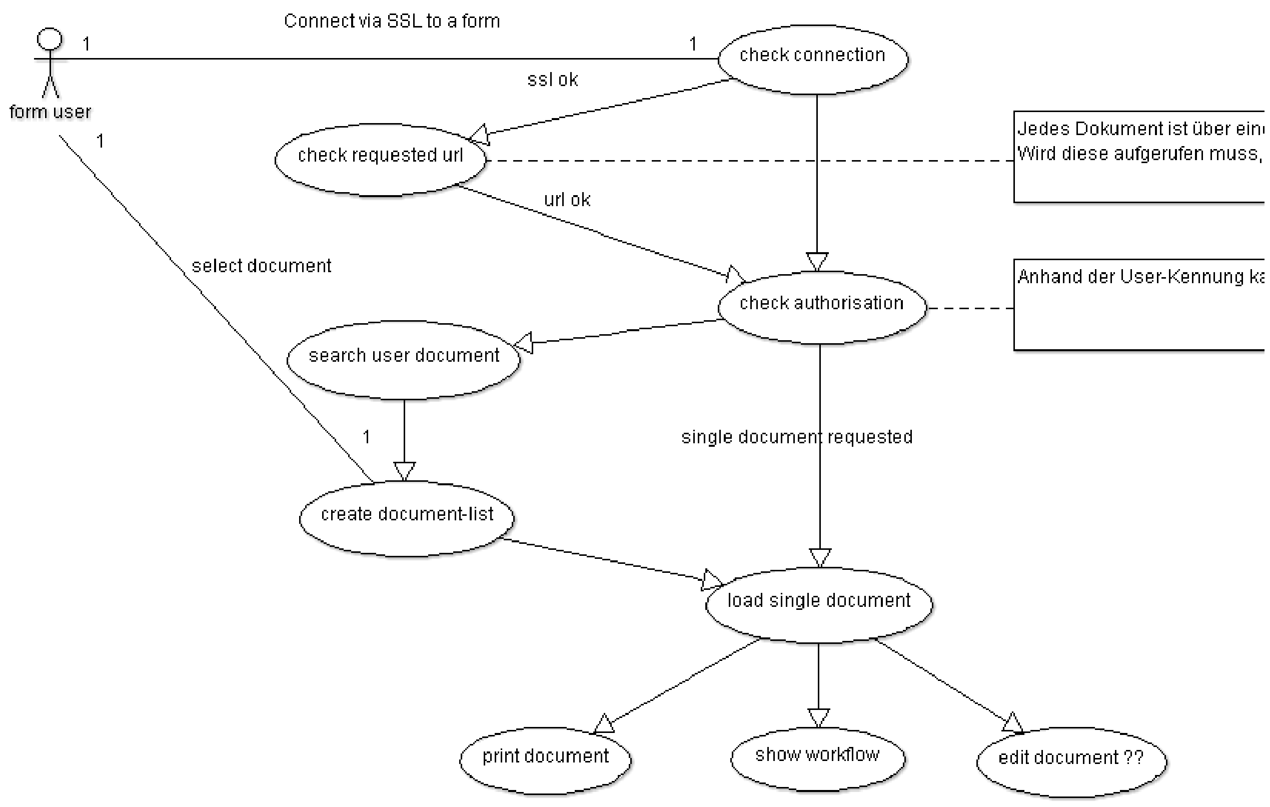
Comes a filled form back to the system, the application will check whether the form is still there and the form to the input date is still active.

Thereafter, the form contents are validated. There are standard validation functions, such as only numerical values, email validation, check for empty content, etc. But it can also validate additional functions in the form components are deposited.

Is the form successfully validated, the system generate the signature applet and the user can sign the form. The signature is checked and, if correct, generates the application a document hash and sent it to the user. Also the document will be activated. With the document hash, the user has the sureness that the form has been accepted and processed.

4.2.3 Restore user documents

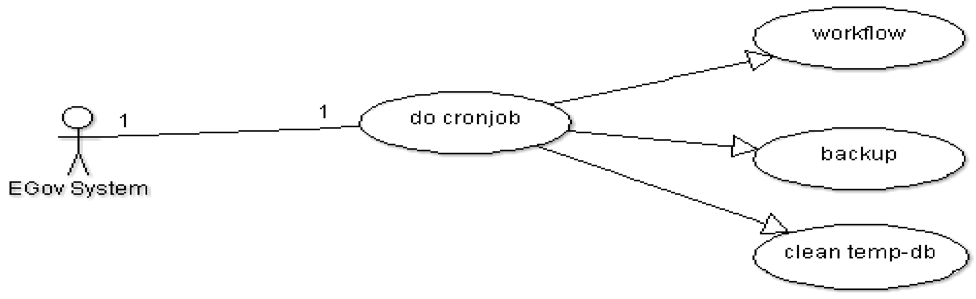
Each user has the ability to view their documents again. Therefore a user is checked by user-ID. Then he can see a list of documents or call a document on the document ID.



Before the delivery of a document the permissions will be checked. Subsequent processing of a document is usually not provided.

4.2.4 Cronjobs

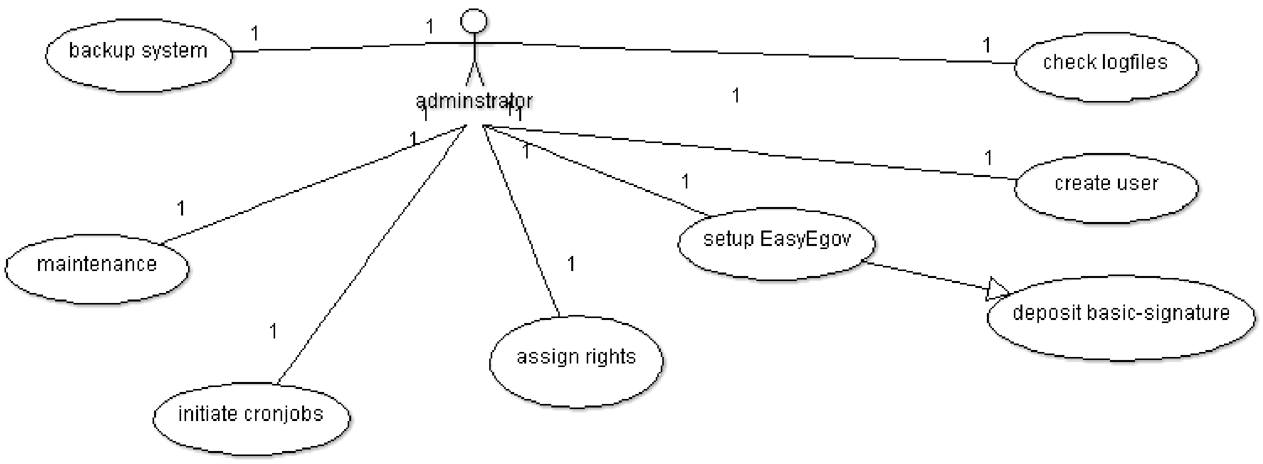
For the smooth handling of the application are cron jobs necessary.



During the cron job will be done workflows, backups and the cleaning of the database.

4.2.5 Administrative jobs

The administrator has to perform a series of initial and routine tasks and thereby provide the functionality of the system.



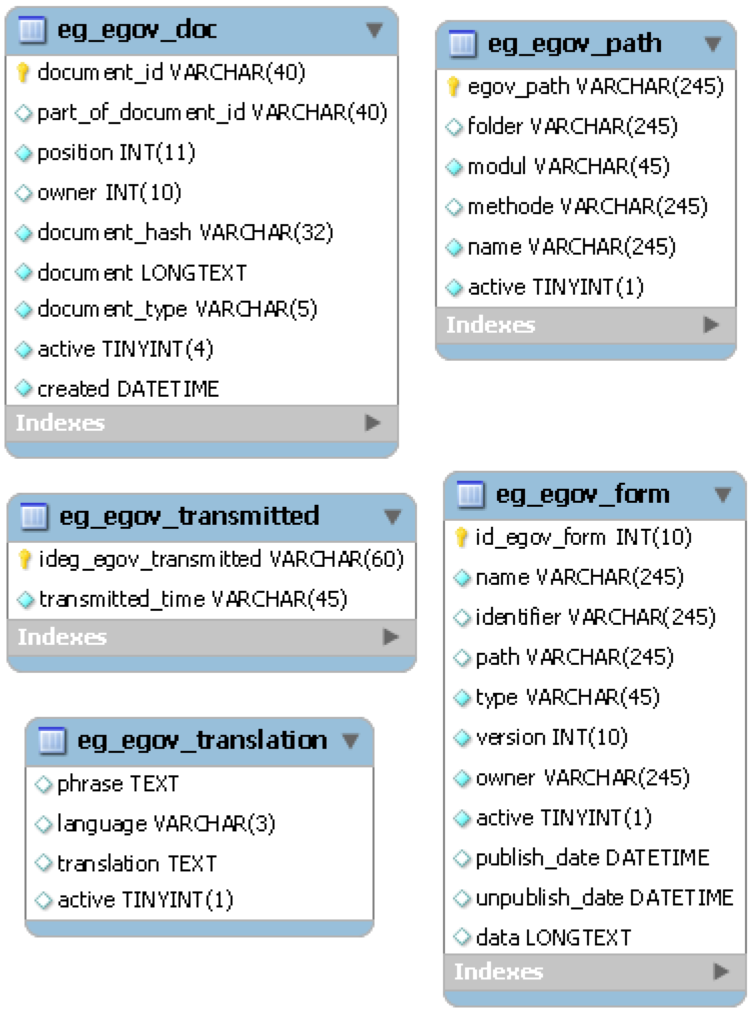
4.3 Structure of the data

To ensure the compatibility of the data for other systems and the future all important data will stored in a XML structure.

These structures can be stored easily and transmitted in any data system. In addition, the data will be encrypted in the XML tags, so that a search in the database is not readily possible (a requirement of the German law BDSG §4a). The disadvantage is the need to transfer the data into an object structure for working in the application. There are, however, utility classes for XML processing in all considered systems. The “egov”-module provides an interface for the XML-handling with necessary functions. In the current version of the class “xmlHandler” the functions are developed with the SimpleXML framework. You can also use any other framework or your own classes.

To simplify and speed up the processing, the XML structures are stored in the database. Only those data are unencrypted stored in the database to facilitate the processing and do not contain personal or other data protection-relevant data.

4.3.1 The database structure of the „egov“-module



eg_egov_doc
- document_id VARCHAR(40)
- part_of_document_id VARCHAR(40)
- position INT(11)
- owner INT(10)
- document_hash VARCHAR(32)
- document LONGTEXT
- document_type VARCHAR(5)
- active TINYINT(4)
- created DATETIME

Indexes

eg_egov_path
- egov_path VARCHAR(245)
- folder VARCHAR(245)
- modul VARCHAR(45)
- methode VARCHAR(245)
- name VARCHAR(245)
- active TINYINT(1)

Indexes

eg_egov_transmitted
- ideg_egov_transmitted VARCHAR(60)
- transmitted_time VARCHAR(45)

Indexes

eg_egov_translation
- phrase TEXT
- language VARCHAR(3)
- translation TEXT
- active TINYINT(1)

eg_egov_form
- id_egov_form INT(10)
- name VARCHAR(245)
- identifier VARCHAR(245)
- path VARCHAR(245)
- type VARCHAR(45)
- version INT(10)
- owner VARCHAR(245)
- active TINYINT(1)
- publish_date DATETIME
- unpublish_date DATETIME
- data LONGTEXT

Indexes

Figure 4 - Database structure

4.3.2 Structure of the forms

It is possible to generate forms in various formats. For the initial implementation forms will be in HTML formats prepared. But also other formats are possible, e. g. PDF, XML- or MS office forms.

4.3.2.1 Structure of the forms

Forms are divided into two areas, general data and form fields. Basically, a form is constructed as follows:

Area	Needed	Description
• Basis data	always	The basic data includes the general information of forms - name, identifier, path, type, version, owner, permissions, publication date and unpublished date. These data are needed to simplify the discovery of the forms. Another important specification is the expected signature

		method to confirm the form.
• Form data	always	The form data area is only a container fort he form pages. In the form data area are no further attributes.
o Form page	1 - n	The tag “pages” contains the attributes required to create a form page, e. g. the action-path, special MINE-types or CCS classes. Not all attributes are obligatory – see tables ”Form Component Type Definition 1”. The tag “page” contains the form components of the form page. To activate a form it is necessary to define at least two form components (form content and submit button). But you can define any number of components in any number of pages.
▪ Form component	2 - n	Form components are the atomic structures of forms, such as text, text boxes, selection lists or radio button. Each form component can have attributes, such as CSS classes, validation instructions, default values, etc. – see tables ”Form Component Type Definition 1”.

You can see the exactly implementation of the form components in the next tables.

a = attributes
r = required
o = optional
- = not available

Table 1 Legend for table 2 + 3

attributes								
type	**button**	**checkbox**	**checkboxes**	**date**	**fieldset**	**file**	**hidden**	**page**
accept	-	-	-	-	-	-	-	o
access							-	-
action	-	-	-	-	-	-	-	o
autocomplete_path	-	-	-	-	-	-	-	-
autocomplete_id	-	-	-	-	-	-	-	-
attached	o	o	o	o	o	o	-	o
class	o	o	o	o	o	o	-	o
collabsible	-	-	-	-	o	-	-	-
collapsed	-	-	-	-	o	-	-	-
cols	-	-	-	-	-	-	-	-
component_validation	-	o	o	o	o	o	o	o
default_value	o	o	o	o	o	-	o	-
description	o	o	o	o	o	o	-	-
disabled	o	o	o	o	-	o	-	-
element_id	o	o	o	o	o	o	o	r
element_type	r	r	r	r	r	r	r	-
encrypt	-	-	-	-	-	-	-	o
javascript	o	o	o	o	o	o	-	o

max_length	-	-	-	-	-	o	-	-
methode	-	-	-	-	-	-	-	r
multiple	-	-	-	-	-	-	-	-
name	-	-	-	-	-	-	-	o
options	-	-	-	-	-	-	-	-
order	a	a	a	a	a	a	a	a
field_prefix	o	o	o	o	-	o	-	o
required	-	o	o	o	o	o	-	-
resizable	-	-	-	-	o	-	-	-
size						o		
field_suffix	o	o	o	o	-	o	-	o
title	o	o	o	o	o	o	-	-
value	o	o	o	o	o	o	o	-
weight	o	o	o	o	o	o	-	-

Table 2 Form Component Type Definition 1

attributes									
type	**password**	**password_ confirm**	**radio**	**radios**	**submit**	**select**	**text**	**textarea**	**textfield**
accept	-	-	-	-	-	-	-	-	-
access									
action	-	-	-	-	-	-	-	-	-
autocomplete_path	-	-	-	-	-	-	-	-	-
autocomplete_id	-	-	-	-	-	-	-	-	-
attached	o	o	o	o	o	o	o	o	o
class	o	o	o	o	o	o	o	o	o
collabsible	-	-	-	-	-	-	-	-	-
collapsed	-	-	-	-	-	-	-	-	-
cols	-	-	-	-	-	-	-	o	-
component_validation	o	o	o	o	-	o	-	o	o
default_value	-	-	o	o	o	-	-	o	o
description	o	o	o	o	o	o	-	o	o
disabled	o	o	o	o	o	o	-	o	o
element_id	o	o	o	o	o	o	o	o	o
element_type	r	r	r	r	r	r	r	r	r
encrypt	-	-	-	-	-	-	-	-	-
javascript	o	o	o	o	o	o	-	o	o
max_length	o	o	-	-	-	-	-	o	o
methode	-	-	-	-	-	-	-	-	-
multiple	-	-	-	-	-	x	-	-	-
name	-	-	-	-	-	-	-	-	-
options	-	-	-	r	-	r	-	-	-
order	a	a	a	a	a	a	a	a	a
field_prefix	o	o	o	o	o	o	o	o	o
required	o	o	o	o	-	o	-	o	o
resizable	-	-	-	-	-	-	-	o	-
size	o	o				o		o	o
field_suffix	o	o	o	o	o	o	o	o	o
title	o	o	o	o	o	o	o	o	o
value	o	o	o	o	o	o	r	o	o
weight	o	o	o	o	o	o	o	o	o

Table 3 Form Component Type Definition 2

With the autocomplete function it is possible to get individual data from customized defined functions or WSDL interfaces. The function can used to build option lists or radio buttons.

4.3.2.2 The database table of forms

For the forms results a database table structure for the table “egov_form”, as follow:

- name – Is the name of the form.
- identifier – The internal id of the data set. It is a MD5-string build from the form name and the version number.
- path – The path to call the form.
- version – The version number of the form. On every changing of an activated form, which is used onetime, it is obligatory to build a new version number. This is necessary because the application should always to be able to rebuild a form like at the time of submission the form was build.
- owner – The user id of the owner of the form.
- active – Says if a form is callable. For blocked forms it is not possible to receive new data.
- publish_date – Is the publishing date of the form. Before this date a form should not transmitted.
- unpublish_date – Is the unpublishing date of the form. After this date a form should not transmitted. With saved form data it is possible to rebuild the form at any time.
- data – Contains the XML-structure of the form.

4.3.2.3 XML-structure of the forms

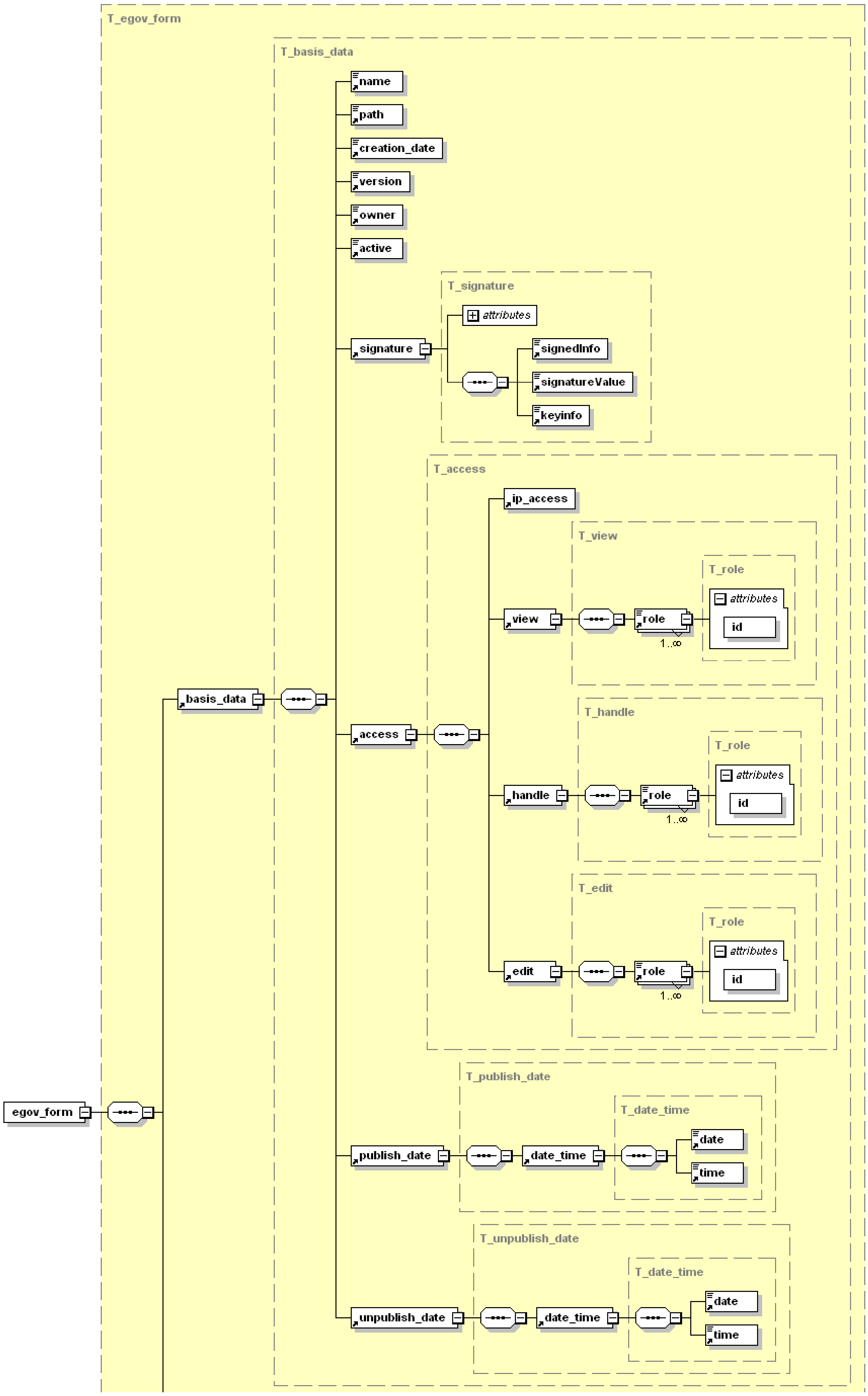

T_egov_form
T_basis_data
name
path
creation_date
version
owner
active
T_signature
attributes
signature
signedInfo
signatureValue
keyinfo
T_access
ip_access
T_view
T_role
attributes
view
role
id
1..∞
basis_data
access
T_handle
T_role
attributes
handle
role
id
1..∞
T_edit
T_role
attributes
edit
role
id
1..∞
T_publish_date
T_date_time
egov_form
date
publish_date
date_time
time
T_unpublish_date
T_date_time
date
unpublish_date
date_time
time

Figure 5 - Structure of an Easy Government-Form 1

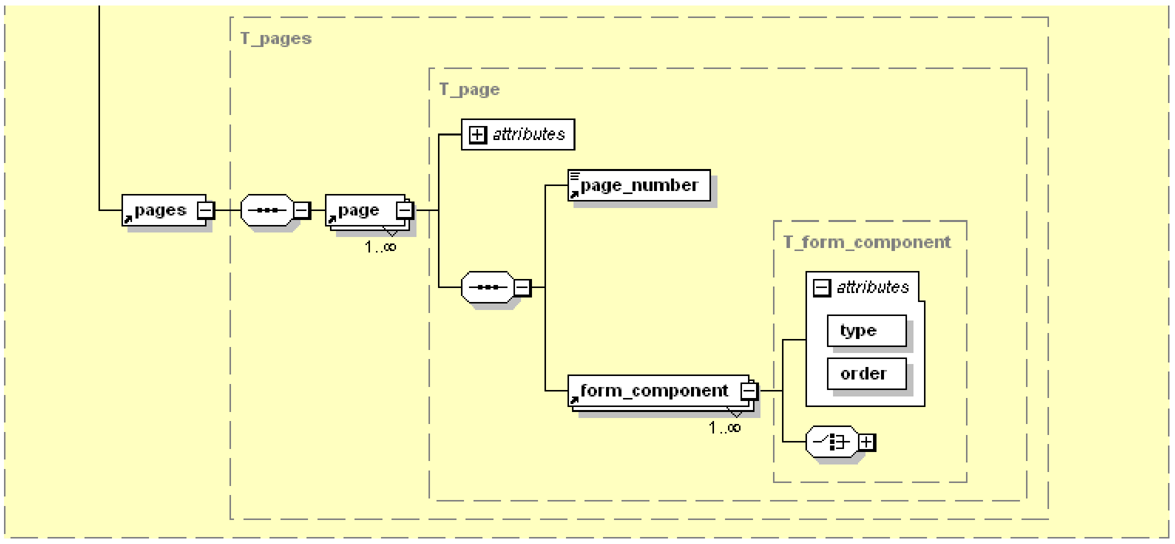


Figure 6 - Structure of an Easy Government--Form 2

```
<?xml version="1.0" encoding="UTF-8"?>
<egov_form xsi:noNamespaceSchemaLocation="egov_form.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <basis_data>
                   <name>text name</name>
                   <path>text</path>
                   <creation_date>text</creation_date>
                   <version>text</version>
                   <owner>text</owner>
                   <active>text</active>
                   <signature keyinfo="0" signatureValue="0" signedInfo="0">
                         <signedInfo>text</signedInfo>
                         <signatureValue>text</signatureValue>
                         <keyinfo>text</keyinfo>
                   </signature>
                   <access>
                         <ip_access/>
                         <view>
                                                 <role id="10">student wilhelmshaven</role>
                                                 <role id="11">student oldenburg</role>
                                                 <role id="12">student elsfleht</role>
                         </view>
                         <handle>
                                                 <role id="6">employee wilhelmshaven</role>
                                                 <role id="7">employee oldenburg</role>
                                                 <role id="8">employee elsfleth</role>
                         </handle>
                         <edit>
                                                 <role id="5">form admin</role>
                         </edit>
                   </access>
                   <publish_date>
                         <date_time>
                                <date>text</date>
                                <time>text</time>
                         </date_time>
                   </publish_date>
                   <unpublish_date>
                         <date_time>
                                <date>text</date>
                                <time>text</time>
                         </date_time>
                   </unpublish_date>
        </basis_data>
```

```
    <pages>
        <page order="1">
            <page_number>1</page_number>
            <form_component order="1" type="text">
                <element_type>text</element_type>
                <field_prefix> text </field_prefix>
                <field_suffix> text </field_suffix>
                <parents>text</parents>
                <title>text</title>
                <weight>0</weight>
                <value> text</value>
            </form_component>
            <form_component order="1" type="textfield">
                <class> text</class>
                <description> </description>
                <default_value> </default_value>
                <disabled>false</disabled>
                <element_type>textfield</element_type>
                <element_id>textfield_name</element_id>
                <field_prefix>Das ist der Prefix</field_prefix>
                <field_suffix>Das ist der Suffix</field_suffix>
                <max_length>200</max_length>
                <required>true</required>
                <size>60</size>
                <title>Textfield 1</title>
                <weight>0</weight>
                <value></value>
            </form_component>
            <form_component order="10" type="button">
                <disabled>false</disabled>
                <element_type>button</element_type>
                <field_prefix></field_prefix>
                <field_suffix></field_suffix>
                <name>submit_button</name>
                <parents>text</parents>
                <title>text</title>
                <weight>0</weight>
                <value>Senden</value>
                <uri>seite=2</uri>
            </form_component>
        </page>
        <page order="2">
            <page_number>2</page_number>
            <form_component order="1" type="text">
                <element_type>text</element_type>
                <default_value>Das ist ein Default-Text</default_value>
                <disabled>false</disabled>
                <value>Das ist ein VALUE-Wert</value>
                <theme>text</theme>
                <title>none</title>
                <weight>1</weight>
            </form_component>
            <form_component order="10" type="button">
                <disabled>false</disabled>
                <element_type>button</element_type>
                <field_prefix></field_prefix>
                <field_suffix></field_suffix>
                <name>submit_button</name>
                <parents>text</parents>
                <title>text</title>
                <weight>0</weight>
                <value>Senden</value>
                <uri>seite=3</uri>
            </form_component>
        </page>
    </pages>
</egov_form>
```

4.3.3 Form data structure of an Easy Government-Document

The form data structure follows the same structure as the structures of forms. The database only stores unencrypted data which are not data protection relevant.

For each incoming document will be a unique document ID (MD5 hash) generated. Each document will also receive a document hash, generated from the pure form data, the form name and a time stamp. This hash will append to the document and stored in the database, sent to the sender and receiver. This hash will detect all changes in the form data without questions. So it is possible to compare different documents and to prove the original document.

Another special feature is the "Part-Of"-function. This can link documents as required. The “Part-Of” is a document ID that refers to parent document. Such chains can be used to define circular letters or workflows. If the "Part-Of" function is used, the user automatically has a view permission on the parent document.

Also, files can be accepted.

4.3.3.1 Structure of the form data

Like the forms (4.3.2.1) divided the structure of the form data into two areas; some general data which required to finding the data set in the database quickly and the “dataset” as container for the received data and the electronical signature. The general data are stored unencrypted and so it is important to store only data’s which are not data protection relevant.

The transmitted data in the “dataset” container are generally encrypted to prevent a search in the contents of the database. Only after loading the data from the database, the data are decrypted in the application.
A special case is the ability to upload files. This data are instantly base64-encoded and subsequent encrypted, otherwise exists the danger of executable scripts.

The signature part is for the, in Germany used, eID-function and –data optimized. It is also possible to handle all other signature methods, in the simplest case an LDAP authentication (in Germany is not legally binding).

4.3.3.2 The database table of the form data

The database table is, as in the forms, kept simple and does not include any data protection relevant data.

- document_id – Is the unique name of the document. With the document ID is it possible to call the document at any time, if the requester is entitled.
- part_of_document_id – Is the document ID of the parent document (optional). For example, if an offer is made, can an attention or approval done in a second form which refers to the first.
- owner - The user id of the sender of the form.
- document_hash – Is a MD5-hash generated from the pure form data, the form name and a time stamp.
  Example: md5(‘<egov_document><form>abc</form>< request_time>2011-03-03 12:12:15</request_time><dataset><data page="1"><component>1</component><value>Fritz

Schneider</value></data><data page="1"> <component>2</component> <value>genehmigt</value> </data><data><component/><value/></data></dataset></ egov_document >‘) results “cacd8907435be3b74628e64d1dfc2029”.

If only one letter or character changes, such as the time set to 12:12:16, the result is another hash “8ca6e65545843a9c047078f2d29730bc”. So it is possible to compare different documents and to prove the original document.

- document – Containing the submitted form data in a XML-structure and encrypted mode. By encrypting is technically a search of the database more difficult (a requirement of the German law BDSG §4a, §9, §14 ff).
- document_type - Specifies the document type (HTML, PDF, etc.).
- active - Specifies whether the form data are valid. For example, a person entitled can declare a data set to be invalid, because the form data are incomplete or inaccurate. Also the form data are generally not active until they are signing.
- created – The receiving date of the form data. The date will be get from a special time server and is legally binding.

The signature will be stored as own data set in the ”egov_document”. So it is also more difficult to search or read the signature data. The siganture is not involved in the hash generation, because the option should be available that a entitled person can sign the data set. This feature allows people, who didn’t have a qualified signature or didn’t have the technical equipment, bring data into the system. Only special persons or groups are entitled to signing of minutes.

In the application is a special class for the system-side encrypting and decrypting. As encryption methodes will be provide ”none” (not allowed in Germany), base64 (only for testing) and a GnuPG implementation. On an interface can be added any other encryption algorithms. The encryption method and key must be specified individually during installation.

4.3.3.3 XML-structure of the form data

In the figure "Structure of an Easy-Government Document" is the structure of the XML file to identify.

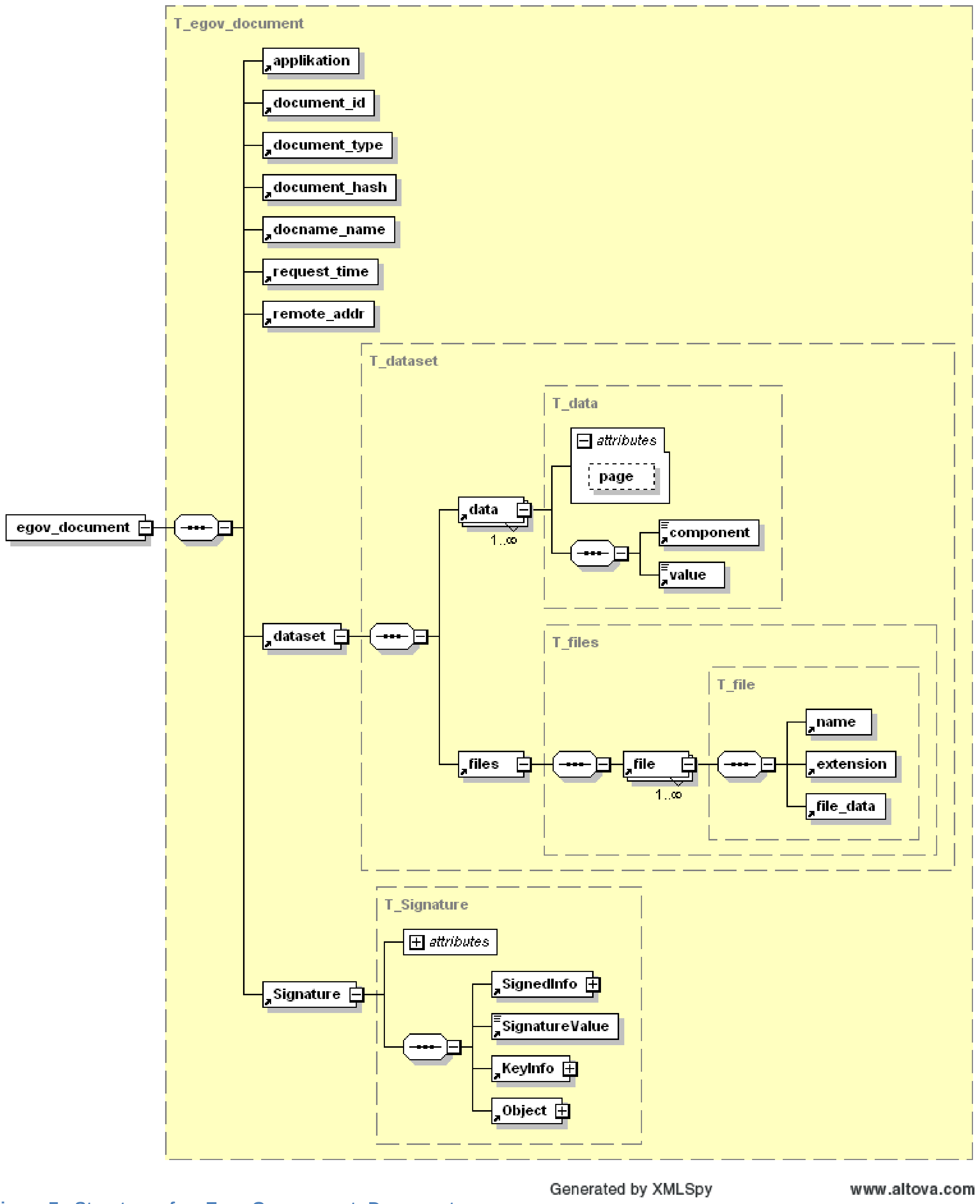


Figure 7 - Structure of an Easy Government--Document

```
<?xml version="1.0" encoding="UTF-8"?>
<egov_document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="egov_document.xsd">
		<applikation/>
		<document_id/>
		<document_type/>
		<document_hash/>
```

```
            <docname_name/>
            <request_time/>
            <remote_addr/>
            <dataset>
                 <data page="1">
                      <component>
                         1
                      </component>
                      <value>
                         Fritz Schneider
                      </value>
                 </data>
                 <data page="1">
                      <component>
                         2
                      </component>
                      <value>
                         genehmigt
                      </value>
                 </data>
                 <data>
                      <component/>
                      <value/>
                 </data>
            <files>
                 <file>
                      <name/>
                      <extension/>
                      <file_data/>
                 </file>
                 <file>
                      <name/>
                      <extension/>
                      <file_data/>
                 </file>
            </files>
</dataset>
            <Signature Id="">
                 <SignedInfo>
                      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000710">
                       </CanonicalizationMethod>
                      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa">
                       </SignatureMethod>
                      <Reference URI="http://www.w3.org/TR/xml-stylesheet/">
                                  <Transforms>
                                             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
                                             <Transform/>
                                  </Transforms>
                                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                  <DigestValue/>
                      </Reference>
                      <Reference URI="http://www.w3.org/TR/REC-xml-names/">
                                  <Transforms>
                                             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
                                  </Transforms>
                                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                  <DigestValue/>
                      </Reference>
                 </SignedInfo>
                 <SignatureValue>
            </SignatureValue>
                 <KeyInfo>
                      <KeyValue>
                                  <DSAKeyValue>
                                             <P>...</P>
                                             <Q>...</Q>
                                             <G>...</G>
                                             <Y>...</Y>
                                  </DSAKeyValue>
                      </KeyValue>
            </KeyInfo>
            <Object>
                <SignatureProperties>
                                 <SignatureProperty Target="">
                                                  <timestamp/>
                                 </SignatureProperty>
                <SignatureProperties>
            </Object>
        </Signature>
</egov_document>
```