IST PROJECT 2001-35399

A Governmental Knowledge-based Platform for Public Sector Online Services

Project Number:	IST-2001-35399
Project Title:	A Governmental Knowledge-based Platform for Public Sector Online Services
Deliverable Type:	Public

Deliverable Number:	D82
Contractual Date of Delivery:	31-8-2003
Actual Date of Delivery:	17-10-2003
Title of Deliverable:	User's guide
WP contributing to the Deliverable:	WP8
Nature of the Deliverable:	Report
Editor(s):	Tomás Pariente Lobo, Pablo Fernández Pardo
Author(s):	Tomás Pariente Lobo, Pablo Fernadez Pardo, Stelios Gorilas, Costas Vassilakis, Akrivi Katifori, Anna Charissi, George Lepouras, Nick Adams, John Fraser, Ann Mackintosh, Vassilis Stoumpos, Pavlos Kattoulas

Abstract: This deliverable constitutes the user's guide of the tool developed in WP5 and WP6. It presents how to use the SmartGov Front-End developing environment and the way to deliver and maintaining services.

Project funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

The SmartGov Consortium consists of:

Partner's Name	Acronym	Role	Country
University of Athens	UoA	Project Coordinator	Greece
T-Systems Nova	TNB	Partner	Germany
Indra Sistemas S.A.	Indra	Partner	Spain
Archetypon S.A.	ARC	Partner	Greece
Napier University	NU	Partner	UK
General Secretariat for Information Systems	GSIS	Partner	Greece
City of Edinburgh Council	CEC	Partner	UK

Table of Contents

Table of Figures

List of Acronyms

Acronym	Explanation
API	Application Programming Interface
BEAN	Java Bean
DSN	Data source name
JDBC	Java Database Connectivity
JSP	Java Server Page
KU	Knowledge unit
LDAP	Lightweight Directory Access Protocol
MVC	Model-View-Controller
PA	Public Authorities
RDBMS	Relational Database Management System
RUP	Rational Unified Process
SGA	SmartGov agent
TS	Transaction service
TSE	Transaction service element
TSE Group	Group of transaction service element
UML	Unified Modeling Language
WAP	Wireless Application Protocol
WML	Wireless Markup Language
XHTML	eXtensible Hypertext Markup Language
XML	Extensible Markup Language
XSLT	Extensible Style sheet Language Template

1 Introduction

The aim of the SmartGov User's Guide is to provide a reference help about how to use the SmartGov platform to desing and deliver e-forms based services. The implementation details and the platform overview are available in the deliverables D51-61 [D51-61], D52 [D52] and D62 [D62].

The document firstly provides an overview about what the users will be able to perform with the SmartGov tool and then introduces the different tools included for the platform.

2 Overview

2.1 Introduction

Some of the parts included in this section also exist within the previous deliverables (mostly in D41 and D51-D61), but are also included here for completeness purposes.

2.2 Platform client requirements

The SmartGov platform requires several client requirements related with hardware and software.

2.2.1 Hardware

No special requirements of hardware are required. A common winows-based PC is enough to run the SmartGov client-side. As an example, the minimum requirements could be:

- PC Pentium 3, 1 GHs or higher
- HDD 2GB or higher
- RAM 64MB or higher

2.2.2 Software

- Browser: Microsoft Internet Explorer 5.5 or superior (recommended)
- Configuration: In Tools/Internet options/General/Configuration, for option "Check if there are new versions of the saved pages" check "Every time the page is visited".

NOTE: Important. If the Configuration of the browser is not set, the Front-end tool could present problems with the cache of the pages, resulting in problems with the data consistency.

2.3 SmartGov groups of users

Five user groups or roles have been identified. These groups are: Managers, Domain Experts, Information Technology Staff (IT Staff), Administrators and End Users.

The roles regarding the Transaction Service Lifecycle are shown in the Figure 1.

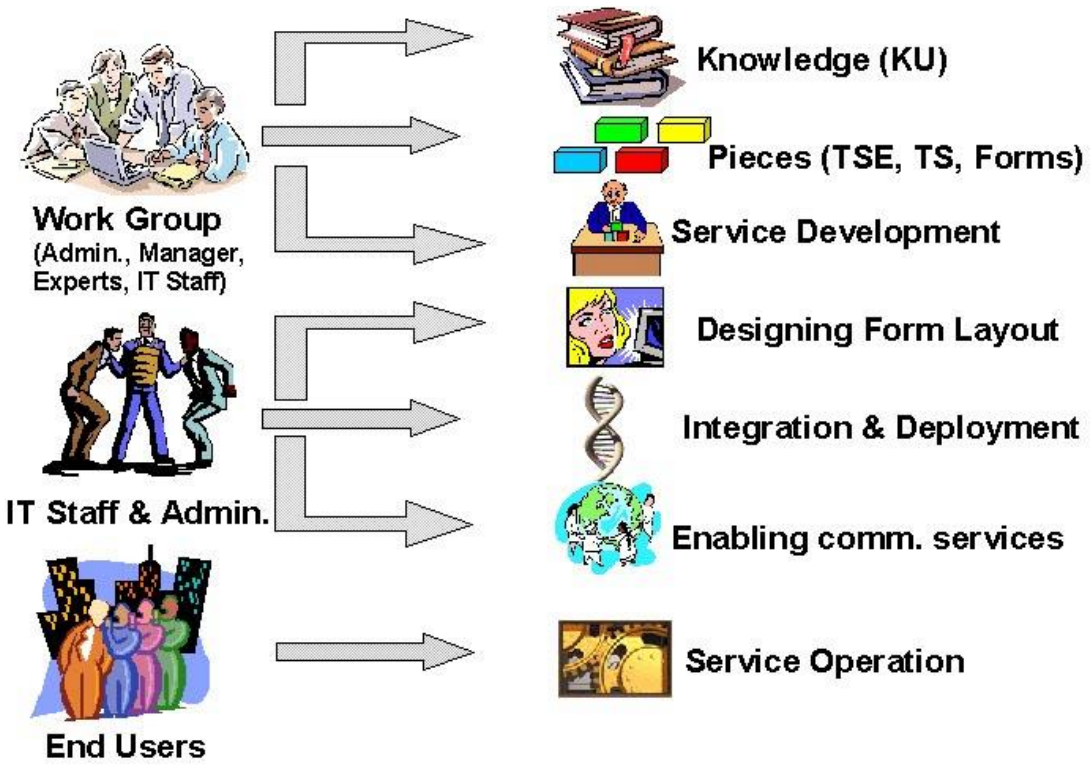


Figure 1 – SmartGov user groups and their main tasks

- Managers: The managers are responsible for organising and supervising public services. They make decisions about the implementation of new services or the alteration of existing ones. In order to accomplish this task, they need to have a strategic view of the provision of services. Managers are able to decide about future changes in the service or the creation of a new one. Usually, there is more than one manager in the same Public Authority, who wishes to have access to the same data and statistics.
- Domain Experts: The domain experts possess the necessary background knowledge for the design and the implementation of a public service. This knowledge includes the legislation that a service is based on, that is laws, processes, directives, prerequisites and so on. Domain experts play a consultative role to the managers for the design, evaluation and possible alterations of public services. To this end, they need to define and obtain statistics and metrics. They design the interface of the service and the structure of the form, which is what service users will fill in. They attach their knowledge about legislation, supporting procedures or required documents to the form elements. They define validation checks, which are not limited to data type constraints, but also include inter–element relations that should be satisfied within the form or even relations that

should hold between different forms. Finally, domain experts provide end users with accompanying manuals, instructions and sets of examples, to help them use the service. It is possible that more than one domain expert works for the implementation of the same service, while each domain expert may participate in the lifecycle of more than one service, when his/her expertise is needed.

During the development of an e–service, the domain experts may have to collaborate with the IT staff to communicate to them their domain knowledge. Collaboration has to take place when the tasks to be performed require higher technical skills than the domain experts possess, and when the links to the installed IT systems or third party systems have to be established.

- IT Staff: The IT Staff possess the necessary technological knowledge for the development of an electronic public service. They design the system from scratch, defining system architecture, database schema, user interface and functionality. They also provide the necessary interfaces for data exchange between the electronic service platform and the back-end systems. During the life cycle of the service they have to collaborate with the domain experts to integrate the domain knowledge, which is of vital importance, to the application. At the same time they play a consultative role to the managers and the domain experts with respect to the technological aspects of the e-service. In addition, they need to define and obtain technical level statistics and metrics to acquire valuable insight about the efficiency of the system. Furthermore, they are responsible of the maintenance of the e-service. They have to handle omissions and problems that may occur in the electronic services, which could be for example programming errors, alterations caused by changes of the supporting legislation, modifications suggested by the managers or the domain experts.
- Administrators: The administrators support the platform users, which are mainly Public Administrators (managers, domain experts and IT staff) and indirectly the end users. They help them to familiarize themselves with the environment of the e-service and cope with possible problems that may occur. This support is offered via e-mail or telephone and may produce helpful feedback to the IT staff about the usability of the e-service. They are also responsible for the management of user accounts, the integrity of the data (back up functions etc.) and the security of the system. One of their tasks is also the specification of log files, which contribute not only to

the accountability and non-repudiation but also to the observation of the system performance and the production of qualitative measurements such as system usability, identification of common errors made by the users etc.

- End User: The end users are the citizens or enterprises that make use of the service. These are not users of the SmartGov tool in the proper sense, because they are only going to use the result of the platform and not the SmartGov platform itself.

2.4 The Transaction Service Lifecycle

Within the lifecycle of a transaction service (i.e. a service that includes filling and submission of forms, whose data are then processed by an organisational back-end system), the following phases may be identified:

1. The manager decides to implement a new service
2. The manager creates a working group, consisting of domain experts, IT staff, managers and service workers.
3. The group produces the service requirements
4. The group derives the service specifications (process model)
5. The group develops the transaction service elements
    a. Forms (domain experts and possibly IT staff)
    b. KUs: Knowledge Units (mainly domain experts)
    c. TSEs: Transaction Service Elements (domain experts and IT staff)
    d. Validation checks (Domain experts and IT staff)
    e. Links to back-end systems (mainly administrators)
    f. Managerial statistics (Managers)
    g. IT-related statistics (IT staff, domain experts).
6. IT staff and administrators integrate elements
7. IT staff and administrators test the new service
8. The group evaluates the new service
9. The service is deployed
10. Service operation and maintenance
11. Collection of feedback
12. Service improvement
13. Discontinuation of a service

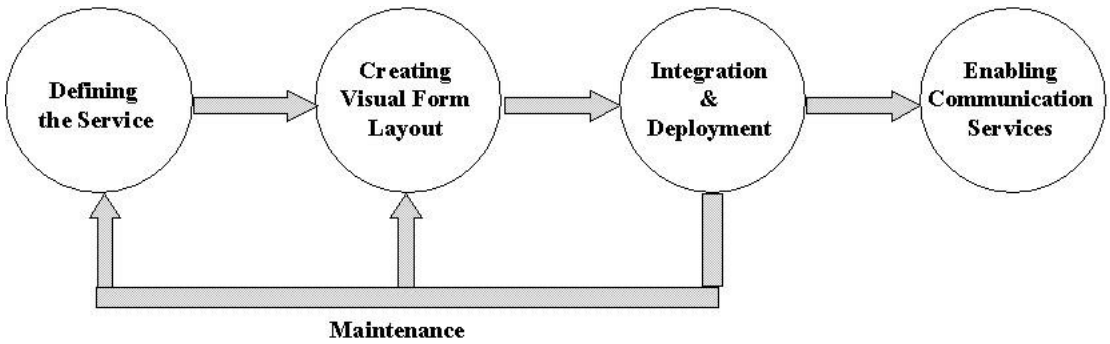


Figure 2 – SmartGov service life-cycle phases road map

Not all of these phases are supported by the SmartGov platform. In particular, phases 1-2 involve managerial actions, such as feasibility studies and human resource management. Phase 2 is partially covered Within phases 3 and 4 the initial definitions and documentation (KUs) are collected and entered in the SmartGov platform. The SmartGov platform comes into full play during phases 5 and 6, where the various transaction service components are developed and integrated. After the integration step in phase 6, the electronic service is instantiated and installed on an internally accessible server for testing and evaluation. These phases may trigger further actions within phase 5, producing new versions of the electronic service, which are again tested and evaluated internally in phases 7 and 8 respectively. The SmartGov platform will not provide tools for service testing and evaluation, but is responsible for generating the instantiated service version.

When the service has reached a satisfactory state, it is deployed on a publicly accessible server (phase 9) so that it can be delivered to the end-users. Service deployment is similar to installing the service on the test environment of phases 7 and 8, with the only difference being the accessibility (and possibly the scale) of the server.

Once deployed a service enters the operation and maintenance phase (10), during which end users access the service. Throughout the operation and maintenance phase, feedback is collected both by end users and via the statistics collection mechanisms of the SmartGov platform (phase 11), which will be exploited for service improvement (phase 12). In these phases, the SmartGov platform offers support for statistics collection, user account management and database backup and recovery.

Finally, if a service becomes obsolete (for example, due to changes in legislation, an expiration deadline, or even because it has not been proven to be popular enough to justify its delivery and maintenance costs), it can be discontinued. In

such cases, delivery of the service through the dissemination platform should cease and, depending on (a) the possibility that the service will go live again and (b) organisational policy, it might be required that the SmartGov platform objects created specifically for this service will be purged.

3 The SmartGov front-end tool

3.1 Introduction

The SmartGov front-end is responsible for providing a development environment for managing knowledge, developing and populating the design of the elements that formulate the different transaction services. The repository populated using the SmartGov front end will be the entry point to the Integrator. This development environment is available to the actors directly involved in the lifecycle of electronic transaction services, namely domain experts, IT staff and managers. The actors employ the SmartGov front end to populate, query and modify the knowledge and transaction services repositories.

Firstly this chapter explains the process to install and configure the Front-end, and later it gives an explanation about how the front-end works. The front-end is a web-based application with a portal-like appearance that allows actors to easily define and query the services. In this section, some common features and the navigation methods are explained.

Once introduced the basic functionality, the different areas of the tool will be described: users management, knowledge management and Service elements management.

The figure 3 shows a quick view of the main functionality covered by the SmartGov Front-End tool.

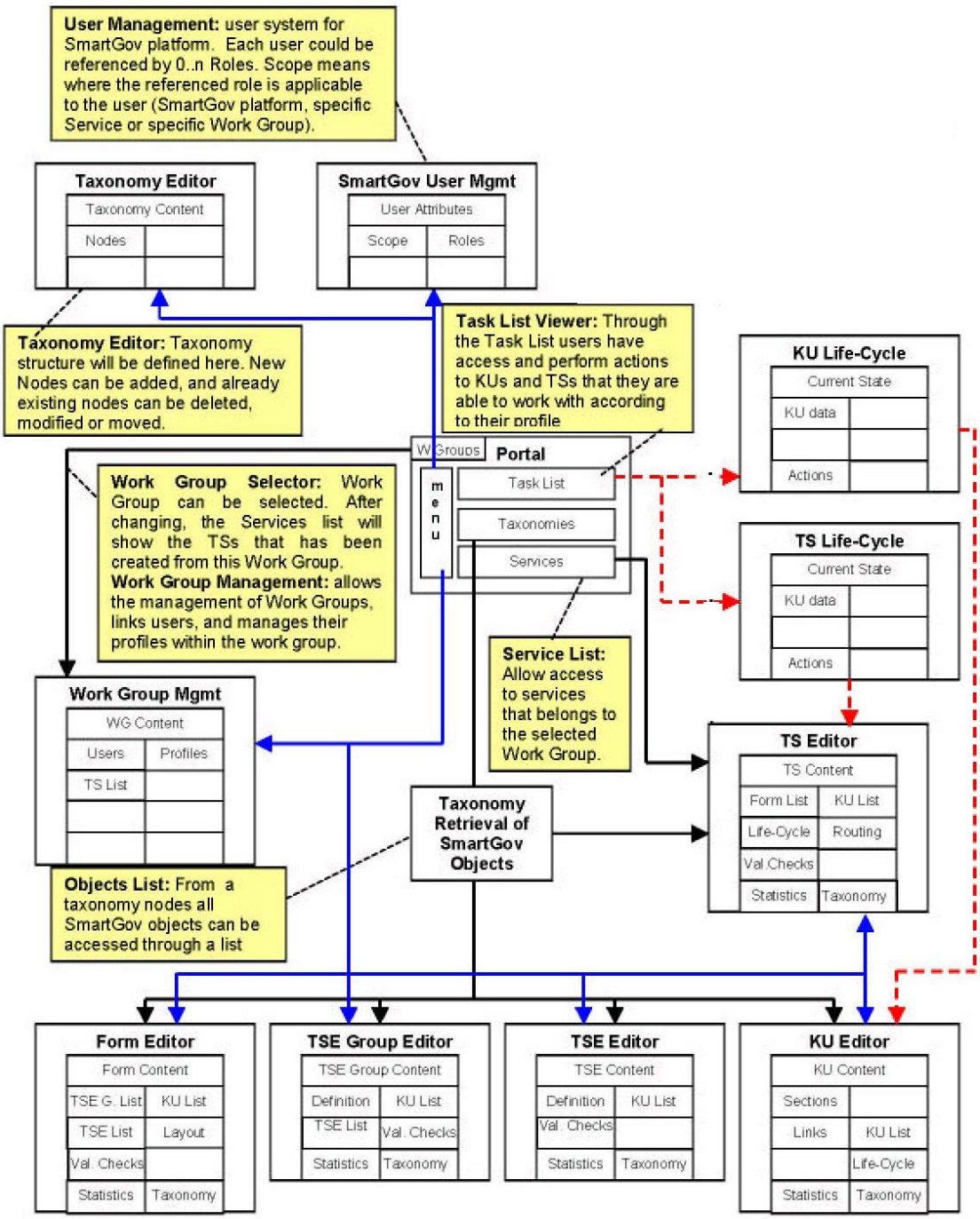


Figure 3 – Front-End road map

3.2 Installation process

3.2.1 Requirements

3.2.1.1 Hardware

CPU: Pentium III, 600 MHz

RAM: > 384 Mb

HDD: > 100Mb free space (depends on the number of hosted services)

3.2.1.2 Software

OS: Windows 2000, Service Pack 3+

Servlet engine: Tomcat 4.1+

JDK: Java2 SE 1.4.2+

RDBMS: Microsoft SQL Server 2000 or Windows MySql 4.x

3.2.2 Environment setup

3.2.2.1 Installing and configuring the data components

The first required step to setup the Front-end involves creating the databases and the XML Repository.

The Front-end requires three databases:

- The user-roles database, which keep all the data related with the workgroups defined in the tool, the users that belong to these groups and the role or roles of these users.
- The outer user database, which keep the basic data of users. This database only contains a table, which may be created in previous database. However, the script for this table is isolated from the other ones because the platform is designed to use outer user systems -LDAP, database system…- (always requiring an extra time to develop the corresponding connectors) already existing in the organization where the platform is installed, and, in that case, this table is not required.
- The Xml store database, which will be shared with the integrator and that will be the way to interoperate between both components.

To create these databases, the script contained in the Appendix C. must be executed. The corresponding to the two first databases create all the structure, but the last one only creates the database and grants access to users, whereas the whole table structure and prepopulation of data will be made later, as described in paragraph 3.2.2.3.

3.2.2.2 Installing the front-end

Once the required databases have been created, the web application will be installed in the Servlet container. Therefore, the last version of the SmartGov.zip file must be unzipped in the tomcat/webapps folder. Once this process has finished, a folder called SmartGov must appear in the webapps folder.

After installing the web application, the next step is changing the configuration files (smartgov.properties), to fit the characteristics of the environment where the platform is being installed. This are the relevant properties defined in this configuration file:

Property name	Most used values	Description
smartgov.databaseType	• MySql • MsSqlServer	RDBMS used.
smartgov.bdUsers.user	db_user	User with granted access to user and roles database.
smartgov.bdUsers.password	egov	Password of the user specified in the previous property.
smartgov.bdUsers.driverClass	• org.gjt.mm.mysql.Driver • com.microsoft.jdbc.sqlserver.SQLServerDriver	JDBC Driver to access the user and roles database.
smartgov.bdUsers.url	• jdbc:mysql://<*db_host_name*>/<db_name> • jdbc:microsoft:sqlserver://<*db_host_name*>;DatabaseName=<db_name>;selectMethod=cursor	Connection string to user and roles database
smartgov.bdOuterUsers.user	db_user	User with granted access to outer user database.
smartgov.bdOuterUsers.password	egov	Password of the user specified in the previous property.
smartgov.bdOuterUsers.driverClass	• org.gjt.mm.mysql.Driver • com.microsoft.jdbc.sqlserver.SQLServerDriver	JDBC Driver to access the outer user database.
smartgov.bdOuterUsers.url	• jdbc:mysql://<*db_host_name*>/<db_name> • jdbc:microsoft:sqlserver://<*db_host_name*>;DatabaseName=<db_name>;selectMethod=cursor	Connection string to outer user database
smartgov.bdXmlRepository.user	xmlstore_user	User with granted access to XML Repository database.
smartgov.bdXmlRepository.password	egov	Password of the user specified in the previous property.
smartgov.bdXmlRepository.driverClass	• org.gjt.mm.mysql.Driver • com.microsoft.jdbc.sqlserver.SQLServerDriver	JDBC Driver to access the XML Repository database.
smartgov.bdXmlRepository.url	• jdbc:mysql://<*db_host_name*>/<db_name> • jdbc:microsoft:sqlserver://<*db_host_name*>;DatabaseName=<db_name>;selectMethod=cursor	Connection string to XML Repository database.
smartgov.xmlRepository.serverName	madarrgesdoc03	Name of the server where the XML Repository database is hosted.
smartgov.xmlRepository.portNumber	• 3306 (for MySQL) • 1433 (for Ms Sql Server)	Port to access the XML Repository database.

Property name	Most used values	Description
smartgov.xmlRepository.databaseName	xmlstore	Name of the XML Repository database.
smartgov.xmlRepository.user	xmlstore_user	User with all the privileges granted on XML Repository database
smartgov.xmlRepository.password	egov	Password of this user
smartgov.xmlRepository.URL	• jdbc:mysql://<*db_host_name*>/<db_name> • jdbc:microsoft:sqlserver://<*db_host_name*>;DatabaseName=<db_name>;selectMethod=cursor	Connection string to the XML Repository database.
smartgov.xmlRepository.selectMethod	cursor	(Only required if Ms. SqlServer is used).
smartgov.xmlRepository.com.archetypon.xml.store.datasource.provider	• com.mysql.jdbc.jdbc2.optional.MysqlDataSource • com.microsoft.jdbcx.sqlserver.SQLServerDataSource	Datasource class to access the XML Repository database.
smartgov.xmlRepository.com.archetypon.xml.store.dbms	• MySQL 4.x • Microsoft SQL Server 2000	RDBMS type
smartgov.xmlRepository.com.archetypon.xml.store.datasource.classpath	..\\webapps\\SmartGov\\WEB-INF\\lib\\mysql-connector-java-3.0.7-stable-bin.jar	Datasource class path
smartgov.availableLocales	es,en,el	Available locales for the Front-end users. The order is important, because the first one is the default.

There are two pre-created configuration files, distributed with the installation in the SmartGov/WEB-INF folder: smartgov.properties.mysql and smartgov.properties.sqlserver.

The corresponding file to the RDBMS used for the platform must be renamed to smartgov.properties file, and then it must be modified, according to the following:

- Changing the name of the server hosting the database in the following properties:
  - ***smartgov.bdUsers.url***
  - ***smartgov.bdOuterUsers.url***
  - ***smartgov.bdXmlRepository.url***
  - ***smartgov.xmlRepository.serverName***
  - ***smartgov.xmlRepository.URL***
- Changing the value of the property ***smartgov.xmlRepository.com.archetypon.xml.store.datasource.classpath*** to the absolute path where the JBDC driver is located (jar file or a directory where the driver is unzipped).

Also some changes are required to configure and populate the Xml Store with the initial elements that are included in the platform. To configure access to the repository is required an update in the integrator.X.properties (where X is the RDBMS type), located in the webapps/SmartGov/scripts folder. This file must be updated to reflect the name of the server where the database is hosted, as described in the Integrator guide, in section 4.3.2.6. If the Integrator has been already installed, the integrator.properties file can be reused.

3.2.2.3 Configuring and populating the XML Repository

The "user and roles" and "outer users" databases are prepopulated with a set of users in the scripts used to create them. However the Front-end uses a more sophisticated way to create all its structure and to load elements in the platform.

Firstly, to create the Xml Repository, with the required structure, there is a file in the folder "webapps/SmartGov/scripts" called "createXmlRepository.BAT". This .bat file uses an XML file ("exampleOfRepositoryConf.xml" in the same folder) to create the repository and all the indexes required by the Front-end and the Integrator. Therefore, once executed this file, it's not necessary to follow the steps described in paragraph 4.3.9, because the XML Repository for the Integrator will be already created. Once this program has been executed, it can be checked in the XML Repository database, which was empty, that some tables

have been created and populated. Therefore, the XML Repository is ready to be used.

Please note that this program uses the smartgov.properties file previously described, so this file must be appropriately modified before proceeding to configure the repository.

Although with the previous step the platform is ready to be used, it is better, in order to take advantage of all the capabilities of the SmartGov Front-end and to make easier its use to the users, to load some pre-created elements in the repository. The usual elements to load are pre-created Taxonomies and KUs, which enrich the Front-end, providing support and knowledge to users. Also some already created services may be loaded, in order to provide examples of the use of the different elements.

To load the selected elements, it would be used the Document Crawler, which can be launched executing "Document Crawler.BAT" in the folder "webapps/SmartGov/scripts". The usage of this utility is described in paragraph 4.3.10. Please note that, as properties file, can be selected the already configured for the integrator if it has been already installed, or the file modified in the previous paragraph ("integrator.X.properties") which is located in the same folder that the .bat file.

With the Document crawler can be loaded in the system the taxonomy and the KUs available in the distribution of the Front-end, and also any pre-created service, that may act as example for future developments.

3.2.2.4 Pre-created users structure

The DB scripts included in Appendix C. to create users databases create a set of users, all belonging to group "Test", with the following characteristics:

User Id	Password	Role	Roles in group Test
user_expert	user_expert	Expert	TS editor
user_manager	user_manager	Manager	KU editor, TS editor
user_staff	user_staff	IT Staff	KU editor, TS editor
administrator	administrator	Administrator	KU editor, TS editor

3.3 Structure of the front-end

3.3.1 Common Features

3.3.1.1 Using a Web application

Some very basic concepts that a user must remember when using a web application:

- **URLs**

  When a user works with a web application -filling fields, navigating pages, opening links- the URL shown in the Address bar of the web browser is changing almost with every action. It is not recommended to "bookmark" a different URL from the URL defined to access the logon page, because the use of these "temporal" URLs may cause errors and unexpected behaviour.

- **Wait until the page is loaded**

  When a button or a link is clicked, the user must wait till the page is completely loaded, because if not, the loaded page could not work correctly.

- **Using the Tab**

  The tab key can be used to move the "focus" (where the cursor is located) from one field to other, or to links or other elements defined in a page. This is very useful when filling a form.

- **Using the Mouse**

  The mouse can be used in several ways when navigating a web page:

  - To move to other page: left-clicking once in a link.
  - To set the focus in one field: left-clicking once on the field.
  - To move around the page: Using the bars at right and below the page, as in any other "Windows" application.

3.3.1.2 Managing sections

Many of the pages in the Front-end application are divided in sections. These sections have a double purpose: organize the data, and make easier viewing this data in screen.

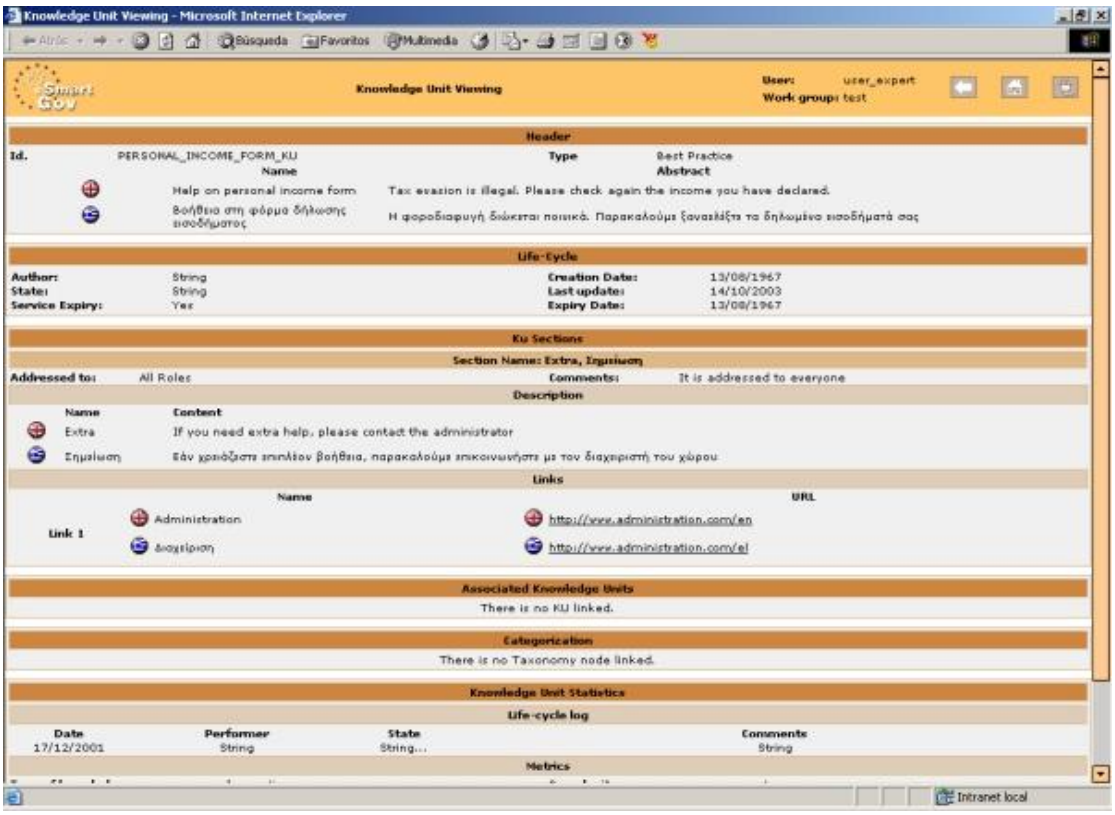


Figure 4 - Sections in a Front-end page (Header, Life-cycle…)

In edition pages (see Figure 5), the user can interact with these sections, collapsing or expanding them, clicking in the triangular icon located in the left of the title bar of the section.

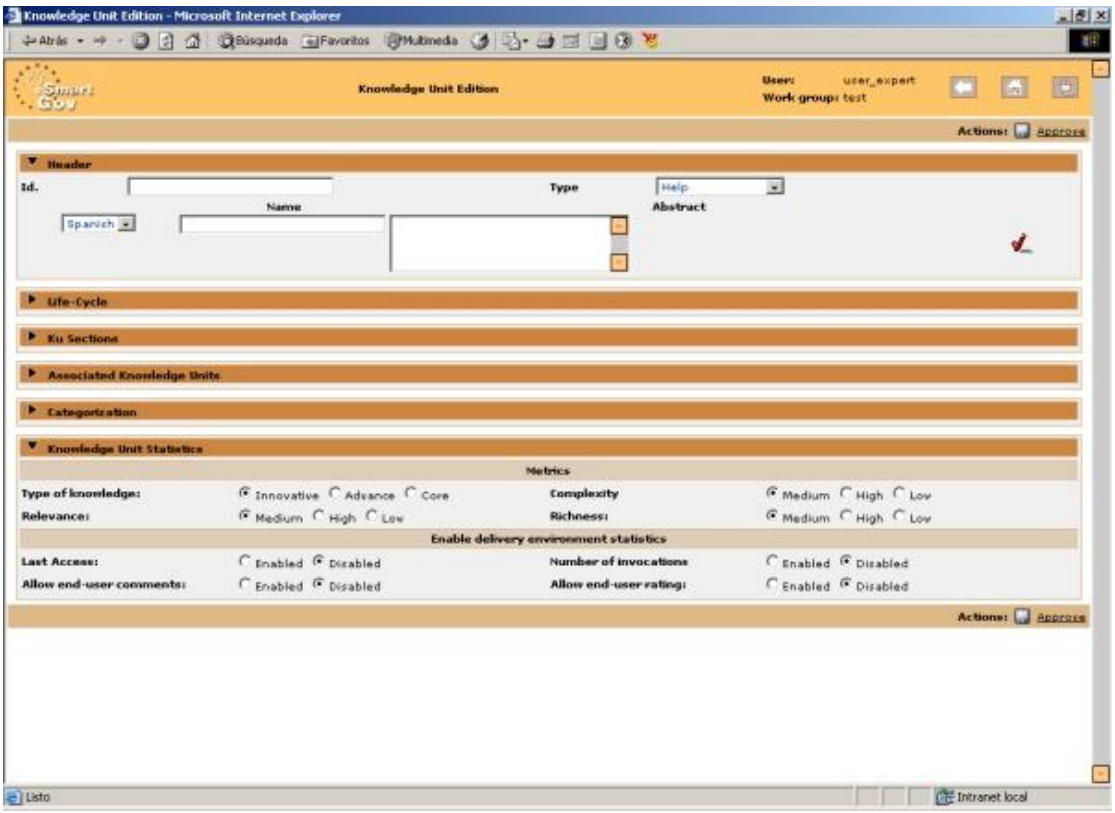


Figure 5 - Sections in an editable page (Header and Knowledge Unit Statistics are expanded, the other sections are collapsed)

3.3.1.3 Managing multi-lingual tables

The SmartGov platform is multi-lingual, not only allowing the user in which language they want to see the pages, but also allowing, in many fields in the different objects, to specify different values for different locations.

To manage these multilingual fields, it has been defined and structure (shown in Figure), which has the following elements:

- Row to add new values: at the bottom of the table, there is a row with a list to select the language to define (from those already undefined), and one or more fields to define the values. Once the fields have been filled, their values are added to the table clicking in the Add button on the right side.
- Already inserted values: they are listed beside a flag representing the language to which they belong. In the example figure, "Ku Name" and "Ku abstract" are the values for English, and "Nombre de la Ku" and "Abstract de la Ku" the values for Spanish.
- Actions to modify or delete already inserted values: in the right side of the Multilingual table there are two icons: a yellow ball (edition) and a trash (deletion).

- Edit: if the Edition icon is clicked, the current values for the corresponding language are moved to the text fields below, so that they can be modified. Once modified, the values are updated to the table click in the Add icon, on the right side.
- Delete: If the deletion icon is clicked, the values corresponding to the selected row are deleted.

Figure 6 – Multilingual table (the selected area)

The available locales are defined in the properties file smartgov.properties, described in the section 3.2.2.2 of this document.

3.3.1.4 Managing other tables

When the application requires the management of a list of values, the table used to this is very similar to the multilingual table:

- Row to add new values: at the bottom of the table, there is a row with a one or more fields to define the values. Once the fields have been filled, their values are added to the table clicking in the Add button on the right side.
- Already inserted values: they are listed in the table.
- Actions to modify or delete already inserted values: in the right side of the table there are two icons: a yellow ball (edition) and a trash (deletion).
    - Edit: if the Edition icon is clicked, the values in the corresponding row are moved to the text fields below, so that they can be modified. Once modified, the values are updated to the table click in the Add icon, on the right side.
    - Delete: If the deletion icon is clicked, the values corresponding to the clicked row are deleted.

Figure 7 – Multiple values table (the selected area)

3.3.1.5 Pagination

There are several list of elements in the Front-end application. These lists include pagination functionality when is considered necessary, so that the user can navigate through a list too long to be fully shown in a screen.

When a list contains more than the number of elements that can be shown in one list (defined in configuration, usually 5 elements), then pagination links are added at the bottom of the page, so that the user can move to the next or the previous part of the list.

Figure 8 – List with pagination

3.3.1.6 Actions applied to a SmartGov object

- Action bar:

  The action bar is an element located at the top and at the bottom of each element (see Figure 9). All the available actions to perform over this object are situated in the bar: save, delete…

Figure 9 – Action bar location

Figure 10 – Action bar example

- Save object (floppy disk icon):

  This element enables the user to update the changes made in the element currently in edition. When the element currently in edition is an auxiliary

element (Ku Section, Method, Validation Rule) this icon don't save the changes to the XML Repository, but it updates the changes in the main element to which the auxiliary element belongs (Ku, TS, TSE…). The changes are only saved to the repository when the main element is saved.

- Delete object (trash icon):
  This item deletes the current element. If the item that we are editing is an auxiliary one, this deletion is not definitive unless the main object is saved.
- Approve object (life-cycle of TS and KU):
  This action causes the current element to be saved an changes its current status. Please refer to life-cycle of Ku and TS (3.5.1 and 3.6.1) for further details.
- Reject object (life-cycle of TS and KU):
  This action causes the current element to be saved an changes its current status. Please refer to life-cycle of Ku and TS (3.5.1 and 3.6.1) for further details.
- Reopen object (life-cycle of TS and KU):
  This action causes the current element to be saved an changes its current status. Please refer to life-cycle of Ku and TS (3.5.1 and 3.6.1) for further details.

3.3.1.7 Associate knowledge units to the SmartGov elements

Almost every SmartGov element (TSs, Forms, Instantiated TSE and TSE Groups, Generic TSEs and TSE Groups, and even other KU) can be linked to a KU. Thus, the knowledge related with the element can be linked to it.

The part of the application to manage these links is always the same (see figure 11). There is a list with the already linked KUs (that can be unlinked clicking in the trash icon), and below a link to add an already existing KU to the list. It is important to notice that it is necessary to create the KU before linking it.

Figure 11 – List of linked KUs

When the link to add new KUs to the list is clicked, the page to select KUs is shown (see figure 12). This page enables user to select KUs using the list with the last modified KUs, searching through exiting taxonomies, and even searching

through the Ku Id. In this field it's allowed the use of wildcards (The "%" matches any number of occurrences of any character).

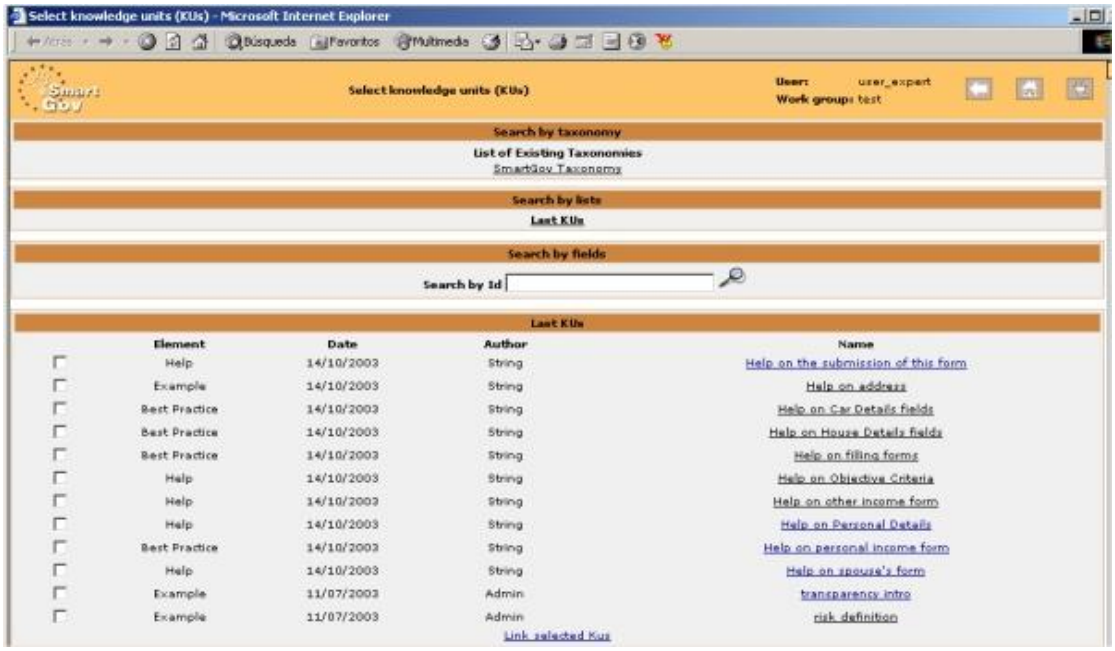


Figure 12 – Select KUs page

3.3.1.8 Categorization of the SmartGov elements using taxonomy nodes

As the KUs, almost every element of the SmartGov platform can be linked to a taxonomy node. Thus, the user is able to categorize and organize these elements, making easier their retrieval and future search.

In each element there is a section when the user can view the taxonomy nodes the element is already related to (see figure 13) and can detach these nodes and select others. Later this element may be retrieved navigating through the taxonomies and selecting one of the linked nodes.

Figure 13 – Linked taxonomy nodes list

To select a node to link, a new page is shown (see figure 14), with a list of all the taxonomy in the top and the selected taxonomy below. The user can select the taxonomy that the nodes belong to, and then navigate through the taxonomy, selecting the nodes to link.

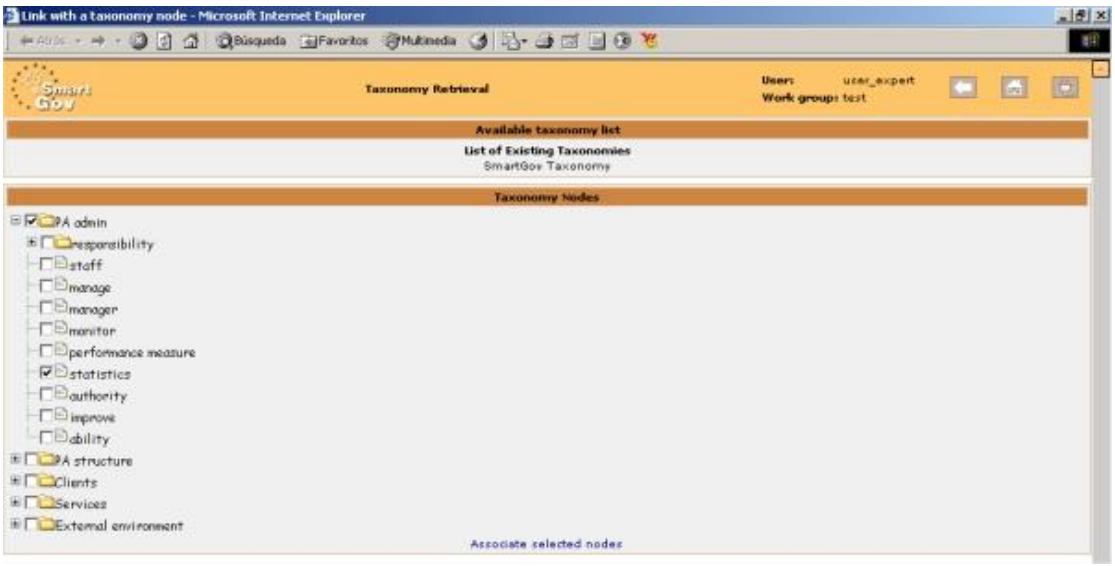


Figure 14 – Select taxonomy nodes page

3.3.1.9 Uploading files

The SmartGov Front-end includes a very simple document manager, which enables user to upload files. These files are uploaded to the web server where the application is hosted, so they can be used later.

Files can be uploaded in different parts of the Front-end:

- Layout XHTML files in forms
- KU attachments
- Native code methods

The mechanism to upload the file is always more or less the same: an auxiliary windows is opened (see figure 15) where the user can select a file and then the file is uploaded and its data included in the element that the user was editing. This process will be analysed in detail for each case during this manual.

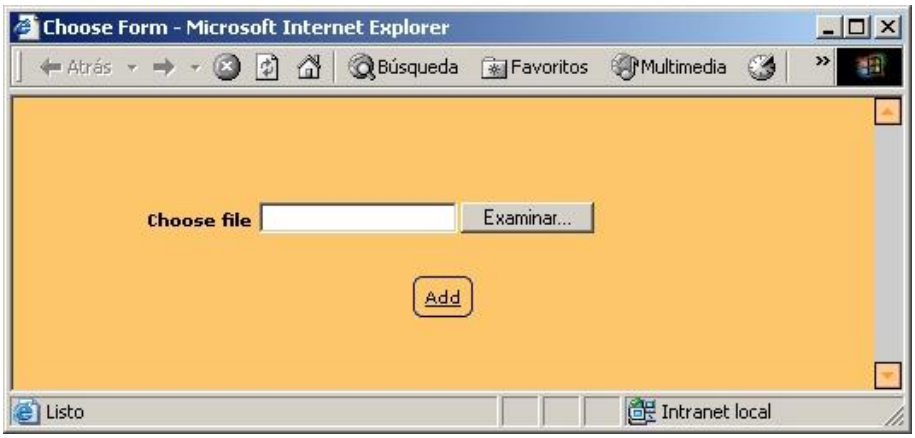


Figure 15 – Upload file page

3.3.2 Login page

To enter to SmartGov Front-end tool, the following URL should be entered:

Login page: http://<host_name>:<port>/SmartGov/logon.jsp

Where <host_name> is the name of the host that should be provided by the SmartGov administrator in every SmartGov Front-end installation.

Once the correct URL is entered, the SmartGov login page is shown.

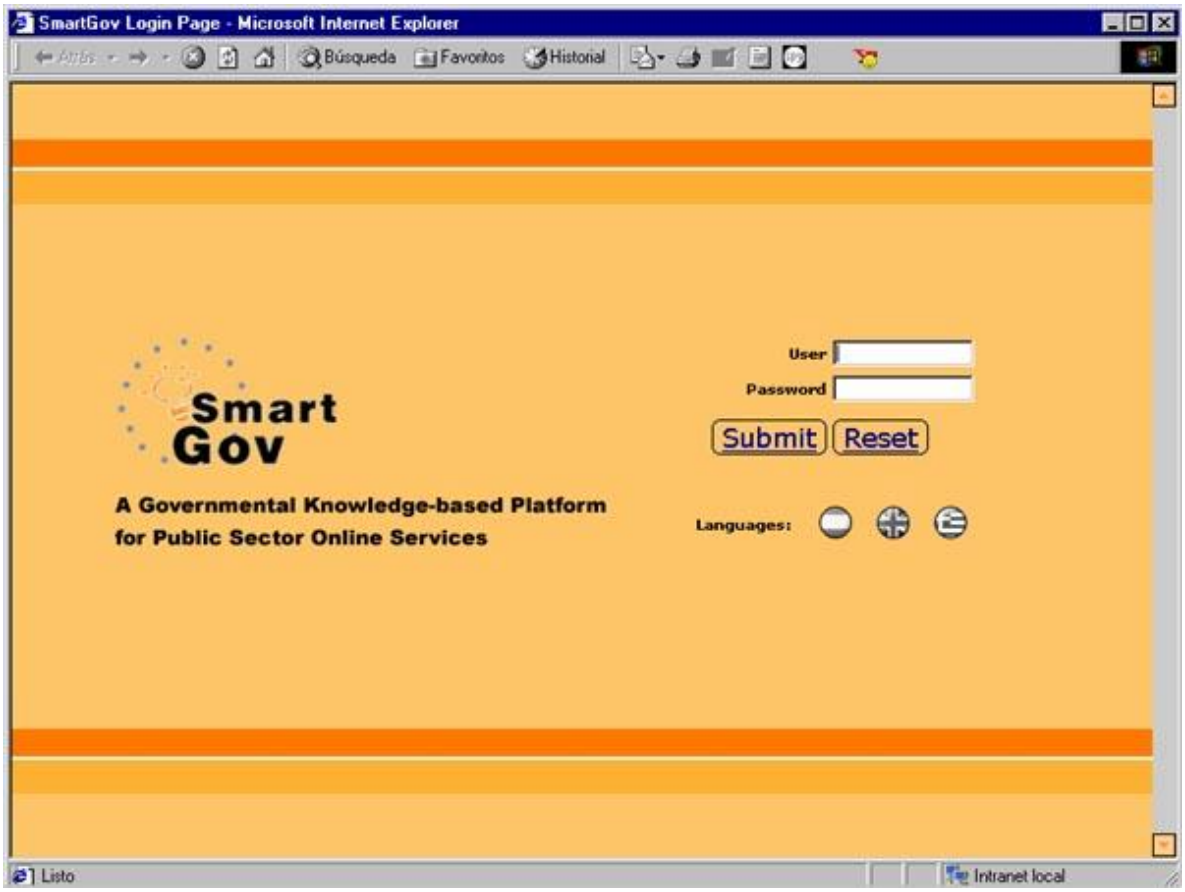


Figure 16 - Login page

Figure 16 shows the SmartGov login page, in which the following fields and actions may be identified:

Fields:

- User: The Id of a user of the SmartGov designing environment of services. This user Id should be provided by the SmartGov administrator.
- Password: The password corresponding to the previous user Id. Firstly provided by the administrator.

Actions:

- Submit: By pressing this action the data introduced is sent to the server and processed. If all validations are OK, the SmartGov main portal page is called.
- Reset: Erases the data entered.
- Flags: Change the language during the duration of the current session.

Validations:

- Mandatory fields: User and Password
- The combination of User and Password should exist in the SmartGov user security system.

Other issues:

- If the user is not able to enter to the system should contact with the administrator.

3.3.3 SmartGov Portal

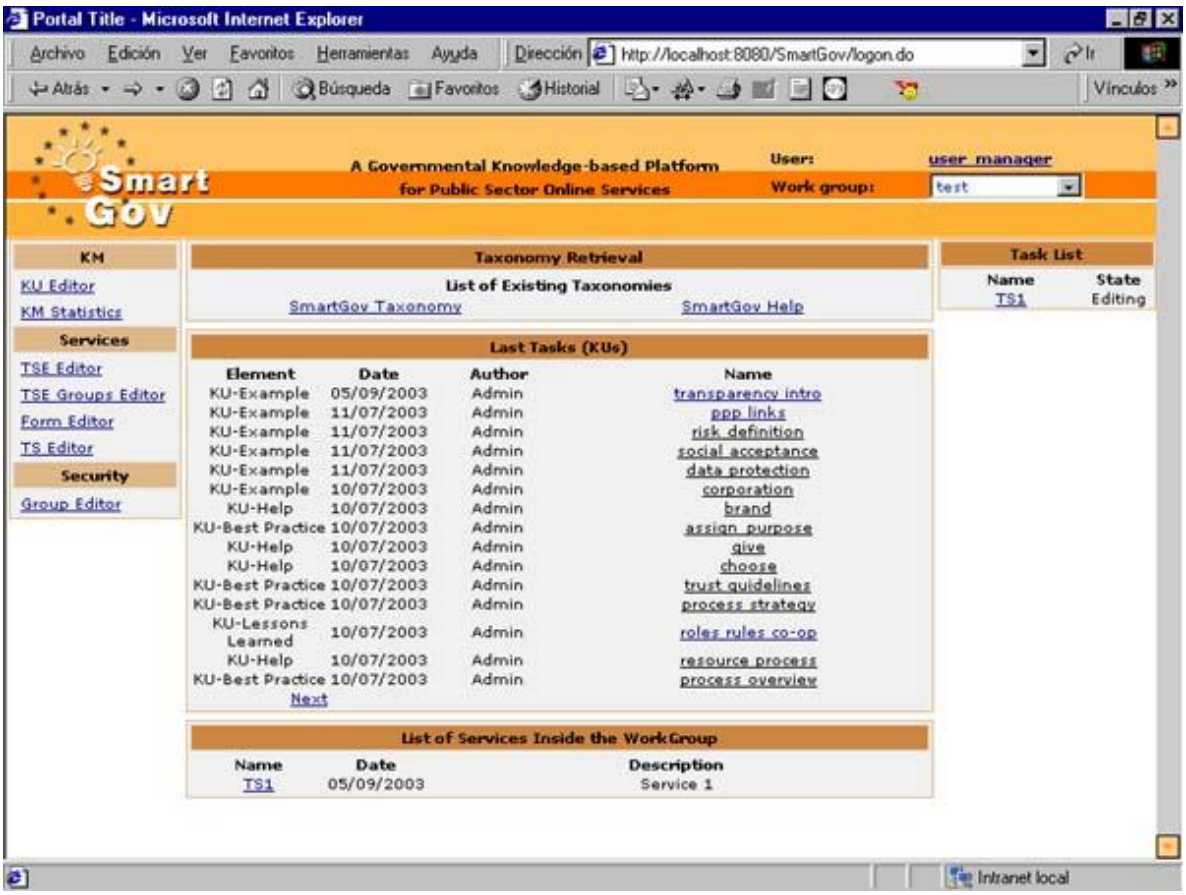


Figure 17 - SmartGov Portal page

Figure 17 shows the SmartGov portal page. This page is shown once the user is successfully logged in the system. The following elements and actions may be identified in it:

Title bar: Shows the user logged-on and the default work group (if any) to which the user belongs.

- User: Shows the user Id as a link to the page where the user logged-on is able to change their password.
- Work Group: show a selectable combo-box where the user is able to switch among the different work groups to which he/she belongs. A change in the work groups causes the reloading of the portal page with the new selected group with a different Task List and List of Services inside the Work Group objects.
- Logout: next to the user id is located the logout icon, that can be used to exit the Front-end and cancel current session. The user will be redirected to the login page (see 3.3.2) and then it is possible to log into de application as other user or as the same user, or to go on using the web browser.

Contextual Menu: Shows the options available to the user, depending on the role that plays in SmartGov. Clicking on one of the options leads to a concrete editor or portal-page. Several options are common to all roles, but others are available just for concrete roles, as it can be seen in the following table:

		Administrator	Manager	Service Worker	Expert	IT Staff
KM	KU Editor	✓		✓	✓	✓
	Taxonomy Editor	✓			✓	
Services management	TSE Editor	✓		✓	✓	✓
	TSE Group Editor	✓		✓	✓	✓
	Form Editor	✓		✓	✓	✓
	TS Editor	✓		✓	✓	✓
User mgnt.	Group Editor	✓				
	User Editor	✓				

Taxonomy Retrieval: Shows the links to the available taxonomies to perform a taxonomy-based search. Every taxonomy link loads a taxonomy tree-like page with its existing taxonomy nodes as it can be shown in 3.5.3.2.

Last Tasks (KUs): Shows a list with some information and the links to the last modified KUs. It helps to know which KUs have been modified recently and the author of the update, and it is a quick shortcut to enter directly to the most recently acquired knowledge.

List of Services Inside the Workgroup: Shows a list with all the TSs that have been created under the umbrella of the currently selected work group.

Task List: Shows a list with all the objects (TSs and KUs) that needs the approval of the role to which the current user belongs. Entering to these objects is the way to approve or reject the tasks. See sections 3.5.1 and 3.6.1 for more details about KU and TS life-cycle.

3.4 User-Roles management

3.4.1 User-roles Portal

The security part of the application is divided in two main parts: user management and group management.

When a user clicks in User Editor option, the User Portal page (see figure 18) is loaded. This portal enables the user to create new users, and to access the already existing users to modify their characteristics.

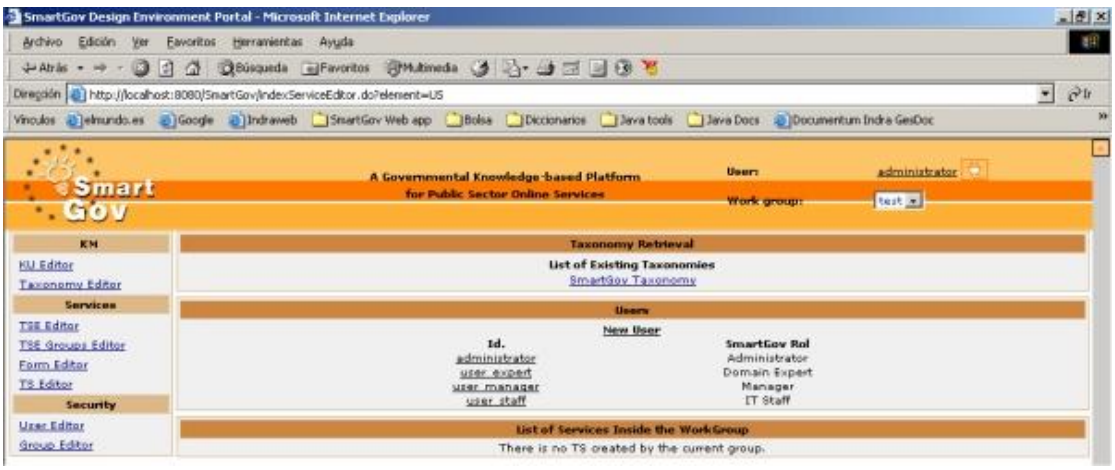


Figure 18 - User Portal page

The structure is very similar to the SmartGov Portal (see 3.3.3) The only differences are the Users list (in the middle of the screen), and the Task List, which is not present in this page.

In the Users list, all the existing users are listed and their attributes can be modified making click over them, or a new user can be added to the system (see section 3.4.2 for more details about the User editor).

The other available option, related with security management, in the contextual menu, is the Groups Portal Page (see figure 19). As in the User Portal, the structure is very similar to the SmartGov main portal, replacing the list of last KUs by a list of the groups the current user belongs to, and eliminating the Task List.

Figure 19 - Groups Portal page

The Groups List will show all the groups the user belongs to. From these list the current user can access to these groups to modify their members or their roles in the group, or can create a new group and add to it user with their corresponding roles (see section 3.4.3 for more details about the Group Editor page).

3.4.2 User Editor

The figure 20 shows the User edition page. This page is shown once the user selects an already existing user or tries to create a new one (in this case the fields will be empty).

Actions:

- Save
- Delete

Fields:

- Id: The Id of the User. If the user is new, this field is updateable. This will be used later by the user to log on the Front-end.
- Password: The password that the user will use to log on the Front-end. It can be changed by the user once logged on the Front-end, as described in section 3.3.3, in the description of the Title bar).
- SmartGov Role: the role of this user in the Front-end. The possible values are: Administrator, Domain Expert, Manager, IT Staff and Service Worker.
- E-mail: Electronic mail address of the user.

- Name and surname: personal data of the user.

Validations:

- Mandatory fields: all the fields should be introduced.
- The user id must not previously exist.

Figure 20 - User Editor page

3.4.3 Group Editor

The figure 21 shows the page to create a new Group. This page is shown once the user clicks in the New Group link, in the Groups portal page. Once the Group Id an Group Name are filled, the Group can be saved and then it is possible to add users to this group.

Figure 21 - Group Editor page

The figure 22 shows the edition of an already existing group (it is similar to the page shown when creating a new group, but contains more data).

The following fields and actions may be identified:

Actions in action bar:

- Save
- Delete

Fields:

- Group Id: The Id of the Group. Only can be modified when creating the group.
- Group Name: The name of the group.
- User in the group: a list with the users in the group and their roles.

Actions in links:

- Edit user roles in the group: Clicking in the user id, the roles of the user in the group can be modified. If no role is specified for the user, it will be deleted from the group.
- Add new User: This allows to select a user and his roles in the group.

Validations:

- Mandatory fields: Group Id and Name.
- The Group id must not previously exists.

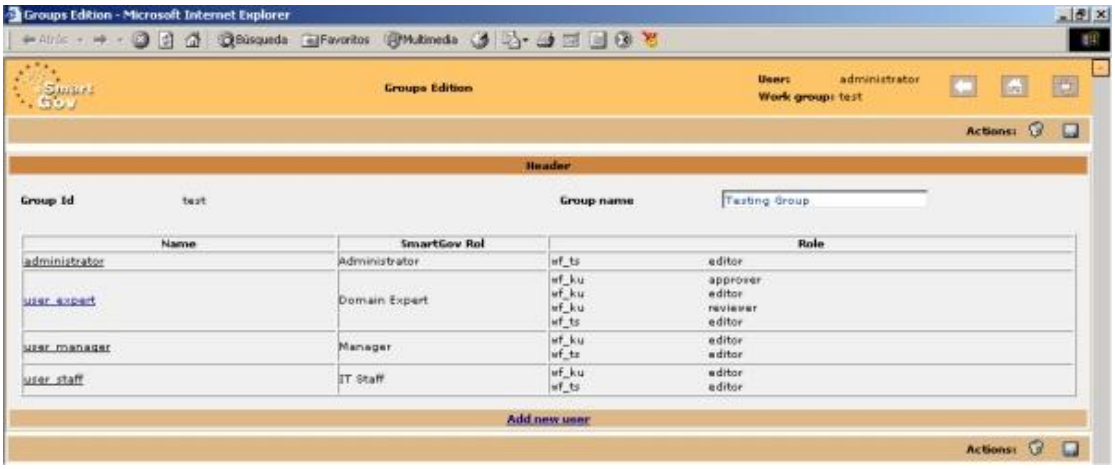


Figure 22 - Group Editor page

When the current user wants to add a user to the group, a new page is accessed (see figure 23), where can be selected the user to add and the roles of this user.

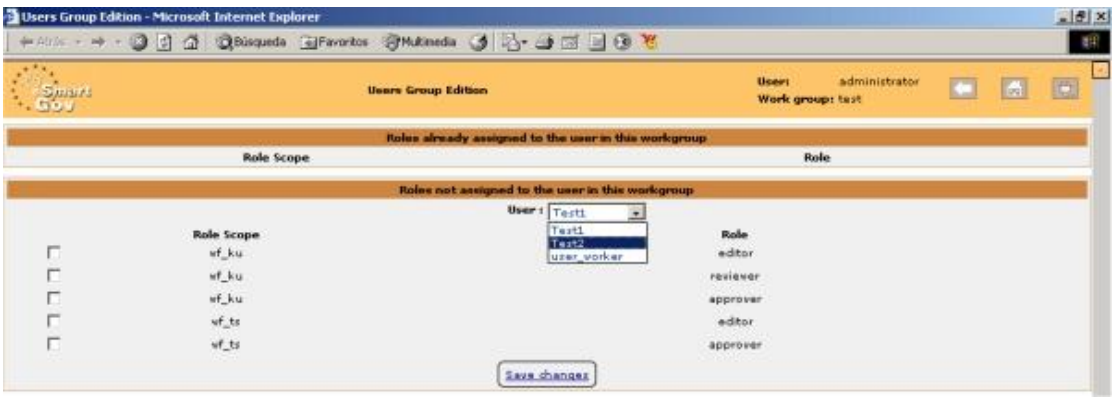


Figure 23 – Linking a user to a group page

The following fields and actions may be identified:

Actions in links:

- Save changes: add the user to the group with the selected roles.

Fields:

- User: The user to be added. This field shows a list with all the user in the system not in the group.
- List of roles of the user in the group: the page shows a list with all the roles that can be assigned to the user in the group, and besides each role there is a check box to select it. The roles are shown as follows:

Role Scope	Role	Explanation
wf_ku	**editor**	**KUs editor**
wf_ku	**reviewer**	**KUs reviewer**
wf_ku	**approver**	**KUs approver**
wf_ts	**editor**	**TS editor**
wf_ts	**approver**	**TS approver**

  See sections 3.5.1(KUs) and 0 (TSs) for a deeper explanation about the TSs and KUs life-cycle.

Validations:

- A user must be selected.
- At least one role must be selected. If not, the user will not be added to the group.

3.5 Managing Knowledge

The Knowledge is key part of the SmartGov platform, especially for the Front-end. Knowledge Units (KUs) and Taxonomies have been included in the system to allow the user to enrich the process of service development, allowing the users to search in the already existing knowledge to solve their doubts or improve their way of working, and also collaborate adding their own knowledge, related with the tool or with the whole process of offering electronic services to citizens.

The Front-end allows the user to create and modify these Knowledge elements. Concerning the KUs, it also allows the user to apply a life-cycle over them, in order to assure the quality and correctness of these elements before using them.

3.5.1 The KU life-cycle

In the figure 24 is shown the KU life cycle. The diagram shows the four possible states for a KU, the available actions in each state and the required role to

perform this tasks. Thus, all the knowledge created can be validated in the context of the workgroup.

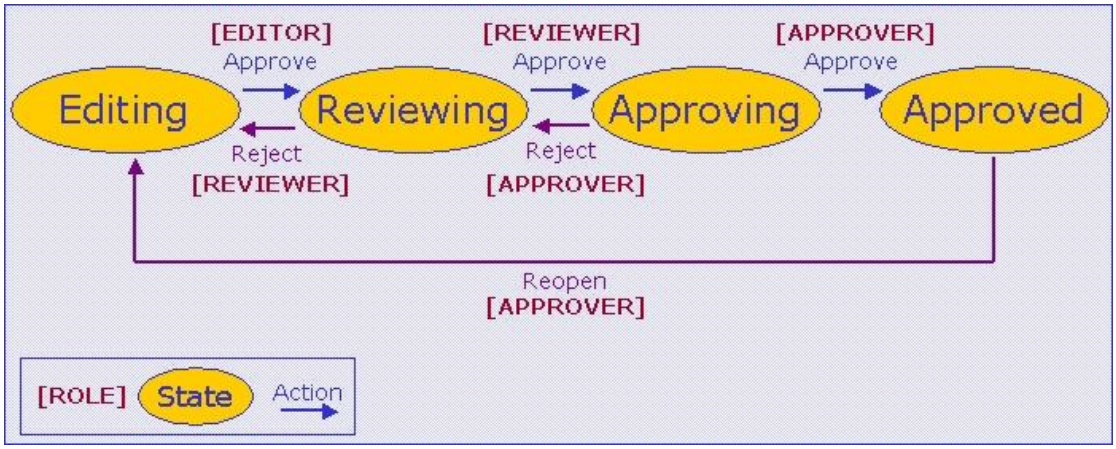


Figure 24 - Knowledge life-cycle figure

When a user logs on the Front-end application, the SmartGov Portal will show on the right side the task list. This list will show all the KUs in a state in which the user has the right role to perform an action, save the approved KUs, that will not be shown in the task list.

3.5.2 Capturing KUs

3.5.2.1 KU Editor

3.5.2.1.1 KU Portal

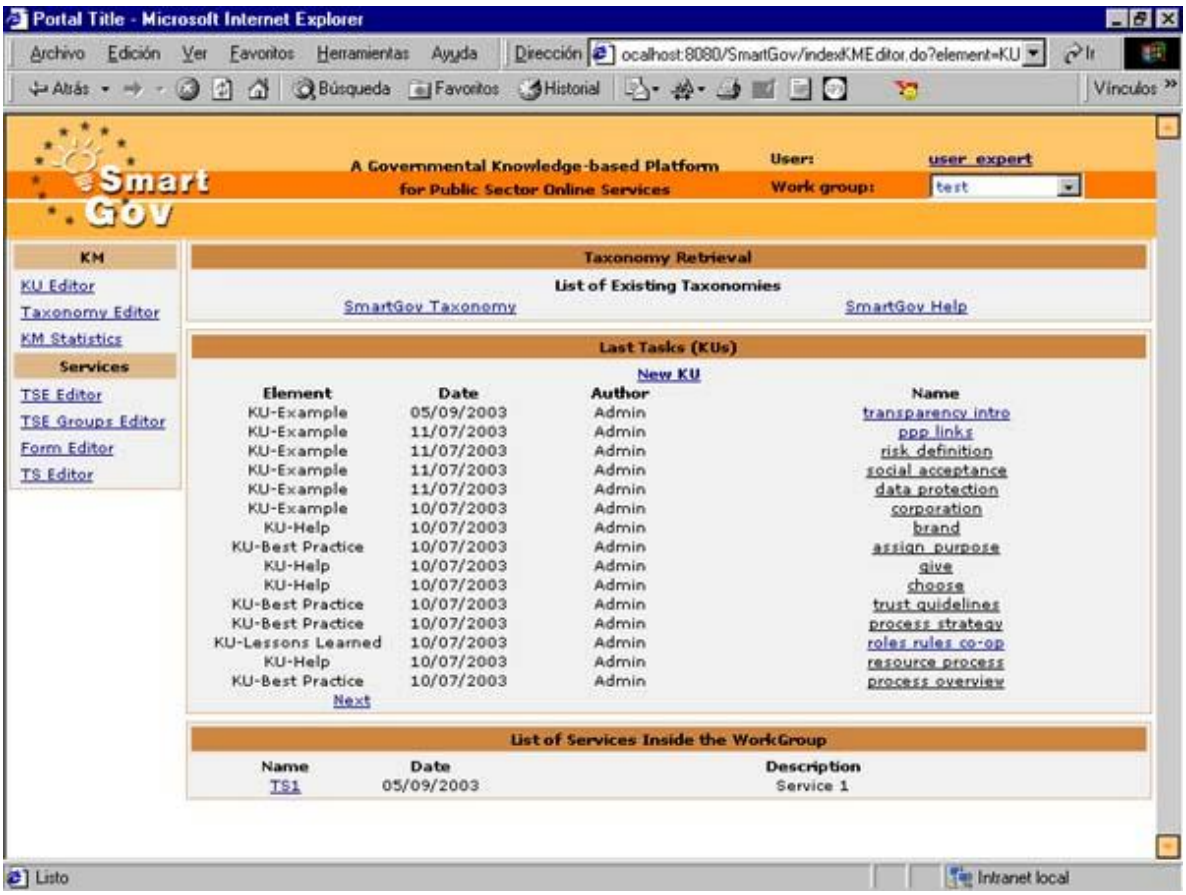


Figure 25 – KU Portal page

Figure 25 shows the KU Editor portal page. This page is shown once the user selects the KU Editor option in the Menu. The structure is similar to the SmartGov portal (see 3.3.3) plus the possibility to add new KUs if the user has the required role to do that (all roles except Administrator). Clicking in the name of one of the KUs the user can access all its data.

3.5.2.1.2KU Edition page

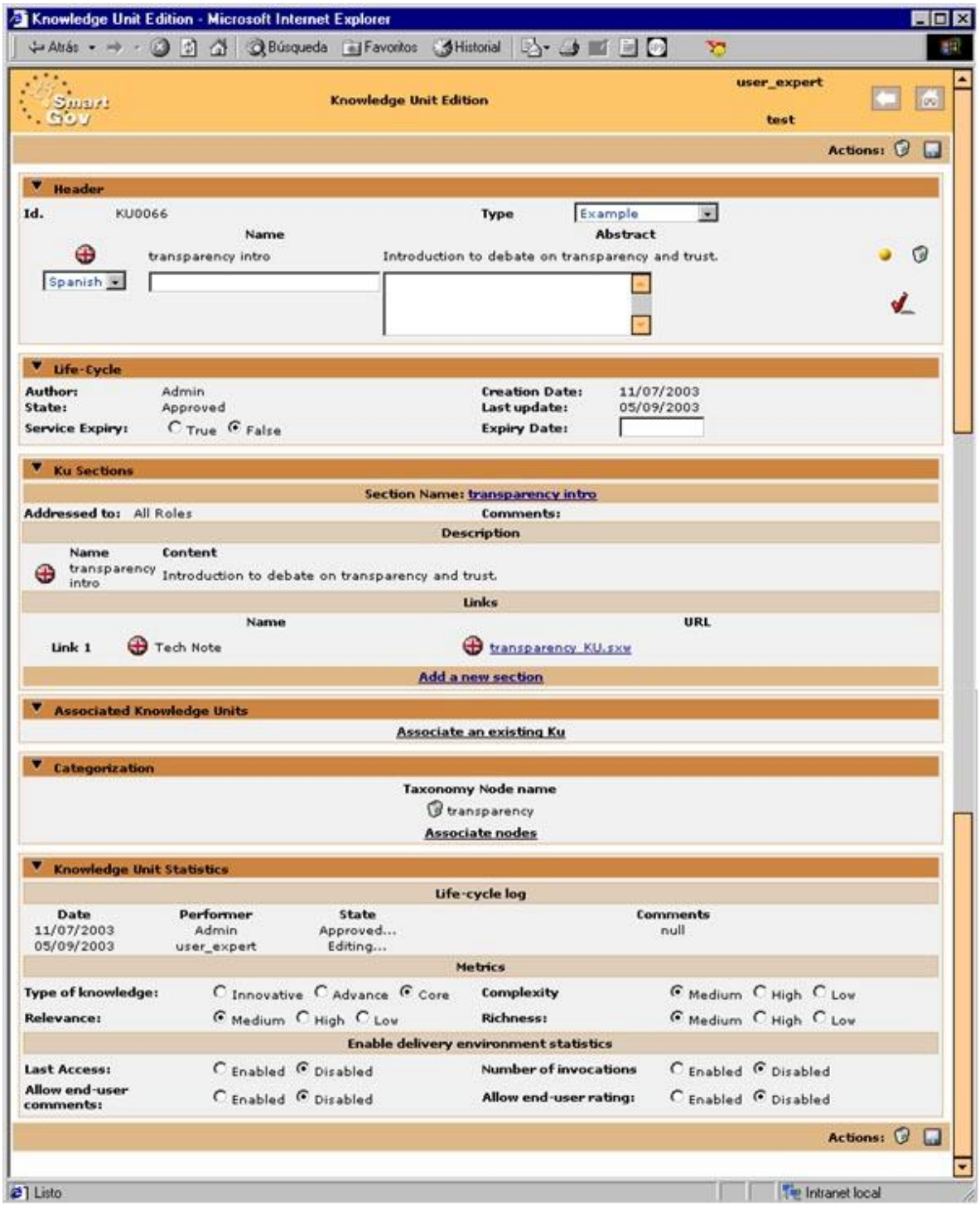


Figure 26 - KU Edition page

Figure 26 shows the KU Edition page. This page is shown once the user selects a KU and he/she has the rights to update the KU. The Figure shows the sections unfolded for clearness purposes. The following fields and actions may be identified:

Actions:

- Save

- Delete
- Approve (Depending of the role of the user and the state of the KU)
- Reject (Depending of the role of the user and the state of the KU)

Fields:

Header Section

- Id: The Id of the KU. If the KU is new, this field is updateable. The user should provide a KU identifier without spaces and special characters, as hyphens, slashes, and so on. Notice that this Id should be human readable in order to easily search and locate the KUs afterwards.
- Type:
  - Help
  - Lessons Learned
  - Just-In-Time training
  - Best Practices
  - Troubleshooting
  - Storytelling
  - Example
- Description Table (Name and Abstract) – See Multi-lingual tables (3.3.1.3).

Life-cycle Section

- Author: The initiator of the KU.
- Creation date
- State: current state of the Ku.
- Last update: the last time that this KU was saved.
- Service Expiry: This flags enables to activate if a KU lost its validity when the service that is linked to expires.
- Expiry Date: Sets when this KU lost its validity.

Ku-Sections Section: This section is the core of the KU. Here the KU content, links and attachments are loaded. The fields are shown as non-updateable fields. To update or create new sections, the action "New Section" or the links to previous existing sections should be selected. Later on (3.5.2.1.3) the way to add or modify sections and the fields involved will be discussed.

Associate knowledge units: A KU can be associated to other KUs in order to see structure knowledge. The way of attaching KUs is explained in 3.3.1.7.

Categorization: The way of linking a KU or other SmartGov elements to Taxonomy Nodes is explained in 3.3.1.8.

Knowledge Unit Statistics: The definition of the desired delivery environment statistics of the KU, and some information about the life-cycle and usability of the KU in the design environment is shown is this section.

- Life-cycle log: Contains a table that shows the most important steps and updates in the KU life-cycle: *Date* of modification, *Performer* or author of the update, *State* to what the KU was driven by the update, and *Comments* (if any) of the performer.
- Metrics: Categorization of the type of knowledge given by the experts.
- Enable delivery environment statistics: The definition of the desired delivery environment statistics of the KU. If the different types of desired statistics are set to "Enabled", then the Integrator will proceed to generate the required code if the KU is driven to the delivery environment.

Validations:

- Mandatory fields: KU Id, and at least one description (name and abstract) and one KU Section.
- The KU id must not previously exists.

3.5.2.1.3KU Sections page

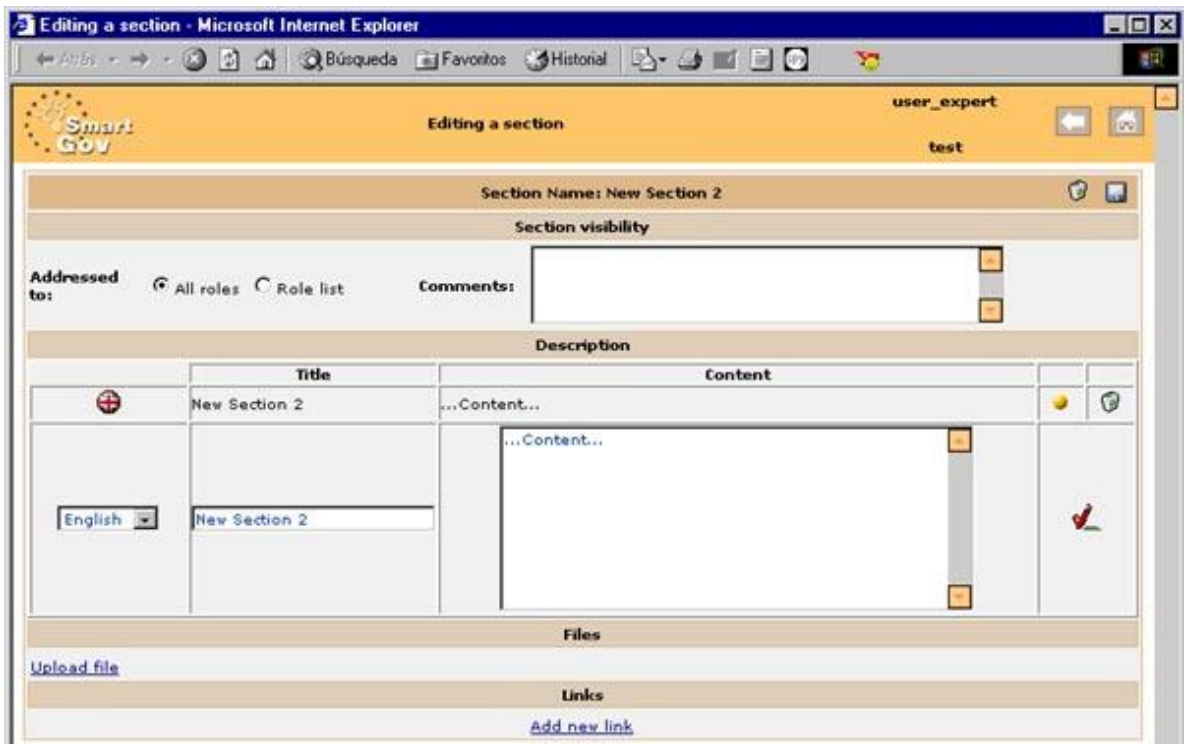


Figure 27 - KU Section edition page

The KU Section is the core of the KU. When a new section is created, the figure 27 is shown with the following fields and actions:

Actions:

- Save: save in session the KU Section. Remember that this saving action does not mean a real storing in the database until the whole KU will be saved ("Save" action in the KU) later on.
- Delete: delete from session the KU Section. As in the previous action, it does not mean a real deletion until the KU will be saved ("Save" action in the KU).

Fields:

- Addressed to: The KU content of this section is addressed to the roles selected in this field ("All roles", "Manager", "Domain Expert", "IT Staff", "Service Worker" or "End user").
- Comments: comments that explain the "addressed to" field allocation (if necessary).
- Description: title and content of the section (core content). The title field will be the name of the section. See Multi-lingual tables (3.3.1.3).
- Links: available links (web pages and files). The links should be entered in pairs of name and URL:
    - Name: Name or textual description of the link. A name in the desired languages should be provided to the link. See Multi-lingual tables (3.3.1.3).
    - URL: URL of the link in web format. It means that the URL should be entered in a format understandable by the web, for instance "http://www.xxx.com" instead of "www.xxx.com". See Multi-lingual tables (3.3.1.3).

      Instead of referring to an existing URL, a file may be uploaded to the server and a reference to that file would be inserted in the section. When the action "Upload file" is selected, a new window to upload a file is presented (see 3.3.1.9). The uploaded file is treated as a new link.

  There is also a shortcut to create a link directly uploading a file. If the action "Upload file" located under the "Files" title is selected, a new link will be created with the uploaded file, using as name of the link the name of the file.

Validations:

- Mandatory fields: at least one description (title and content) should be provided.

Other issues:

- When a new section is added, the page presents a default name ("New Section n", where n is the number of the section in the current KU), and a Title ("New Section n") and Content ("…Content…") for the default language. These fields should be modified in order to set the correct text of the section.

3.5.2.1.4 KU read-only page

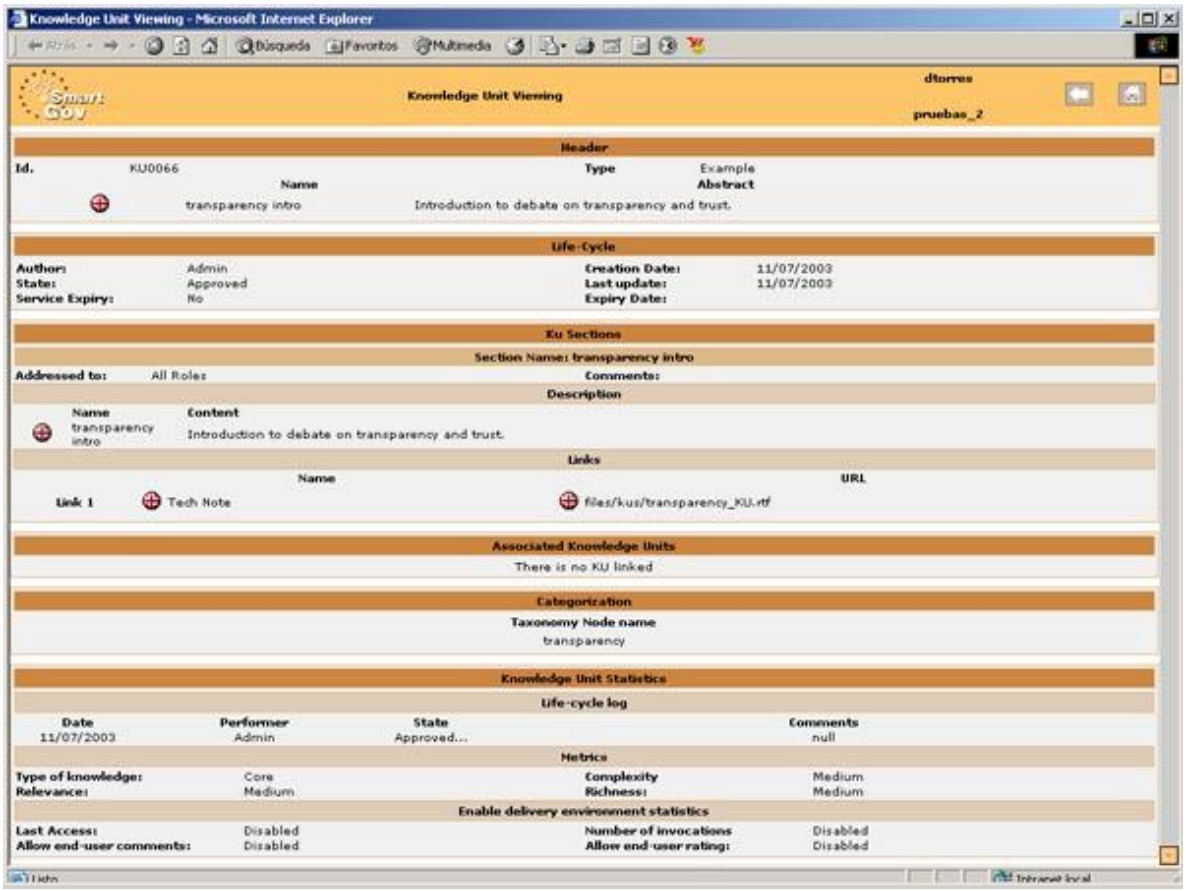


Figure 28 - KU read-only page

This page shows the KU in read-only mode. This page is shown when selecting from the different portals and elements with related KUs in the following cases:

1. The user has not rights to modify the KU.
2. The KU is in a state that not allows updates.

Actions:

- Reopen (only if the user belongs to the group where the Ku was created and has the Approver role).

3.5.2.1.5KU reduced read-only page

Figure 29 - KU read-only reduced page

This page shows a reduced view of the KU in read-only mode. This page is shown when selecting the KU from the list of elements related with a taxonomy node. It simplify the view of the KU to its core (name, abstract, description section, links and associated KUs), what means a user friendly view of the most important knowledge contained in the KU, avoiding all the data related with the creation and management process.

3.5.3 Retrieving Knowledge

3.5.3.1 Taxonomy Editing tool

3.5.3.1.1 Taxonomy Editing portal

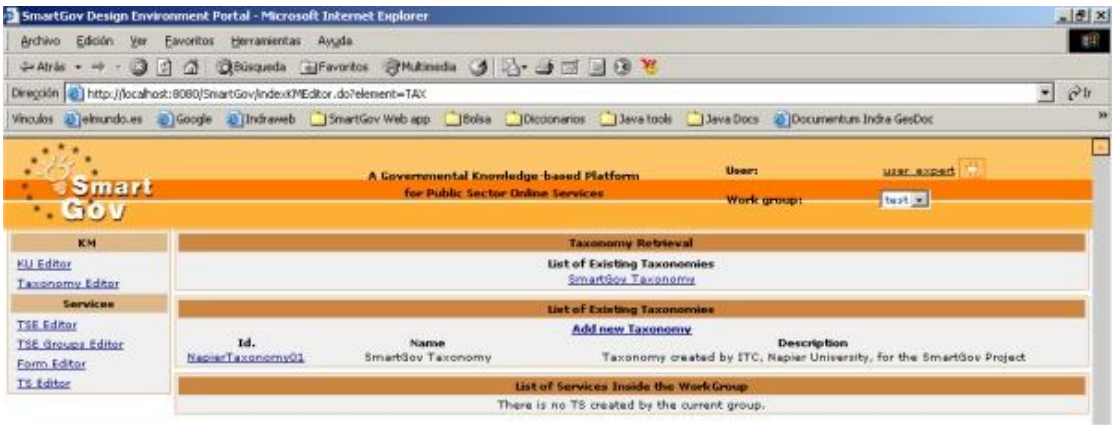


Figure 30 - Taxonomy portal

Figure 30 shows the Taxonomy portal page. This page is shown once the user selects the Taxonomy Editor option in the Menu. The structure is similar to the

SmartGov portal (see 3.3.3) plus the possibility to add new Taxonomies if the user has the required role to do that (Administrator or Expert).

3.5.3.1.2 Edit Taxonomy

Figure 31 shows the Taxonomy Edition page. This page is shown once the user selects a Taxonomy from the Taxonomy portal.

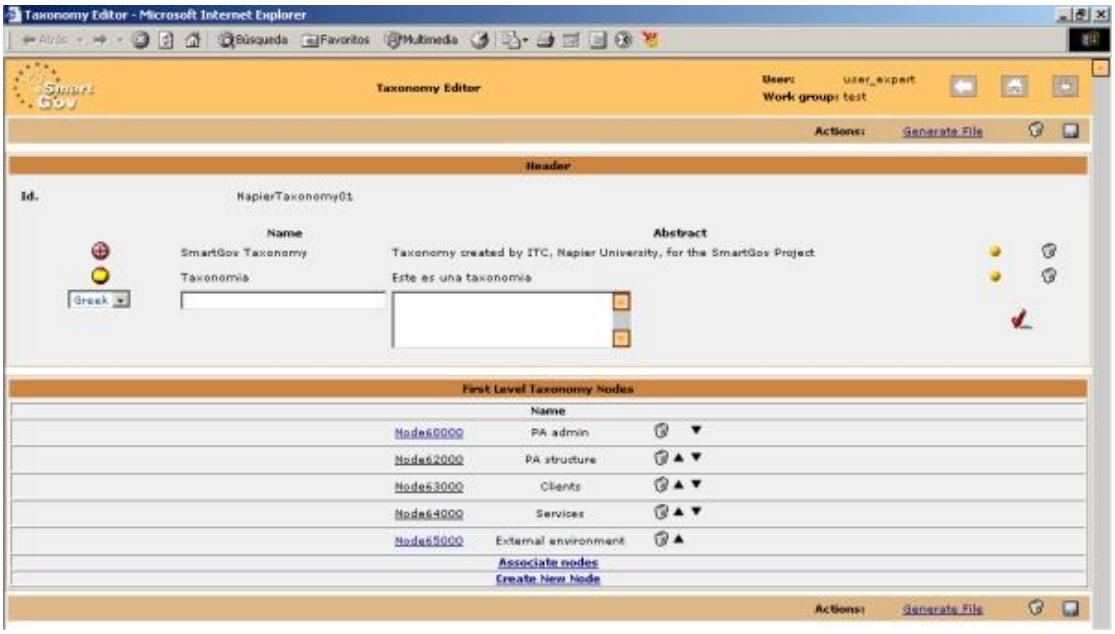


Figure 31 - Edit Taxonomy page

The following fields and actions may be identified:

Actions:

- Save
- Delete
- Generate file: updates the current state of the Taxonomy to the tree view that is displayed when clicking in the Taxonomy Retrieval section included in all the portal pages.

Fields:

Header Section

- Id: The Id of the Taxonomy. If the Taxonomy is new, this field is updateable. The user should provide a Taxonomy identifier without spaces and special characters, as hyphens, slashes, and so on.
- Description Table (Name and Abstract) – See Multi-lingual tables (3.3.1.3).

First Level Taxonomy Nodes Section:

This sections shows a list with all the taxonomy nodes in the first level of the taxonomy. Nodes can be moved up and down using the arrows beside them, and also deleted from the list with the trash icon. Also the nodes can be accessed by clicking in their id.

In the bottom of the list there are two actions:

- Associate nodes: allows selecting an already existing node (see sections 3.5.3.1.4 and 3.5.3.1.5 to see how to select nodes).
- Create new node: allows creating a new node (see section 3.5.3.1.3 about how to edit nodes). This action will not associate the new node to the taxonomy; the node must be linked using the previous action.

Validations:

- Mandatory fields: Taxonomy Id, and at least one name.
- The Taxonomy id must not previously exists.

3.5.3.1.3 Edit Taxonomy Node

Figure 32 shows the Taxonomy Node Edition page. This page is shown once the user selects a Taxonomy Node from the Taxonomy edition page or from other Taxonomy Node edition page.

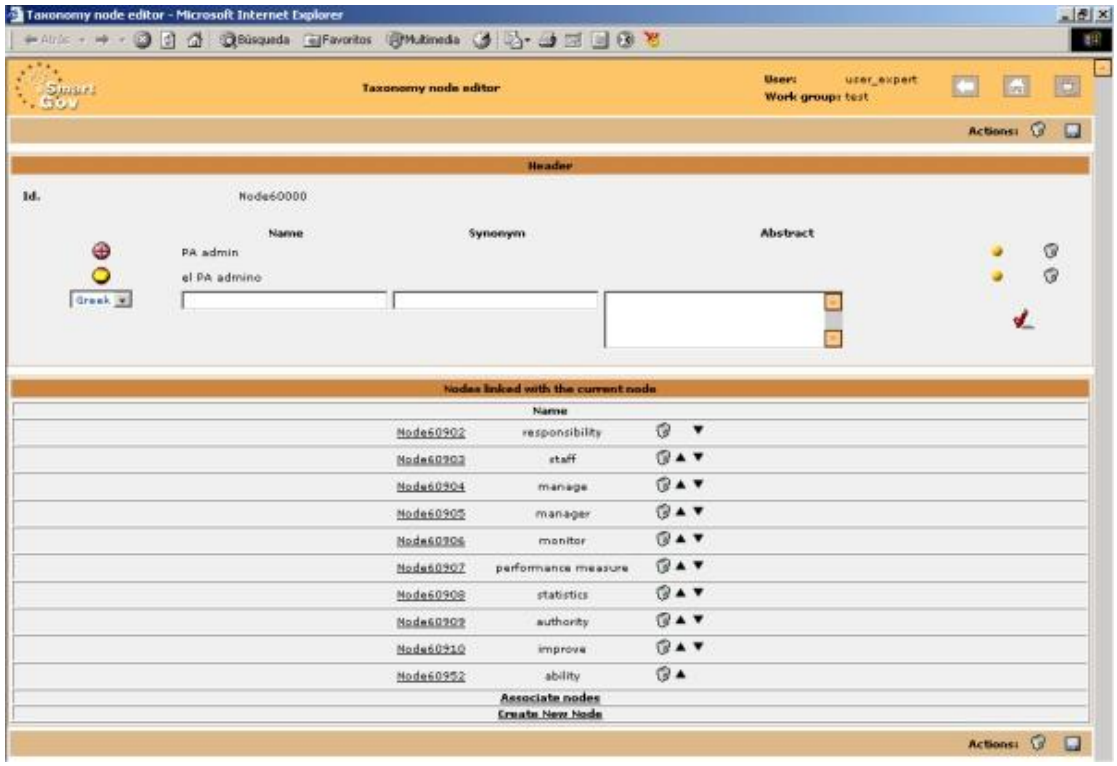


Figure 32 - Edit Taxonomy Node page

The following fields and actions may be identified:

Actions:

- Save
- Delete

Fields:

Header Section

- Id: The Id of the Taxonomy node. If the Taxonomy is new, this field is updateable. The user should provide a Taxonomy node identifier without spaces and special characters, as hyphens, slashes, and so on.
- Description Table (Name, Synonym and Abstract) – See Multi-lingual tables (3.3.1.3).

Nodes linked with the current node Section:

- This sections shows a list with all the taxonomy nodes linked with the current node. The functionality of this list is very similar to the First Level Nodes list in Taxonomy Edition (see section 3.5.3.1.2 for more details).

Validations:

- Mandatory fields: Taxonomy Node Id, and at least one name.
- The Taxonomy Node id must not previously exists.

3.5.3.1.4 Associate nodes by Taxonomy

When selecting nodes to associate them to a taxonomy or a taxonomy node, a way of selecting these nodes is navigating the already existing taxonomies, to select a node already located in other taxonomy, or in the same taxonomy in different branch.

For this purpose, a list of all the taxonomies is shown in the page to select nodes and, by clicking in one of them, the taxonomy is shown below (see figure 33) allowing to select the nodes that we want to add to the taxonomy or taxonomy node currently in edition.

Once we have selected the nodes, we can click in the "link nodes" action to confirm our selection.

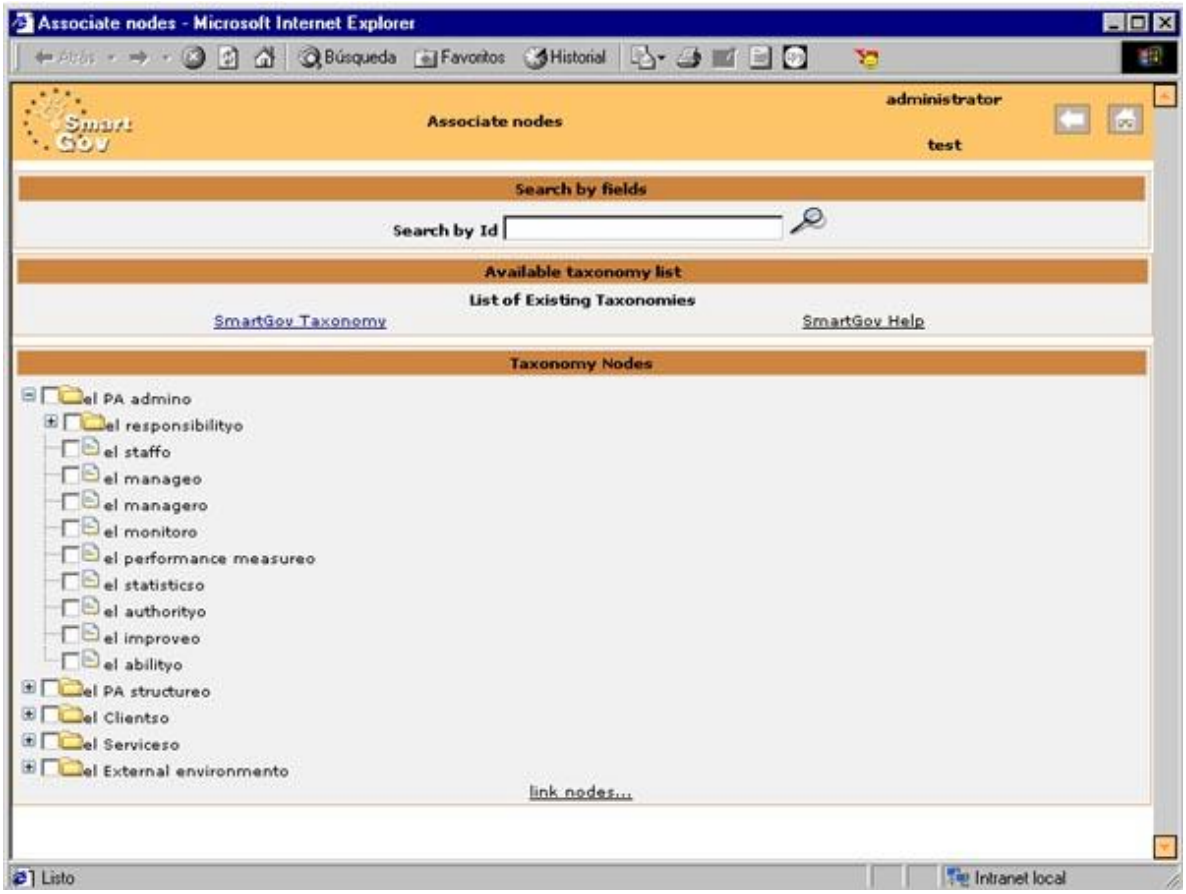


Figure 33 - Select nodes by Taxonomy page

3.5.3.1.5 Associate nodes by Node Id

Although the selection of nodes using the taxonomy can be useful in some cases, the more usual way of selecting a node will be entering its Id, because the most frequent case is that a node is only in a taxonomy, so we cannot locate that node navigating other taxonomy.

Therefore, it will be necessary remembering the Ids of the Nodes that we are creating, so it is important to use a coherent way of assigning id to the nodes. Anyway, it is allowed the use of wildcard (The "%" matches any number of occurrences of any character) in the Id field, making easier to search nodes (see example in figure).

Once performed the search, a list of the nodes fitting the introduced string is displayed. There is possible to select as many nodes as desired, to associate them to the taxonomy or taxonomy node currently in edition.

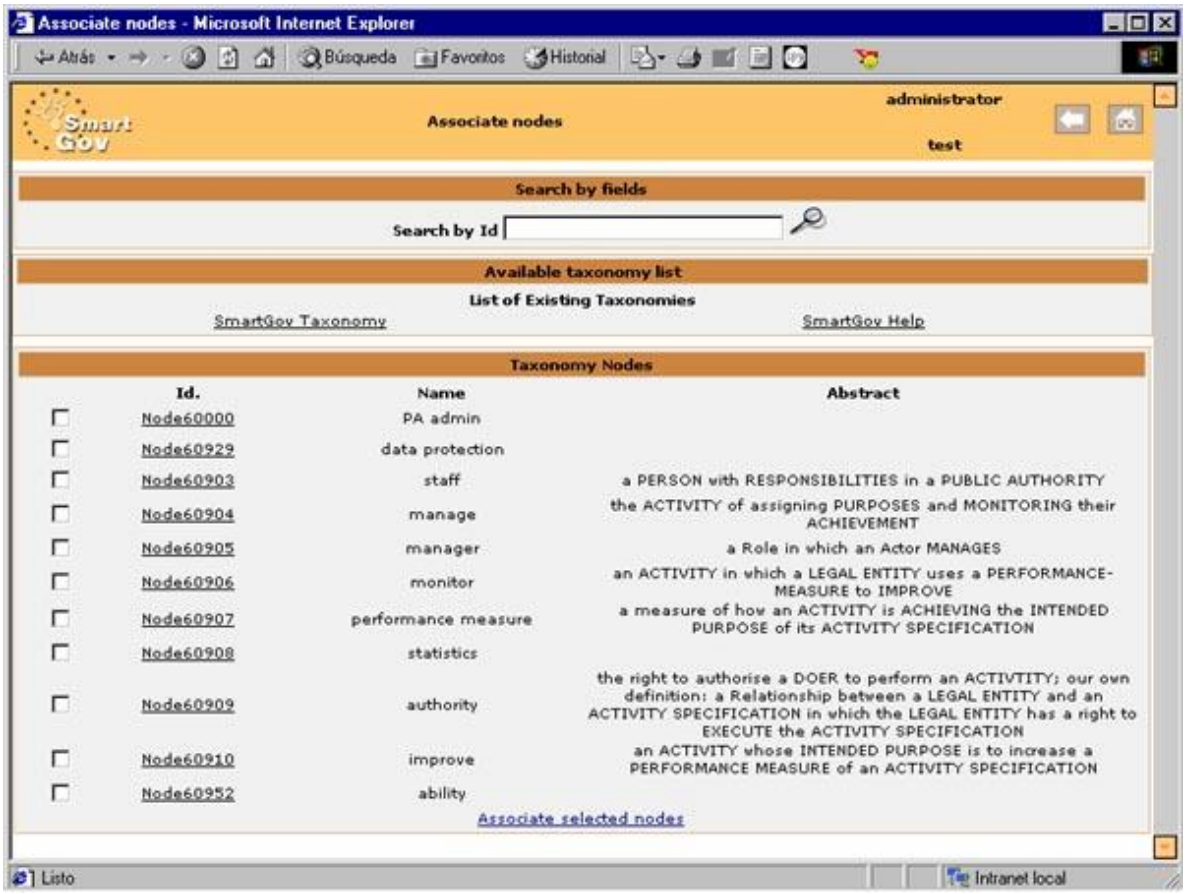


Figure 34 - Select nodes by Node Id page (after searching 'Node60%')

3.5.3.2 Taxonomy Retrieving tool

Given that it is possible to link all the elements of the platform to taxonomy nodes, a useful way of searching elements is through the taxonomy, retrieving all the elements linked to a node.

This option is available through the "Taxonomy retrieval" list, available in all the portal pages of the Front-end. In this section there is a list with all the available taxonomies and, clicking in one of them, it is possible to navigate through the taxonomy to find the searched node.

3.5.3.2.1 Taxonomy tree view

When the user selects a taxonomy in the "Taxonomy retrieval" list, then the taxonomy is shown as a tree view, allowing the user to expand or collapse the different branches (clicking in the plus or minus icon, similar to Windows Explorer), in order to find the searched node. The figure **Error! Reference source not found.** shows a taxonomy with some expanded nodes.

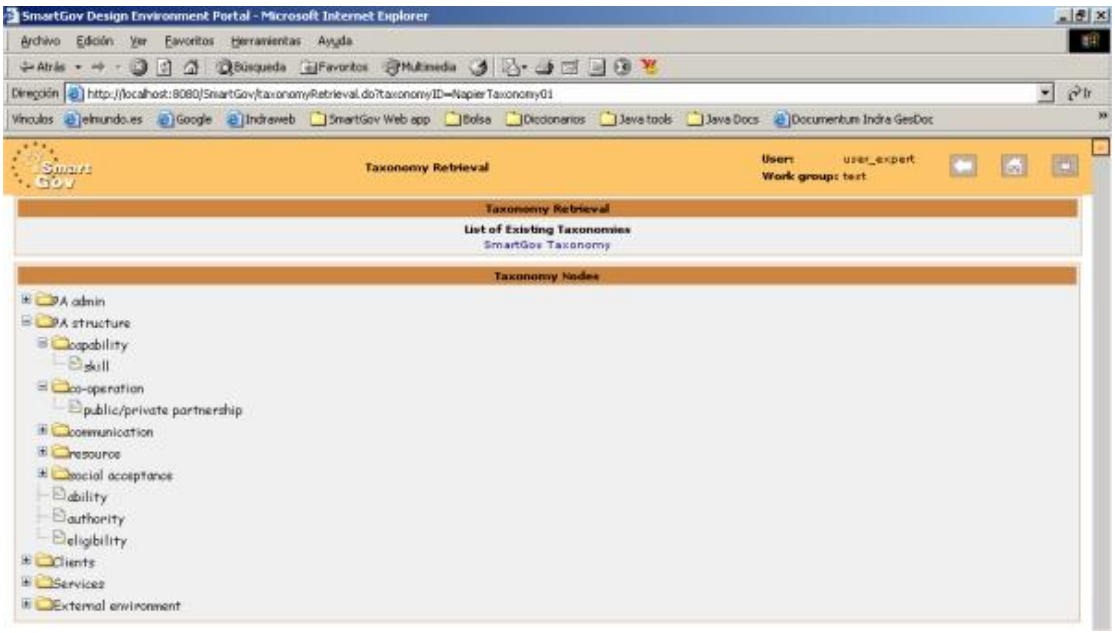


Figure 35 - Taxonomy Retrieval tree view page

Once the user has located the node, the related elements can be seen clicking on the name of the node.

3.5.3.2.2Taxonomy node related objects page

When the user selects a node in the taxonomy tree view (see previous paragraph) a new page opens with the list of related nodes (see figure 36). From this list it is possible to access all the elements related with the taxonomy node, clicking on the name.

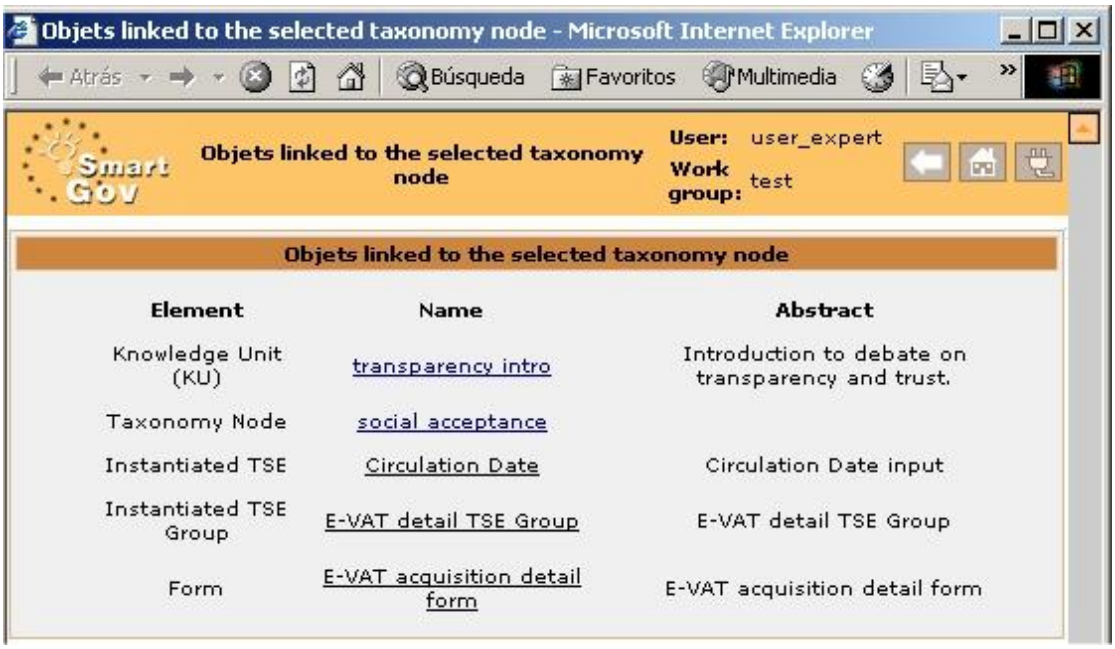


Figure 36 - Taxonomy node related objects page

3.6 Managing Service elements

The main goal of the SmartGov platform is develop electronic service. Therefore the creation and management of all the elements related with these services is a key function in the Front-end. With the added value of the associated knowledge, the Front-end coordinates the process to create a service, including the management of the whole life-cycle of the Service.

3.6.1 The TS life-cycle

In the figure **Error! Reference source not found.** is shown the TS life cycle. The diagram shows the three possible states for a TS, the available actions in each state and the required role to perform this tasks. The cycle is very similar to the KU's, but simpler.

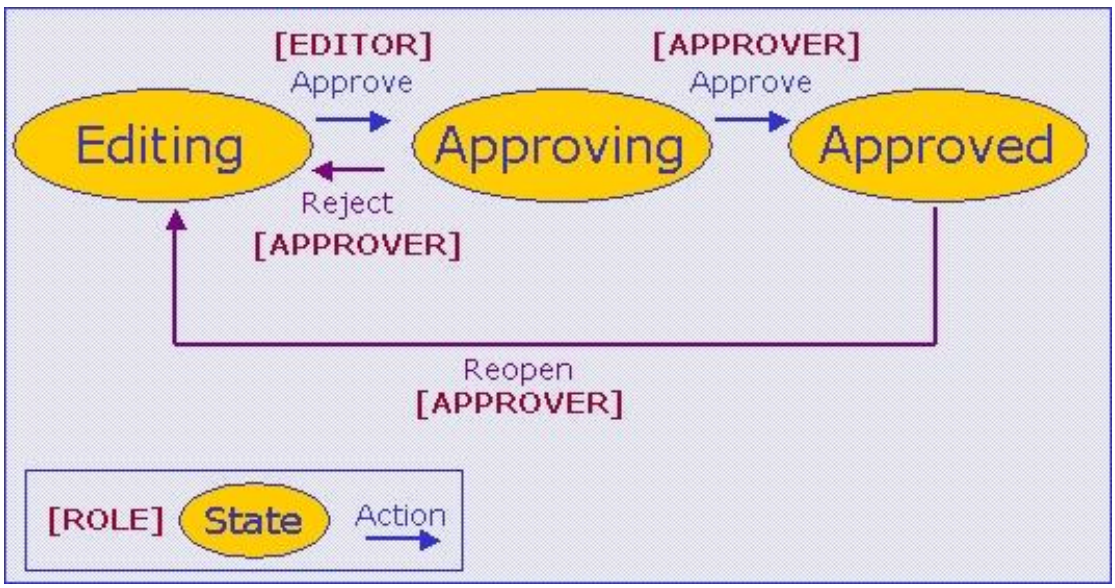


Figure 37 – TS life-cycle figure

When a user logs on the Front-end application, the SmartGov Portal will show on the right side the task list. This list will show all the TSs in a state in which the user has the right role to perform an action, save the approved TSs, that will not be shown in the task list.

3.6.1.1 Designing the service

Before start working with the Front-end tool, the service operation, the roles involved, the business rules governing the service and the data that must be presented and/or collected should be identified and documented. Portions of the documentation (e.g. supporting legislation, information regarding the workflow,

service development expected time schedule) may be stored within the SmartGov platform as knowledge units associated with the whole service.

In this phase, the service name is entered, along with a high-level description of it and at least one set of forms. The description may document the overall service functionality, the result of the feasibility study and so on. Finally, Knowledge Units and some taxonomy nodes related with the service may be entered.

The only **pre-requisite** before the TS generation consist in having at least one form element created. It is mandatory for the TS to have at least one form set. The form could be just a dummy form (just the ID and the description) if anything else have been defined.

After that, Domain Experts and IT Staff add the form elements (Forms, TSE, TSE Groups), and the KUs attached to them, the categorization and all the methods and validation rules required.

Once the service is finished, the experts should send TS to approve. The manager should approve the TS before the integration.

3.6.1.2 Integrating components

Once all necessary elements for a transaction service have been defined, the integration phase will arrange for performing a synthesis of these elements into an operational instance of the transaction service. In more detail, the integration step will perform the following actions:

1. It will access the service definition, extracting from it the links to the forms that implement the service, the validation checks pertaining to the service as a whole and the associated KUs.
2. It will retrieve the form definitions and the definitions of the TSEs appearing on each form, together with the associated validation checks and KUs. If a TSE group has been placed on a form, all TSEs belonging to the group will be retrieved, together along with their descriptions, KUs and validation checks. KUs and validation checks pertaining to the TSE as a whole will also be retrieved.
3. It will load the information regarding the statistics that need to be collected.

Once this information is available to the integrator, the service instantiation task may proceed. The integrator module will generate a page for each form defined within the service, using the form layout specification. Forms belonging to the same service will be suitably linked, based on form sequence information specified for the service; "submit" buttons will also be placed on the forms that have been designated to provide such functionality. At this stage, the

completeness of references to TSEs should be verified: each TSE declared to participate in a form, should be linked with an element of the form layout. If this is not the case, the SmartGov platform user should be informed of the discrepancy, in order to amend the situation.

Validation checks defined at TSE level and form level will be used to generate code that will validate user input. This code may be executed:

1. *At the front-end* (client-device side), if the service designers have designated that this is desirable and if the client device supports active features. Regarding the timing of the execution, these checks may be performed either when the user changes a field value (usual case when the validation check pertains to the field data type or the field value range), or when the user leaves the page (typically when the validation check involves multiple fields).
2. *At the server-side*. All input should be always validated at the server-side, since in a distributed environment clients should be considered untrustworthy, and thus the system may not rest on the perception that all client-side checks have been properly executed. Server-side checks may be run when the user leaves a page or when a final submission is made, depending on the timing specified by the service designers.

The integrator will also generate code for the validation checks defined at service level. These will be executed on the server-side upon the final submission, since in general they involve TSEs appearing in different forms, which inhibits execution at the front-end upon form change (it is not guaranteed that all involved values will have been provided).

Finally code will be generated to arrange for the communication with third-party systems through the SmartGov agent. This communication will be mainly performed when the user invokes a service, in order to retrieve values for TSEs that need to appear pre-filled in with values obtained from registries or databases.

Knowledge units that are associated with TSEs, TSE groups, forms and the transaction service and that have been designated as "help items for end-users" should be appropriately linked to the forms. The integrator should arrange for the proper generation of help pages from KUs and embedding of the hyperlinks to the appropriate anchors.

Statistics definitions will also be translated to pieces of code that will arrange for collection and storage of relevant statistics. For example, if the sum of the values filled in a specific form element has been requested to be computed, the integrator will generate code that will add the value of each submitted form to a

database element; if the number of submissions should be counted, code will be generated for adding up one to a specific database element upon submission.
Once the final pages and the associated programs have been generated, the files produced may be installed on a restricted access server for testing and evaluation purposes, or on a public access service for full service deployment.

3.6.1.3 Reopening services

After the approval of the service, modifications may be required. For instance it will be common that the service required several integrations and changes in the form definitions before a total deployment, or a service already deployed may suffer updates during its life-time.
In order to do that, the Ts should be reopened by the users with the "Approver" role. The Ts will be in "Editing" state after this operation.
The integrator is the responsible for all the tasks described in section 3.6.1.2. In the section 4 this module is completely described.

3.6.2 Introduction and common task

3.6.2.1 Introduction

The Front-end enables user to create service or to modify already deployed services. In both cases this application allows user to collaborate and work together to complete the service, establishing the Forms and elements taking part in the service, and also the "help" –using Knowledge Unit" related to the service.
In the following sections this document describes all the tasks to perform concerning the development of a service. These tasks include:

- Create or modify a TS.
- Define its forms, with their associated XHTML layouts.
- Create the required elements in the form (Generic and Instantiated TSEs and TSE Groups).

3.6.2.2 Working with Validation Rules

The validation rules enable the user to add "intelligence" to the service, by incorporating checks to assure that the data entered are conferment to the organisation's business rules governing the service.
Validation checks may apply to individual TSEs, TSE groups, forms or TS. Validation checks pertinent to specific TSEs will mainly check the data type

(integer, string, date etc) and the value range of the data entered. These validation checks may be considered as properties of the relevant TSEs.

Validation checks applying to TSE groups, forms or services will mainly check if a certain relationship between different TSEs holds. The TSEs referenced in the validation check should all be valid in context of the object within which the validation check is defined; for example, a validation check defined at TSE group level only references TSEs participating in the TSE group.

Validation checks may be entered either via a graphical interface or in textual form, using SmartGovLang, a validation rules language that has been defined within the SmartGov project. In both cases, the definition of complex validation checks will be carried out by IT staff, rather than domain experts. IT staff should be allowed to code validation checks directly in the language used by the service delivery platform (e.g. Java, JavaScript etc.) if this is found to be convenient, or if the coding language/environment provided is not expressive enough to implement the desired functionality.

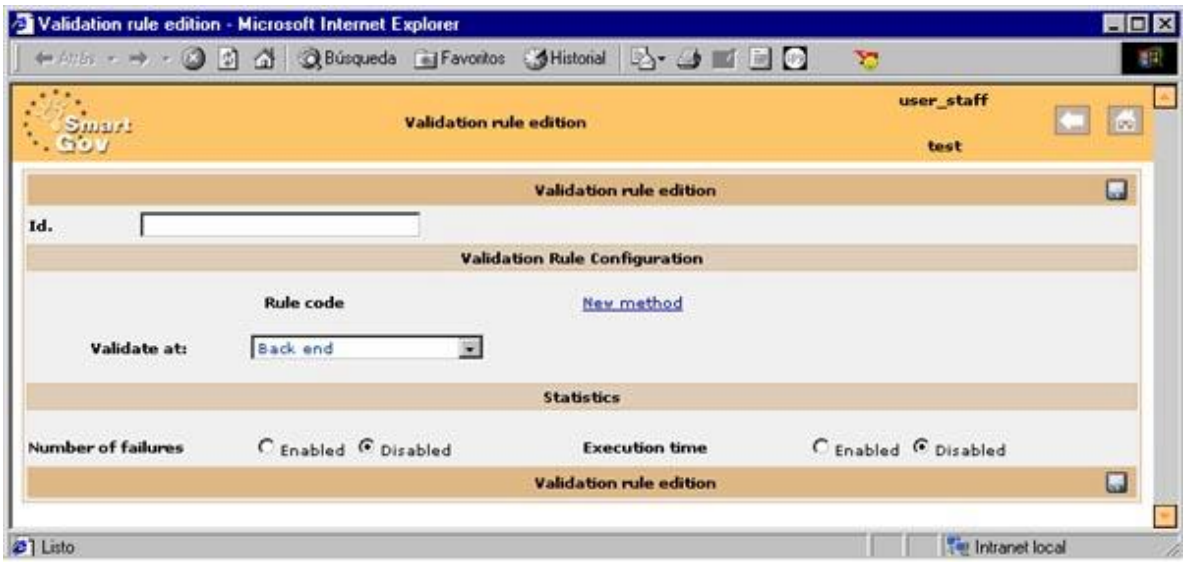


Figure 38 - Validation Check

For a deeper view over the validation checks, please review Appendix B.

3.6.2.3 Methods

Methods are used to define both validation check codification and actions to be performed when some event occurs in the related elements (for instance when a form is loaded).

The user has to choose between the following types of methods:

- Native Language
- SmartGov Language
    - Full Rule
    - Compact Rules

Please review Appendix B for a deeper explanation.

3.6.3 Service Portal

In the service area of the SmartGov portal menu there are four options, to access for different portal-like pages.

3.6.3.1 TS Portal

This portal page (see figure 39) is very similar to SmartGov main portal, but the list of last KUs has been replaced by a list with the last TSs, and the task list is not shown. The user is able to access the complete definition of the different TSs clicking in their names.

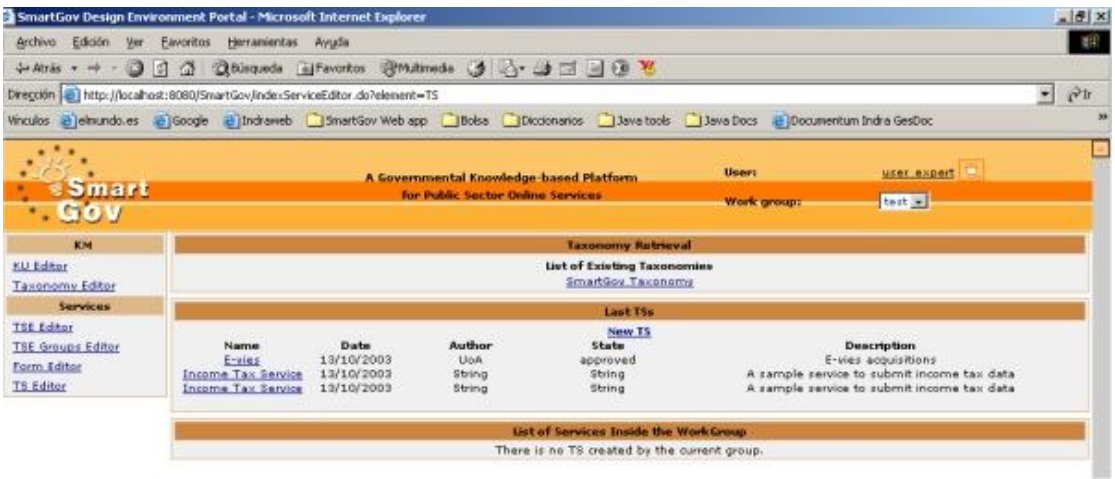


Figure 39 - SmartGov TS Portal

3.6.3.2 Form Portal

This portal page (see figure 40) is very similar to SmartGov main portal, but the list of last KUs has been replaced by a list with the last Forms, and the task list is not shown. The user is able to access the complete definition of the different Forms clicking in their names.

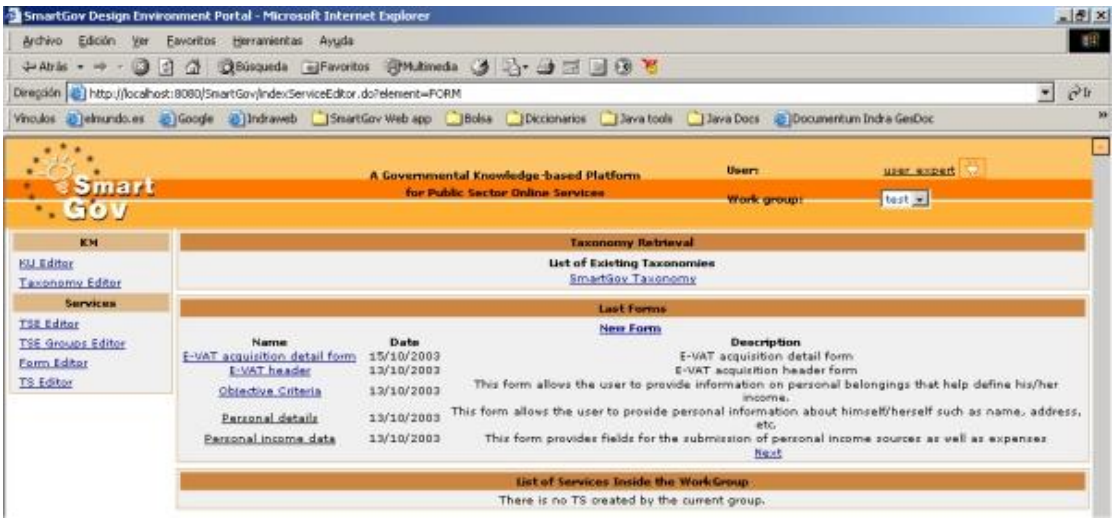


Figure 40 - SmartGov Form Portal

3.6.3.3 TSE Portal

This portal page (see figure 41) is very similar to SmartGov main portal, but the list of last KUs has been replaced by a list with the last Generic TSEs and other list with the last Instantiated TSEs. The task list is not shown. The user is able to access the complete definition of the different TSEs (Generic or Instantiated) by clicking in their names.

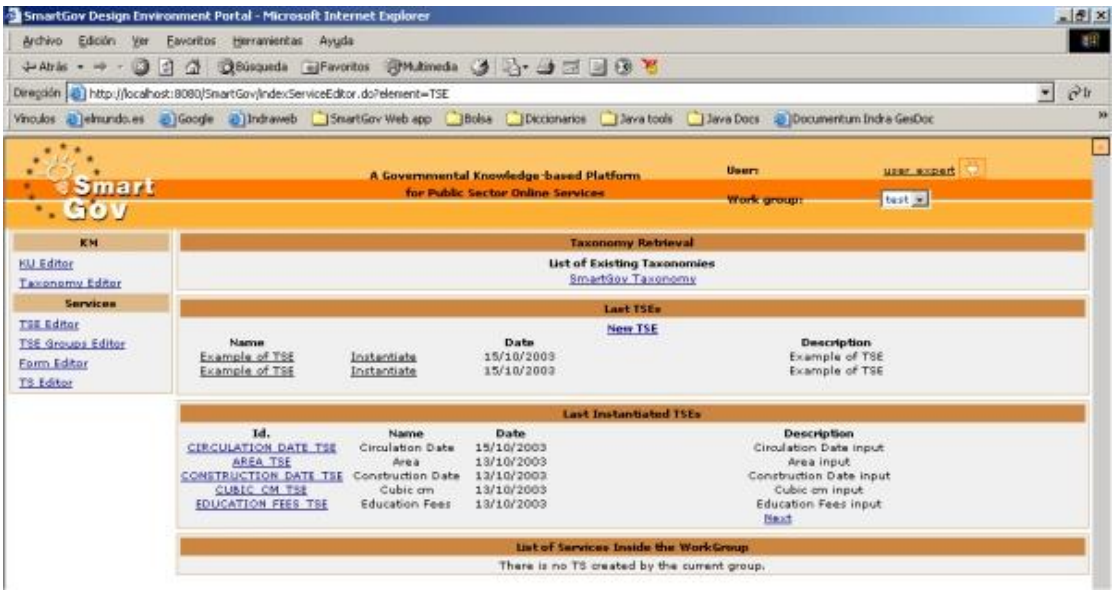


Figure 41 - SmartGov TSE Portal

3.6.3.4 TSE Group Portal

This portal page (see figure 42) is very similar to SmartGov main portal, but the list of last KUs has been replaced by a list with the last Generic TSE Groups and

other list with the last Instantiated TSE Groups. The task list is not shown. The user is able to access the complete definition of the different TSE Groups (Generic or Instantiated) by clicking in their names.

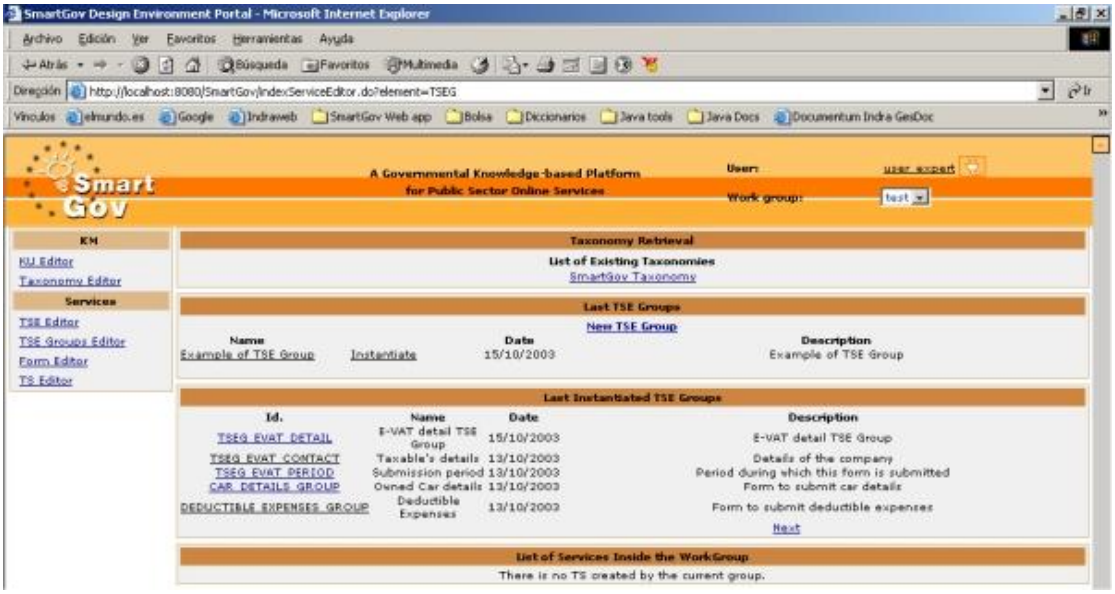


Figure 42 - SmartGov TSE Group Portal

3.6.4 Development of Transaction Service Components

3.6.4.1 Introduction

During this phase the various components of the electronic service are taking a concrete form within the SmartGov platform. The following paragraphs elaborate on the process of creating the different elements. It is important to note that, once the service process model has been defined, the object definition need not be carried out sequentially. For instance, the form layout may be developed in parallel with the TSEs or the KUs that will be placed on the form, and links to external IT systems can be established independently of all other activities.

Restrictions are placed on the development timeline only when a specific object depends on the existence of another: for instance a validation check involving two TSEs cannot be modelled until both involved TSEs have been defined and placed in a Form or in a TSE Group.

It is necessary to have a form before creating a Ts, a Generic TSE before creating a TSE Group, and other restrictions described in the following sections.

3.6.4.2 Transaction Service (TS) Edition page

Figure 43 shows the TS Edition page. This page is shown once the user selects a TS and he/she has the rights to update it. The Figure shows the sections unfolded for clearness purposes.

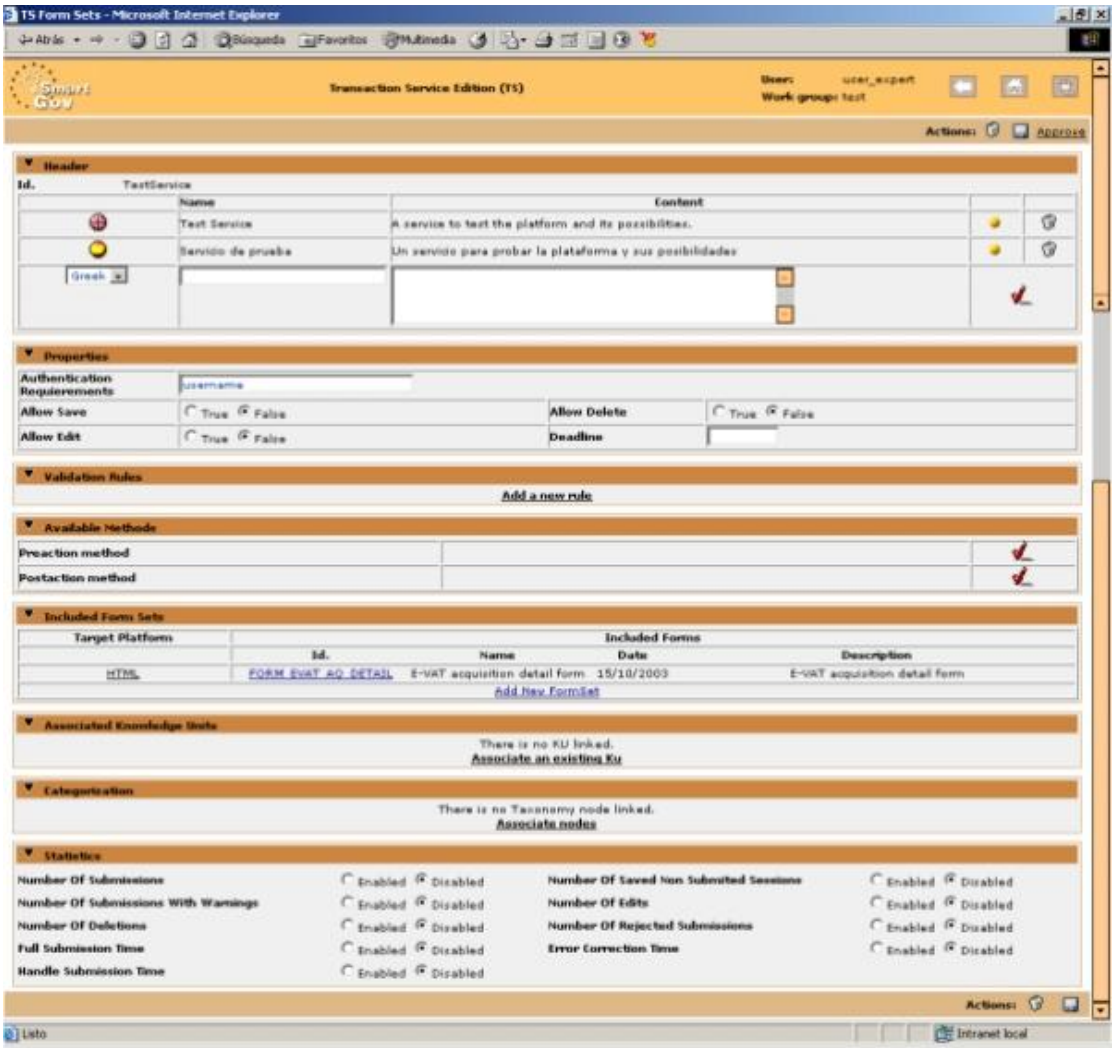


Figure 43 - TS Edition page

The following fields and actions may be identified:

Actions:

- Save
- Delete
- Approve (Depending of the role of the user and the state of the TS)
- Reject (Depending of the role of the user and the state of the TS)

Fields:

Header Section

- Id: The Id of the TS. If the TS is new, this field is updateable. The user should provide a TS identifier without spaces and special characters, such as hyphens, slashes, and so on. Notice that this Id should be human readable in order to easily search and locate the TSs afterwards.
- Description Table (Name and Content) – See Multi-lingual tables (3.3.1.3).

Properties Section

- Authentication requirements: The type of authentication that will be used to verify the identity of the service end-users. The username and password method is supported by default; all other methods should be supplied by the organisation's IT staff.
- Allow save
- Allow edit
- Allow delete
- Deadline: Defines when this TS stops being valid; after this date, the deployed TS will not be usable by end-users.

  Validation Rules Section: This section contains all the validations check to be performed when a service end-user submits a document. Please see section 3.6.2.2 for further information about Validation Rules.

  Available Methods Section: This section contains two methods: preaction and postaction, which enable SmartGov platform users to specify actions to be performed when launching the service (preaction) and when it is finished (postaction).

  Included form sets: This section enables the user to create form sets. A form set is a group of forms directed to a specific platform. The possibility of defining different form sets has been implemented as a future capability of the tool to generate services for different platforms (HTML, WAP…), although currently only "HTML" form sets are supported.

  When a new form set is created (clicking in "Add new Form set") or an already defined form set is modified (clicking in the name of the form set, in "Target Platform" column) a new page opens, allowing the user to select the forms to include in the form set (see figure 44).

Figure 44 – Form set definition

For a more detailed reference about managing tables, see section 3.3.1.4.

Associated knowledge units: KUs can be associated with a TS. The way of attaching KUs is explained in 3.3.1.7.

Categorization: The way of linking a TS or other SmartGov elements to Taxonomy Nodes is explained in 3.3.1.8.

Statistics: The definition of the desired delivery environment statistics for the TS. In this section a set of choices are displayed which enable users to activate or deactivate the collection of these statistics in the delivery environment.

Validations:

- Mandatory fields: TS Id, and at least one description (name and abstract) and one Form set.
- The TS id must not be in use by any other object in the SmartGov platform.

If the user does not have the privileges required to edit the TS, a read-only page will be shown (see figure 45), allowing him/her to view all the characteristics of the TS. The structure and fields are the same with the Edition Page.

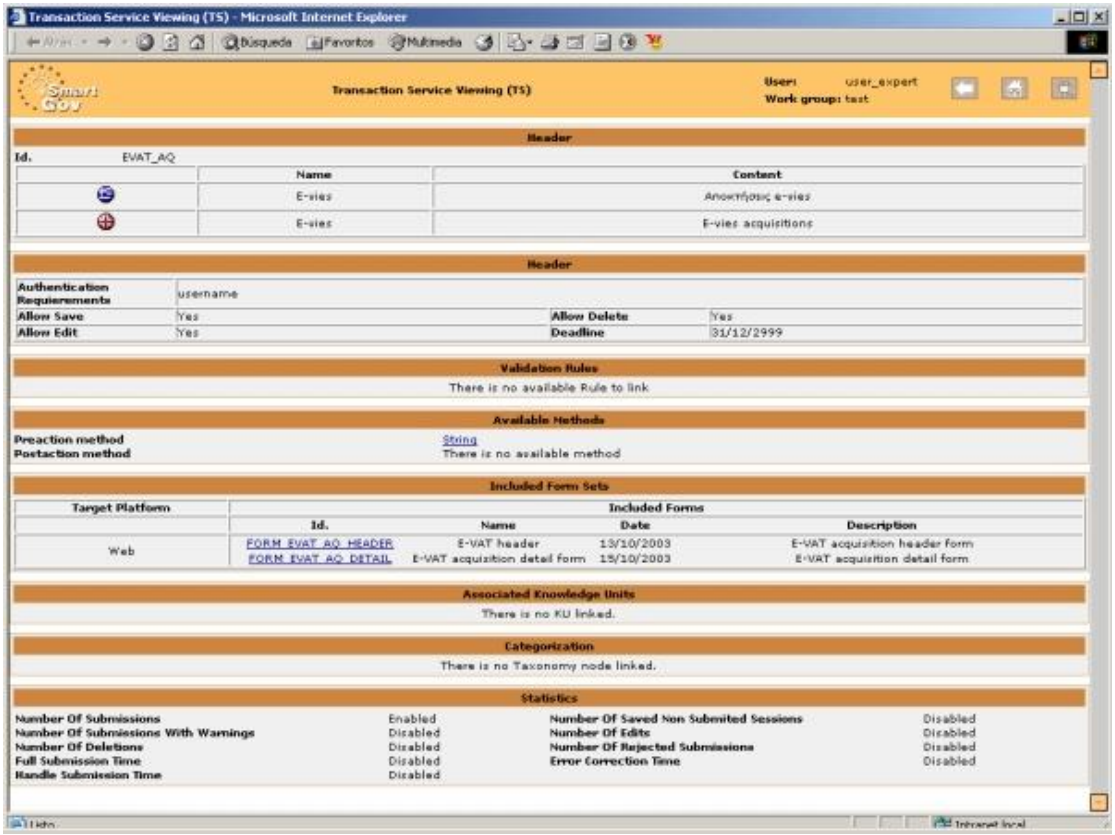


Figure 45 - TS read-only page

3.6.4.3 Forms

Forms are the basic presentation and interaction unit for the end user of the transaction service. In the context of the SmartGov platform, a form is divided in two parts:

1. The *semantic part*, which defines what information is entered in the form, the validation checks that apply to the form and the knowledge units, which will be presented to the user.
2. the *layout part*, which defines the appearance of the elements on the client device through which the electronic service is accessed.

Although in an ideal world both these parts would be developed in an integrated environment, in the context of the SmartGov project this is not feasible because (a) developing a web page editor with modelling power and user friendliness comparable to the commercial tools service designers are used to work with, is a huge task outside the scope of the project (b) devoting person power in development of such a module is not in line with the objectives of the key action (c) existing products are "closed" platforms and

cannot be extended. Taking these into account, the two parts will be developed independently as follows:

1. The semantic part is developed using the SmartGov development platform.
2. The layout part is developed outside the SmartGov platform using any appropriate tool for form layout definition that targets the dissemination channel through which the service will be delivered. For example, if the service will be delivered through the WWW, HTML form editors should be employed (e.g. DreamWeaver, FrontPage, vi etc.); if the service will be delivered through the WAP, a WAP page editor (3TL WBuilder, Rasquares Wap, vi etc) might be used. For services that will be deployed through multiple dissemination channels, appropriate form sets should be developed, one for each dissemination channel.

Since the two parts will be developed independently, there is a need to integrate them, by establishing links between the elements of the semantic part and the elements of the layout part. This procedure is covered in 3.7. The development of both parts should adhere to the results produced by the service process model creation phase, so these parts will be consistent with one another. Any inconsistencies between the semantic part and the layout part (such as a reference from the layout portion to a TSE or KU that does not exist) will be detected at the integration phase, and users will be advised on the actions that need to be taken to resolve the inconsistencies.

The key advantage of this separation is the independence between presentation (covered by the layout) and the logic (the Form element itself). In this way, it is very easy change the visual aspect of a complete service without modifying the logic, or reusing this logic, given that it is isolated and stored in the different elements (TSE, TSE Groups, Forms) that can be reused.

Figure 46 shows the Form Edition page. This page is shown once the user selects a Form and he/she has the rights to update it. The Figure shows the sections unfolded for clearness purposes.

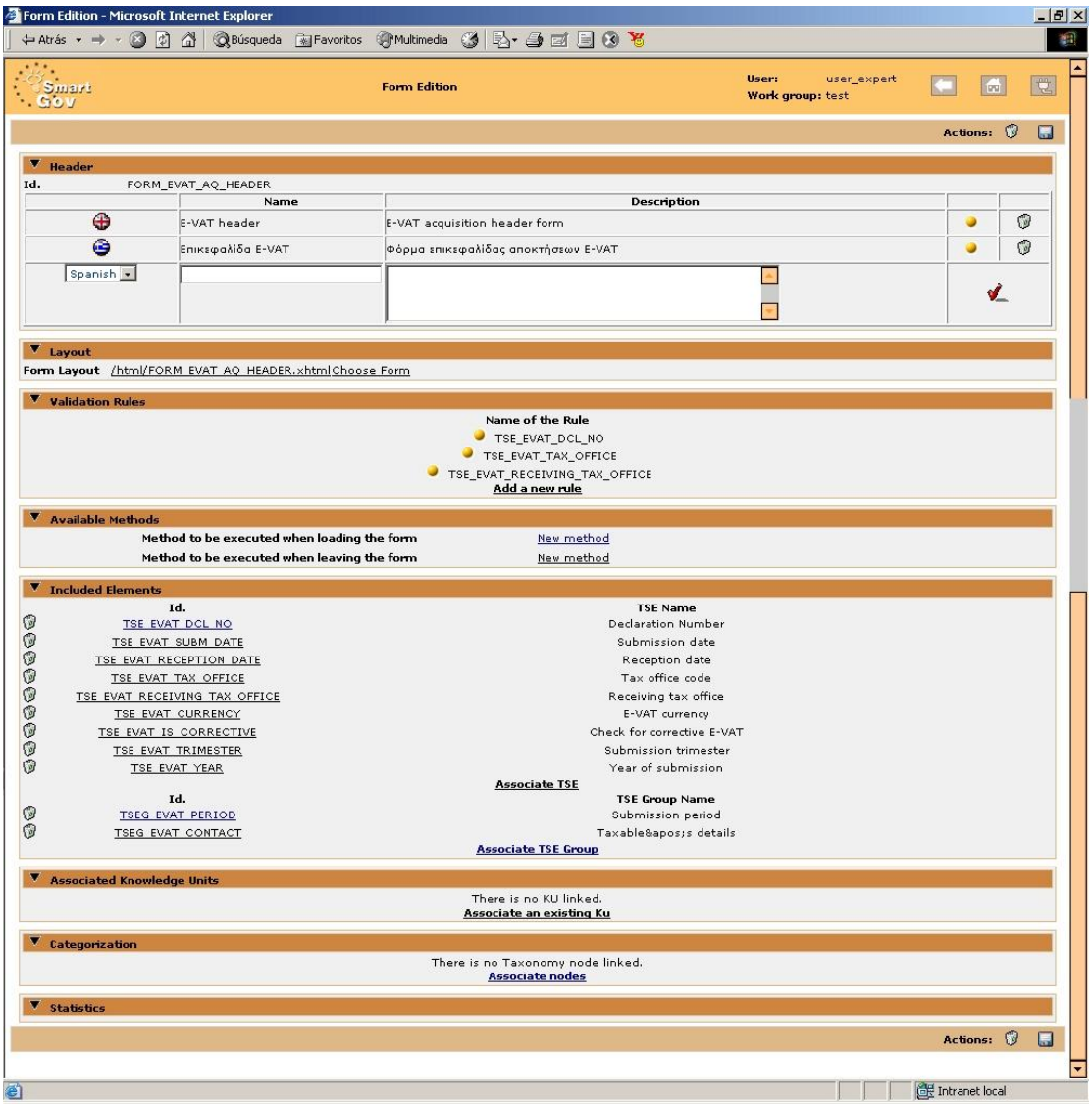


Figure 46 - Form Edition page

The following fields and actions may be identified:

Actions:

- Save
- Delete

Fields:

Header Section

- Id: The Id of the Form. If the Form is new, this field is updateable. The user should provide a Form identifier without spaces and special characters, such as hyphens, slashes, and so on. Notice that this Id should be human readable in order to easily search and locate the Forms afterwards.
- Description Table (Name and Description) – See Multi-lingual tables (3.3.1.3).

Layout Section: this section enables the user to select the layout file associated with this form. The file will be uploaded into the server where the application is installed (see section 3.3.1.9 for further details about uploading).

Validation Rules Section: This section contains all the validations check to be performed over the form, when the form is submitted. Please see section 3.6.2.2 for further information about Validation Rules.

Available Methods Section: This section contains two methods: the first one to be executed when the form is loaded, and the second one to be executed when the form is submitted.

Included elements: This section enables the user to include Instantiated TSEs and TSE Groups to the form, using the links "Associate TSE" and "Associate TSE Group". These links open new windows to select the elements to be included in the form. For a more detailed reference about managing tables, see section 3.3.1.4.

Figure 47 shows an example of the page to select Instantiated TSEs.

Figure 47 – Select ITSE to include in a form

Associate knowledge units: KUs can be associated to a Form. The way of attaching KUs is explained in 3.3.1.7.

Categorization: The way of linking a Form or other SmartGov elements to Taxonomy Nodes is explained in 3.3.1.8.

Statistics: The definition of the desired delivery environment statistics of the Form. In this section, a set of choices are displayed which enable users to activate or deactivate the collection of these statistics in the delivery environment.

Validations:

- Mandatory fields: Form Id, and at least one description (name and abstract). The layout file is not mandatory to save the form but a warning message will be displayed while no file has been selected.
- The form id must not be in use by any other object in the SmartGov platform.

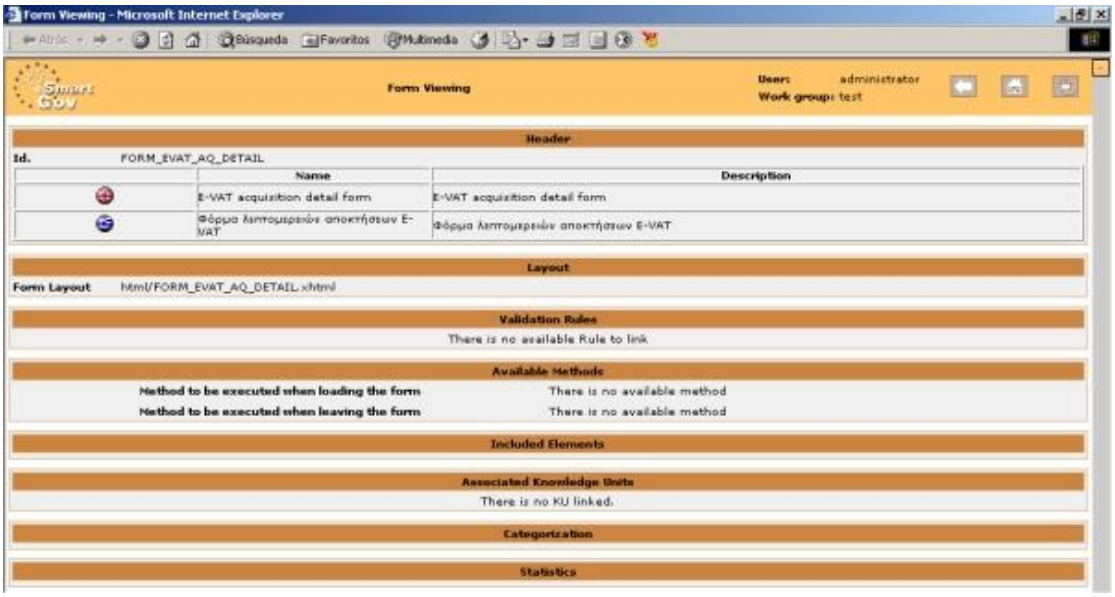


Figure 48 - Form Read-Only page

If the user does not have the privileges required to edit the form, a read-only page will be shown (see figure 48), allowing him/her to view all the characteristics of the form. The structure and fields are the same with the Edition Page.

3.6.4.4 Transaction Service Elements (TSEs)

Transaction service elements will be the basic building blocks for transaction services. TSEs will be mainly defined by domain experts, and their work will be complemented by IT staff, who will code the IT related portions, and by service workers, who may contribute by adding knowledge units that will serve as help items for the end-users of the transaction service.

Figure 49 shows the Generic TSE Edition page. This page is shown once the user selects a TSE and he/she has the rights to update it. The Figure shows the sections unfolded for clearness purposes.

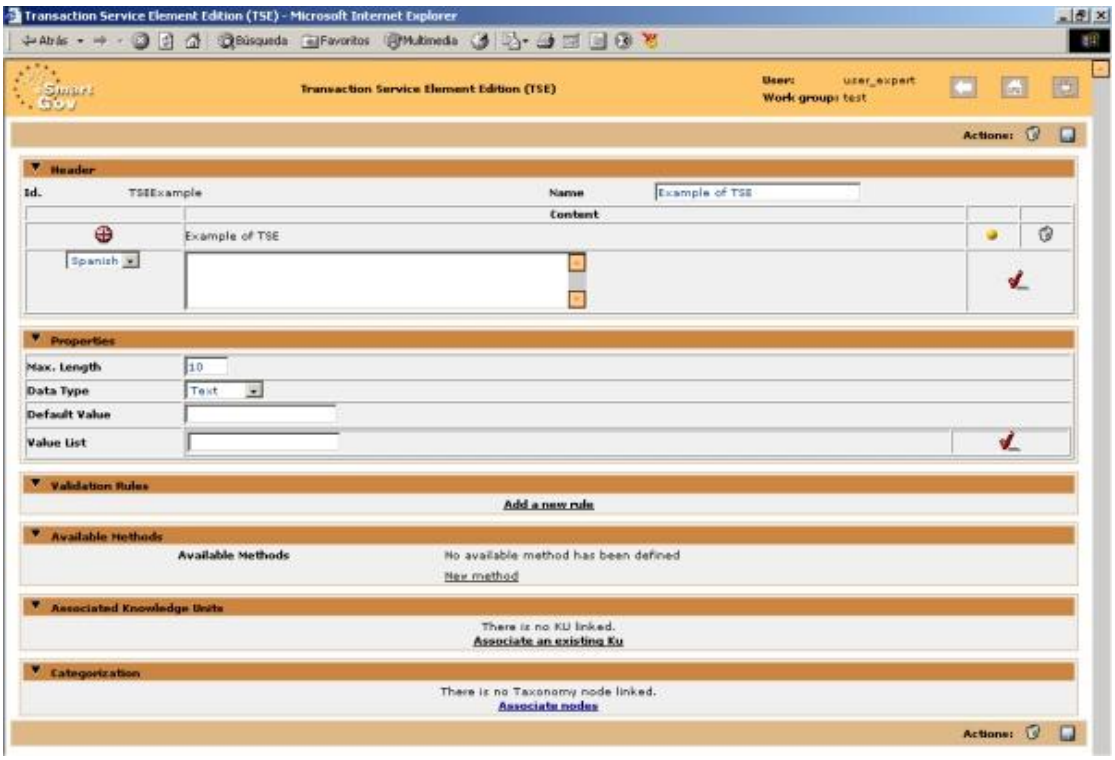


Figure 49 - TSE Edition page

The following fields and actions may be identified:

Actions:

- Save
- Delete

Fields:

Header Section

- Id: The Id of the TSE. If the TSE is new, this field is updateable. The user should provide a TSE identifier without spaces and special characters, such as hyphens, slashes, and so on. Notice that this Id should be human readable in order to easily search and locate the TSEs afterwards.
- Name: The name of the TSE.
- Content Table – See Multi-lingual tables (3.3.1.3).

Properties Section:

- Max. Length: The maximum allowed length for this TSE. Zero means unlimited length.
- Data type: The data type that this TSE will contain. The possible values are:
  - Currency
  - Date
  - Integer

- Boolean
- Text
- Real

- Value list: Provides a table to add possible values of the TSE, and allowing to specify the default value (see 3.3.1.4 for more details about table management).

Validation Rules Section: This section contains all the validations check to be performed over the form when submitted. Please see section 3.6.2.2 for further information about Validation Rules.

Available Methods Section: In this section, it is possible to add methods. This is a provision for future platform extension". In future, methods placed in this section (java code, SmartGovLang etc) may be included once and could be used in several validation checks.

Associate knowledge units: KUs can be associated with a Form. The way of attaching KUs is explained in 3.3.1.7.

Categorization: The way of linking a Form or other SmartGov elements to Taxonomy Nodes is explained in 3.3.1.8.

Validations:

- Mandatory fields: Tse Id and data type.
- The TSE id must not be in use by any other object in the SmartGov platform.

Figure 50 - TSE Read-Only page

If the user does not have the privileges required to edit the Generic TSE, a read-only page will be shown (see figure 50), allowing him/her viewing all the characteristics of the Tse. The structure and fields are the same with the Edition Page.

3.6.4.5 Instantiated Transaction Service Elements (ITSEs)

The ITSEs represent instances of Generic TSEs. These instances are created to include the TSEs in forms, allowing the user to modify or adjust some of the characteristics of the TSE. Therefore, the structure of the ITSE is very similar to the TSE's, as the figure 51 shows. This Instantiated TSE edition page is shown once the user selects an Instantiated TSE and he/she has the rights to update it. The Figure shows the sections unfolded for clearness purposes. Given that the structure is very similar to the previous paragraph, only the differences will be commented.

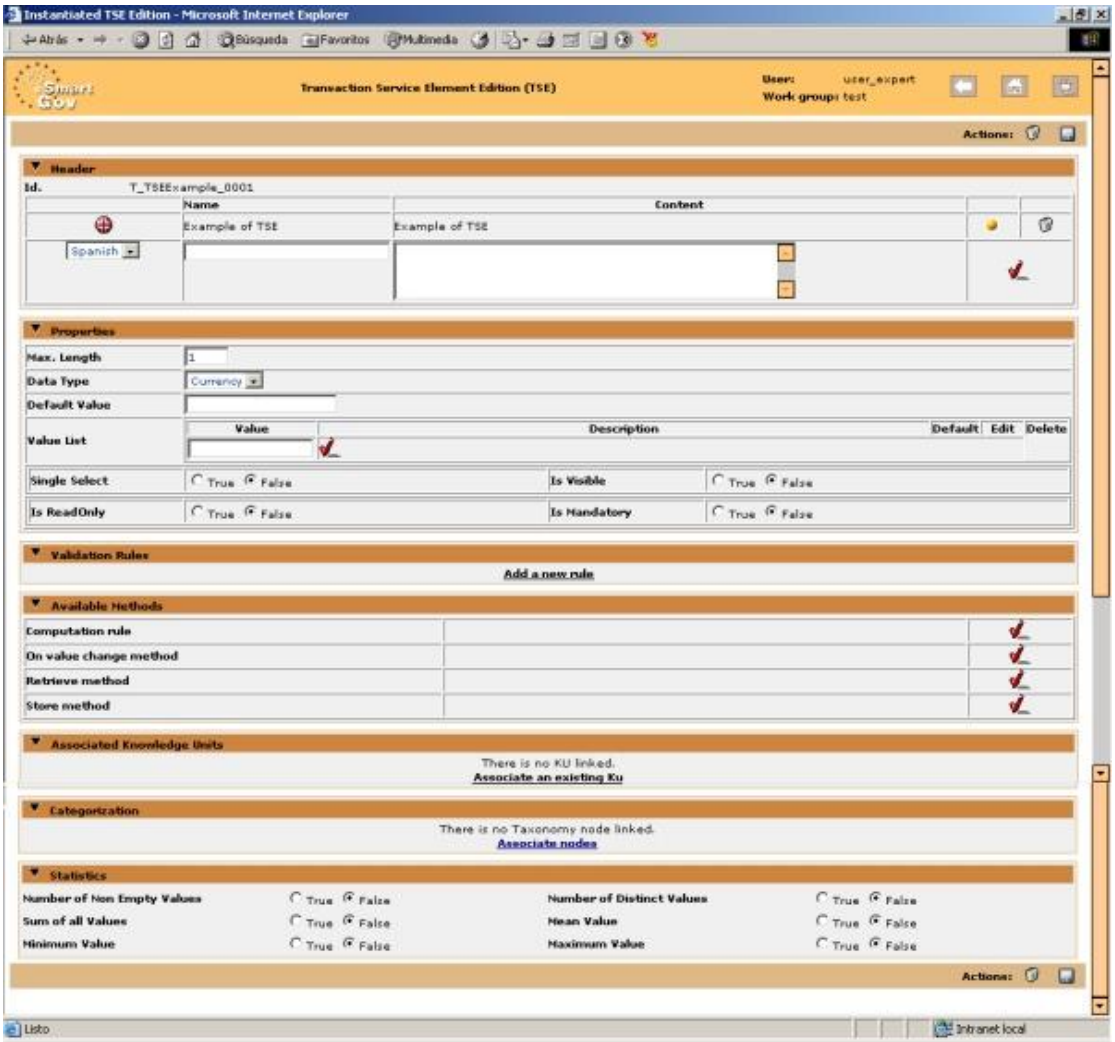


Figure 51 – Instantiated TSE Edition page

The following fields and actions may be identified:

Fields:

Header Section

- Id: The Id of the ITSE. This id is generated using as base the Id of the Generic TSE used to instantiate it. A "T_" prefix is added if the ITSE is instantiated from a Generic TSE, while a "G_" prefix is added if the ITSE is instantiated from a Generic TSE Group.
- Description Table (Name and content) – See Multi-lingual tables (3.3.1.3).

Properties Section:

- Single select: For TSEs that present a list of values to the end-user, if this flag is "true", only a single value may be selected; if this flag is "false", multiple values may be selected.
- Is visible: designates whether the TSE will be visible by the end-user or will remain hidden.
- Is read-only: if this flag is "true" the end-user will not be able to modify the TSE value; if this flag is "false", modifications will be allowed.
- Is mandatory: if this flag is "true" and the end-user provides no value for this TSE, an error will be flagged; if this flag is "false", provision of a value is not mandatory.

Available Methods Section: In this section is possible to specify four methods:

- Computation rule: a method to compute the value of this field.
- On value change: What to do when the value of this ITSE changes.
- Retrieve method: how to load its value when the form where it is included is loaded.
- Store method: how to store its value when the form where it is included is submitted.

Statistics: The definition of the desired delivery environment statistics of the ITSE. In this section are displayed a set of flags which enable users activate or deactivate the collection of these statistics in the delivery environment.

Validations:

- Mandatory fields: all pre-filled when instantiating the object.

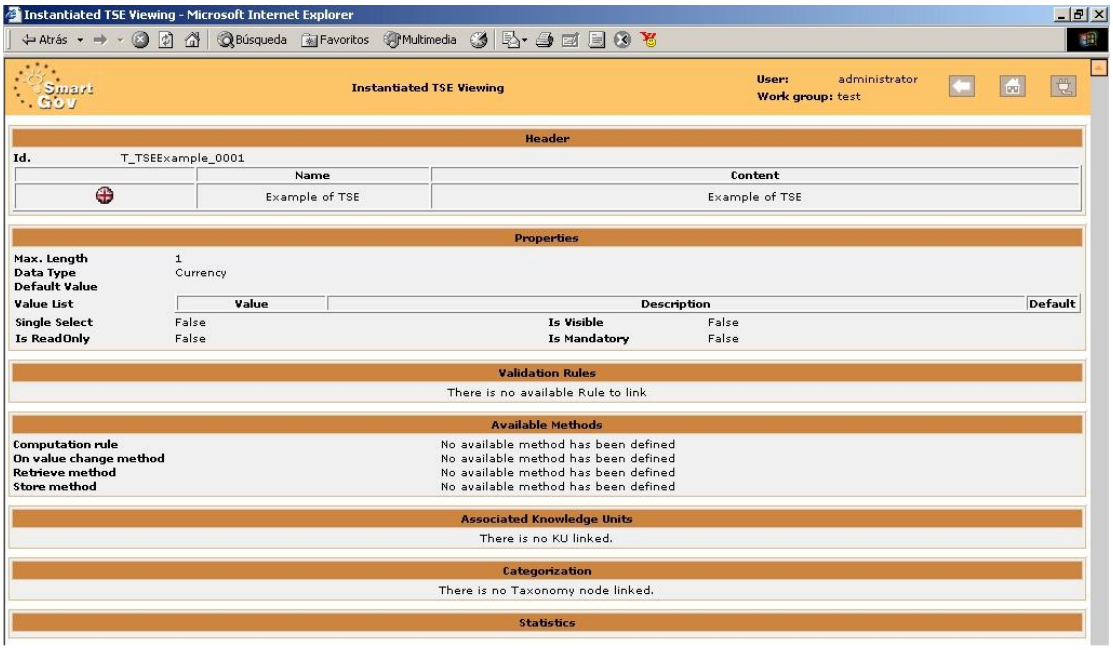


Figure 52 - ITSE Read-Only page

If the user does not have the privileges required to edit the Instantiated TSE, a read-only page will be shown (see figure 52), allowing him/her to view all the characteristics of the ITSE. The structure and fields are the same with the Edition Page.

3.6.4.6 Group of Transaction Service Elements (TSE groups)

With layout definition being developed outside the SmartGov platform, a TSE group defines the following:

- a set of TSEs that appear together within services
- Repetition information, indicating whether only one instance or multiple instances of the member TSEs is required. For groups allowing multiple instances, the member TSEs actually form a *table row*, which is repeated as many times as needed, and may be used to model "detail" sections, e.g. the items that are included in an order along with their prices, the customers of an enterprise together with the net value and the tax due for the transactions conducted with each one etc. The repetition information may indicate the initial, minimum and maximum number of instances and the step for adding new rows in the group.
- Validation checks that must hold among the elements of this set
- Knowledge units that apply to the set of TSEs, rather than to individual elements (e.g. for a TSE group representing a citizen's identification data,

a KU containing the law that states which information is considered as "required identification data" may be defined)

It is worth noting that a single TSE may participate in more than one TSE group, thus the relationship between TSEs and TSE groups is of cardinality "many-to-many". For instance, the identification number of a citizen may appear in the TSE group "Personal Identification Data" and in the TSE group "Page footer", which can be placed on the bottom of a page to provide an immediate reference to the service context. TSE groups may not be nested, i.e. a TSE group may only contain individual TSEs, not TSE groups. This restriction leads to a more comprehensible and easy-to-manage framework for SmartGov platform users to work in, while it does not downgrade the platform functionality since (a) the same result may be obtained by adding the individual TSEs belonging to the source group to the target group and (b) the cases that such a functionality will be needed will be –if existing at all- rare.

TSE groups with no repetition requirements are not an indispensable element of the SmartGov platform; they are provided for convenience purposes, since the working team will be able to package in a single entity all the necessary information for TSEs that usually appear together. Determination of whether a set of TSEs should be packed in a group with no repetition specification should follow some "rules of thumb", such as "if some TSEs will be frequently used together, it will be beneficial if they were grouped together once and used thereafter as a single entity".

Figure 53 shows the Generic TSE Group Edition page. This page is shown once the user selects a TSE Group and he/she has the rights to update it. The figure shows the sections unfolded for clearness purposes.

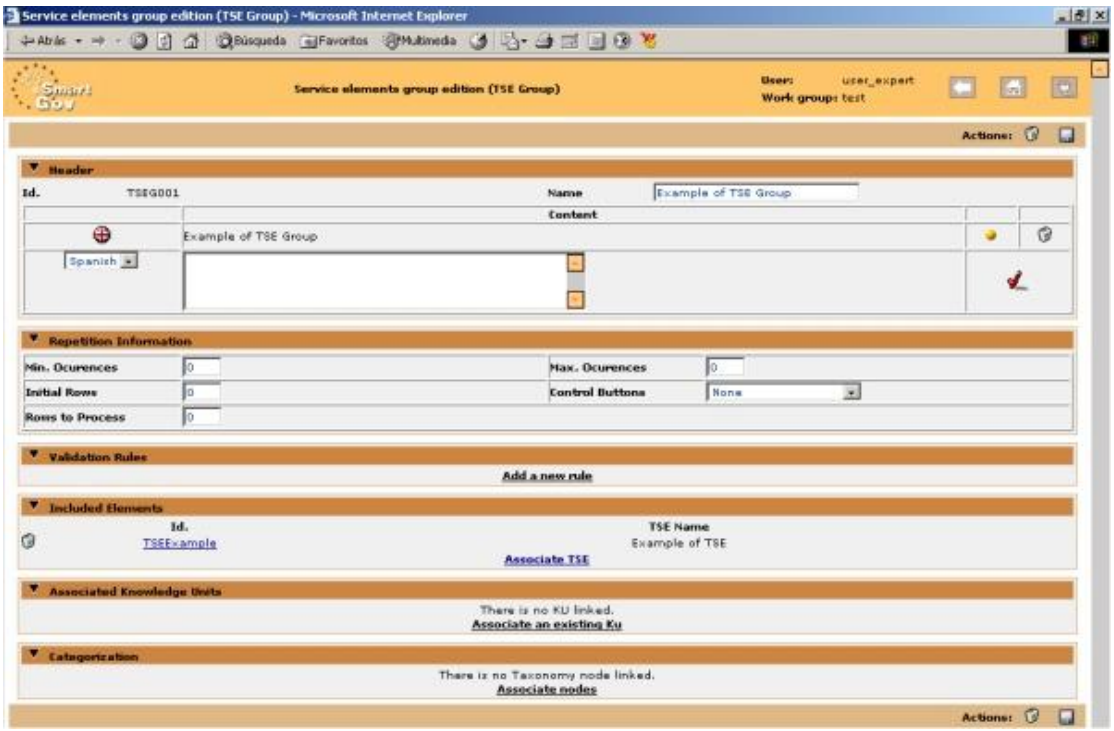


Figure 53 - TSE Group Edition page

The following fields and actions may be identified:

Actions:

- Save
- Delete

Fields:

Header Section

- Id: The Id of the TSE Group. If the TSE Group is new, this field is updateable. The user should provide a TSE Group identifier without spaces and special characters, as hyphens, slashes, and so on. Notice that this Id should be human readable in order to easily search and locate the TSE group afterwards.
- Name: The name of the TSE.
- Content Table – See Multi-lingual tables (3.3.1.3).

Repetition Information Section:

- Min Occurrences: The minimum number of rows for the TSEG.
- Max. Occurrences: The maximum number of rows for the TSEG.
- Initial rows: number of rows shown when the TSEG is loaded.
- Rows to process
- Control buttons: this field enables the user to specify the type buttons that must be added to the group in order to manage the number of rows. The possible values are:

- None
- Delete rows
- Add rows
- Delete and Add rows

Validation Rules Section: This section contains all the validations check to be performed over the form when submitted. Please see section 3.6.2.2 for further information about Validation Rules.

Included elements: The TSEs included in this TSE Group.

Associate knowledge units: KUs can be associated to a Form. The way of attaching KUs is explained in 3.3.1.7.

Categorization: The way of linking a Form or other SmartGov elements to Taxonomy Nodes is explained in 3.3.1.8.

Validations:

- Mandatory fields: Tse Group Id, name and one content. At least one element must be included.
- The TSE group id must not be in use by any other object in the SmartGov platform.

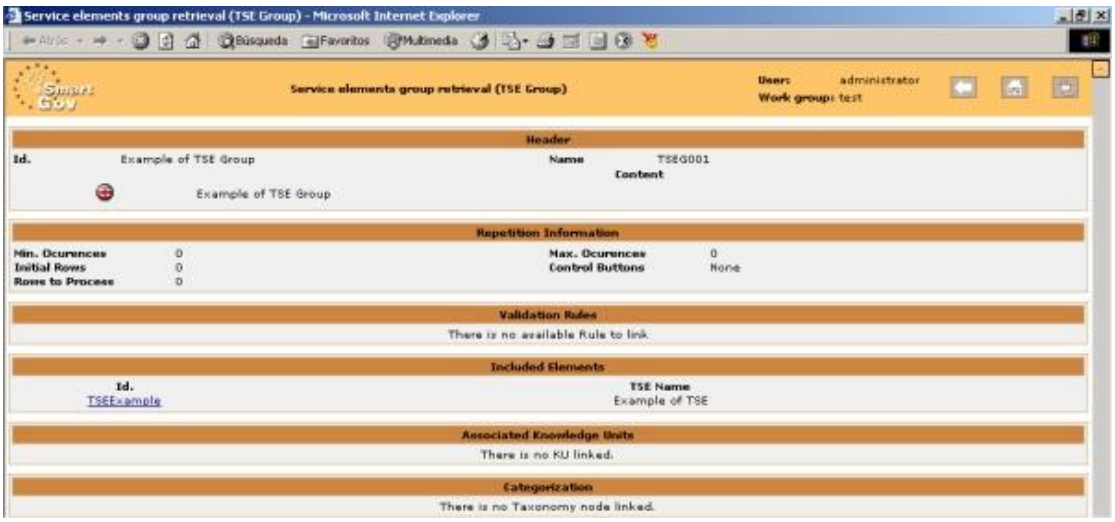


Figure 54 - TSE Read-Only page

If the user does not have the privileges required to edit the Generic TSE group, a read-only page will be shown (see figure 54), allowing him/her viewing all the characteristics of the group. The structure and fields is the same that for the Edition Page.

3.6.4.7 Instantiated Group of Transaction Service Elements

The Instantiated TSE Group represent instances of Generic TSE Groups. These instances are created to include the groups in forms, allowing the user to modify or adjust some of their characteristics. Therefore, the structure of the ITSEG is very similar to the TSE Group's, as the figure 55 shows. This Instantiated TSE group edition page is shown once the user selects an Instantiated TSE Group and he/she has the rights to update it. The Figure shows the sections unfolded for clearness purposes. Given that the structure is very similar to the previous paragraph, only the differences will be commented.

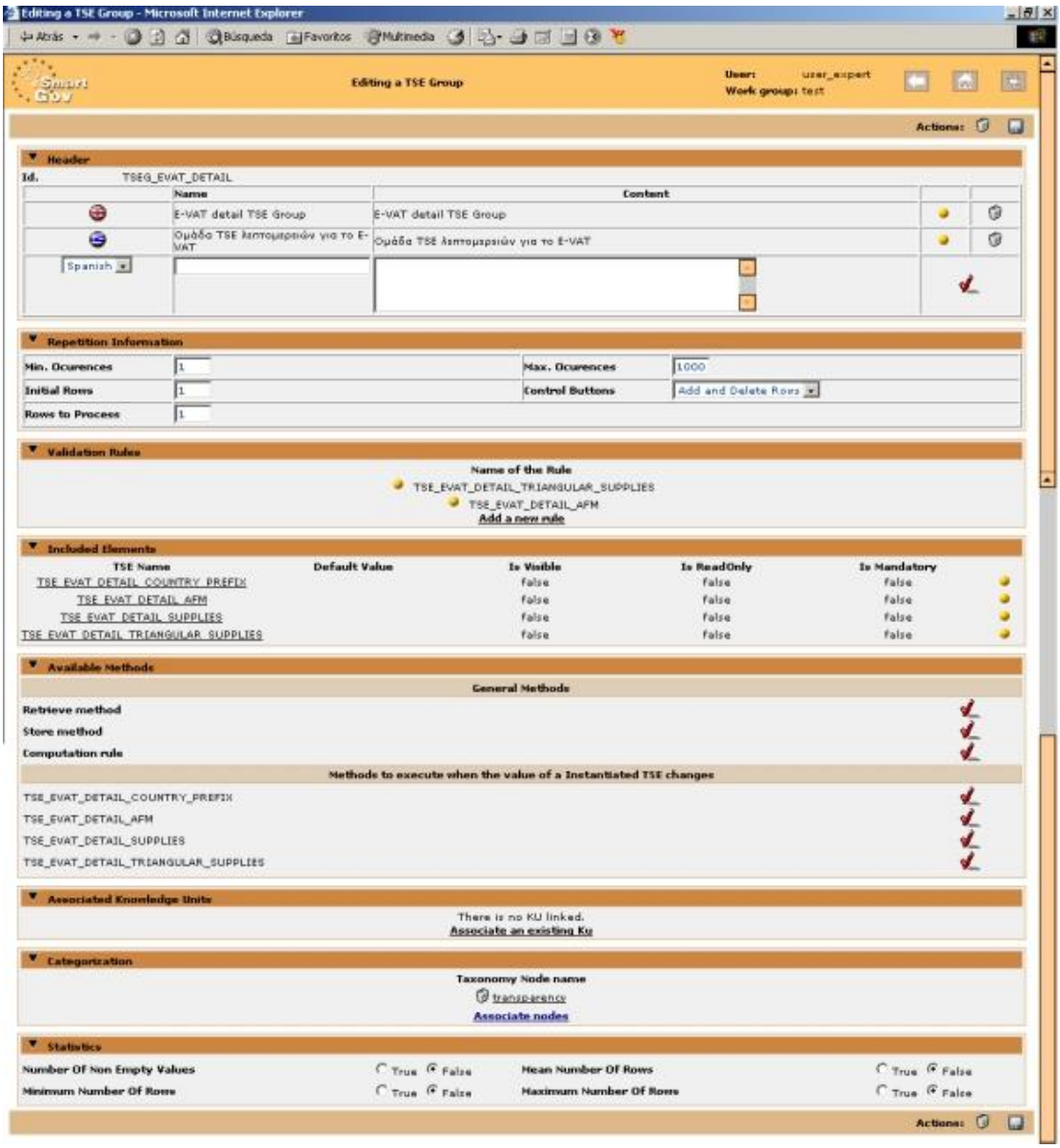


Figure 55 – Instantiated TSE Group Edition page

The following fields and actions may be identified:

Fields:

Header Section

- Id: The Id of the ITSE Group. This id is generated using as base the Id of the Generic TSE Group used to instantiate it. A numeric suffix is added to distinguish the different instances of a TSE Group. If the Instantiated TSE Group has been created outside the Front-end, its name may follow different rules.
- Description Table (Name and content) – See Multi-lingual tables (3.3.1.3).

Included elements Section: This section is very similar to the section in the TSE Group, but in this page the list elements are Instantiated TSEs, not generic TSEs.

Available Methods Section: In this section is possible to specify three general methods:

- Retrieve method: how to load the values of the different TSEs when the form where it is included is loaded.
- Store method: how to store the values of the different TSEs when the form where it is included is submitted.
- Computation rule: a method to compute the values of the TSEs.

In addition to these general methods, specific methods to execute when the value of an ITSE changes can be defined.

Statistics: The definition of the desired delivery environment statistics of the ITSE Group. In this section a set of choices are displayed enableing users activate or deactivate the collection of these statistics in the delivery environment.

Validations:

- Mandatory fields: all pre-filled when instantiating the object.

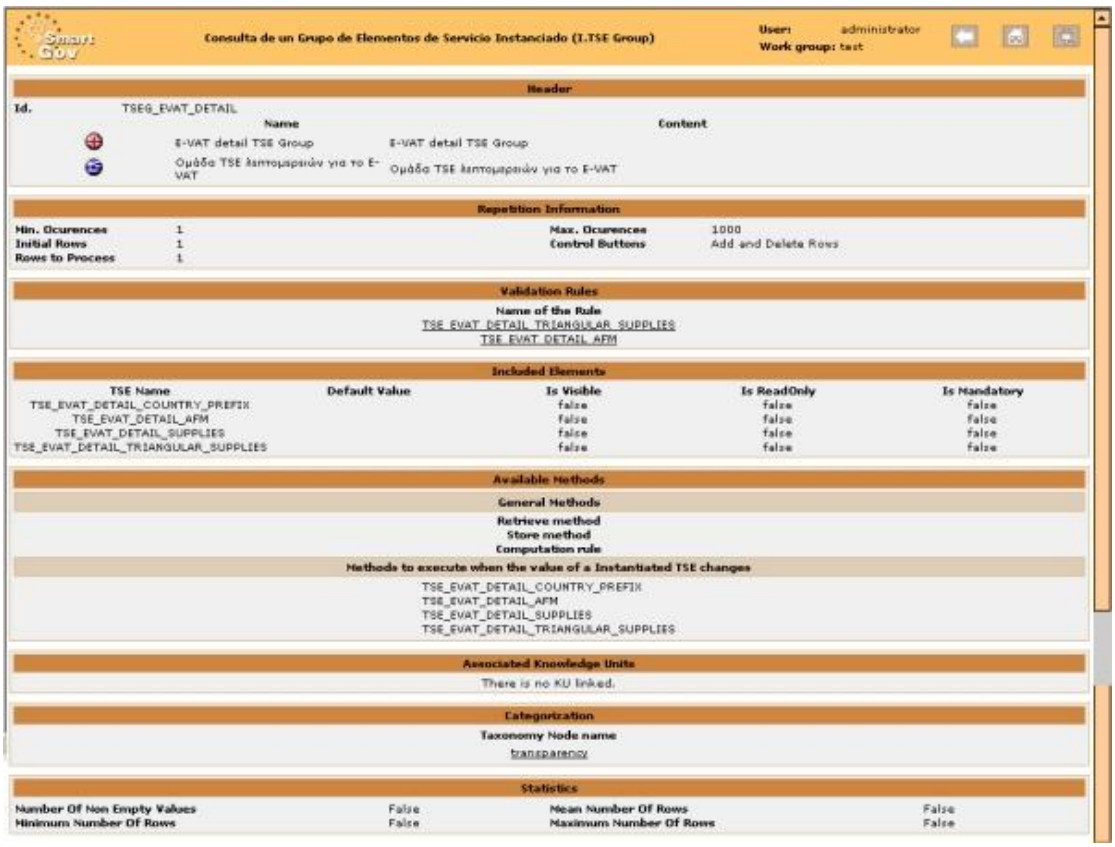


Figure 56 - ITSE Group Read-Only page

If the user does not have the privileges required to edit the Instantiated TSE Group, a read-only page will be shown (see figure 56), allowing him/her viewing all the characteristics of the ITSE Group. The structure and fields is the same that for the Edition Page.

3.7 Establishing links between the form visual elements and SmartGov semantic elements

A form participating in a SmartGov service essentially combines two facets:

1. the visual part, comprising of XHTML elements
2. the semantic part, consisting of links to SmartGov objects, such as KUs, TSEs, TSE groups etc.

These two facets must be integrated in a way that is (a) easy and intuitive for the domain experts to use, with basic only technical skills and (b) is possible to be sequentially processed in order to produce the final service forms, together with the accompanying code. Moreover, it is highly desirable to produce high-quality forms, in order to make the service attractive to the users if its target group.

Taking these facts into account, the SmartGov project has specified a procedure for extending one of the most popular HTML editors, namely DreamWeaver MX, to allow for the integration step to be performed easily by domain experts that only have basic skills in HTML page editing. According to this procedure, domain experts use DreamWeaver MX to specify the associations between visual elements of the XHTML forms and SmartGov platform items. Domain experts select through a click-and-drag procedure the visual elements and then select the associated SmartGov item through intuitive dialog boxes. When these selections have been made, DreamWeaver MX formulates a proper *custom tag* that uniquely identifies the SmartGov platform item, and embeds this tag into the XHTML code. Upon service generation the integrator module recognises these custom tags and arranges for retrieving the information pertaining to the relevant SmartGov platform items from the SKDB and appropriately enhancing form functionality.

More specifically, the complete procedure comprises of the following steps:

1. domain experts populate the SKDB with SmartGov platform items (transaction services, forms, TSE groups TSEs and KUs), using the SmartGov front end. It is important that all links between SmartGov platform items have been established i.e.:
    a. the transaction service refers to all forms (through the *available form sets*) and all KUs it contains
    b. each form description is complete in regard to the KUs it is associated with and TSE groups and TSEs appearing on the form
    c. each TSE group refers to all TSEs and KUs it contains

d. each TSE description is complete in regard to KUs the TSE is associated with

This step is accomplished as described in previous portions of this manual.

2. domain experts, possibly assisted by the IT staff, prepare the XHTML forms for the service. This is accomplished using standard commercial off-the-shelf tools such as Macromedia DreamWeaver, Microsoft FrontPage etc.
3. the IT staff exports SmartGov items (KUs, TSEs, TSE groups etc) from the SKDB into appropriately formatted XML files. These files are installed in predefined locations, in order to be accessible by the DreamWeaver MX environment. File installation is also performed by the IT staff.
4. the domain experts establish the links between the visual XHTML entities and the semantic items of the SmartGov platform by highlighting first the desired XHTML entities and then selecting the SmartGov item that the highlighted elements should be linked to. XHTML entity highlighting is performed through the standard "click-and-drag" methodology of window-based environments, whereas the selection of the SmartGov platform items is performed via a tree-structured index that may correspond to the organisational taxonomy that has been entered in the SmartGov platform or, alternatively, to the Service/form set/form hierarchy which is used by the SmartGov development environment. It is also possible that both selection paths may co-exist, and the users can make use of the one more suited to their preferences.

In the following paragraphs, steps (2), (3), (4) and (5) will be covered in detail, since step (1) has been discussed in previous portions of this manual. The actual procedure for addressing step (2) is documented in the relevant tool's manual, however some issues on preparing forms to be used in the context of electronic services developed using the SmartGov platform are presented.

3.7.1 Preparing the HTML forms

An XHTML form to be used within a SmartGov platform service may be prepared as any other XHTML form, using an HTML editor. However, the form design should cater for all phases of the user interaction with the service i.e.:

1. *data input*. Input areas should be provided with appropriate labels. Note here that since SmartGov services are multilingual, simple provision of text labels is not sufficient.
2. *access to help*. Anchors from where the user can access the on-line help texts should be provided
3. *navigation/submission*. Widgets that will allow the user to navigate between forms and submit the document should be made available
4. *system error output.* Values provided by the user are validated by the system and, in case of errors appropriate errors are emitted. The designers should reserve space on the form for these error messages to be displayed.
5. *dynamic group expansion and shrinking.* For repeating groups, in particular, certain controls have to appear on the form to enable the user to add and delete rows.

   Important note: for repeating groups (i.e. TSE groups whose elements may be cloned multiple times) only one instance of the elements should be placed on the form, as shown in Figure 57. The Integrator module will cater for producing code that will allow addition and deletion of rows dynamically, during service execution.

Figure 57 presents an example of a form designed for a SmartGov service in the DreamWeaver MX design environment, while Figure 58 presents the same form rendered in a browser. On the top, a short title (VIES ACQUISITIONS) and a long description (Form for VIES acquisitions) of the form appear. The question mark icon appearing on the right of the short title is intended to serve as an anchor for accessing help associated with the form. Similarly, the other question marks on the form provide anchors for accessing help on the items appearing on their left.

Below the general information on the form, a line appears displaying the taxable entity's VAT id, which is effectively a TSE. On the line we can identify two portions:

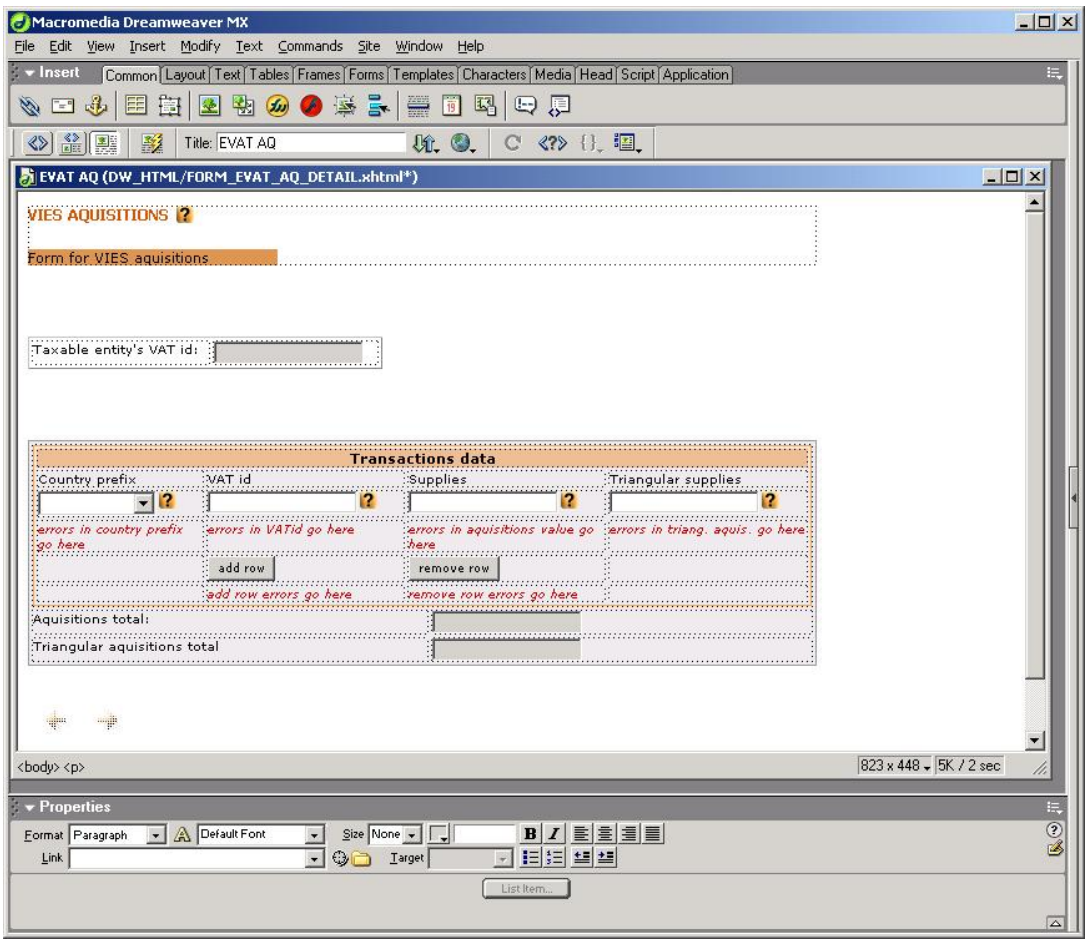


Figure 57 - A SmartGov form designed in the DreamWeaver MX environment

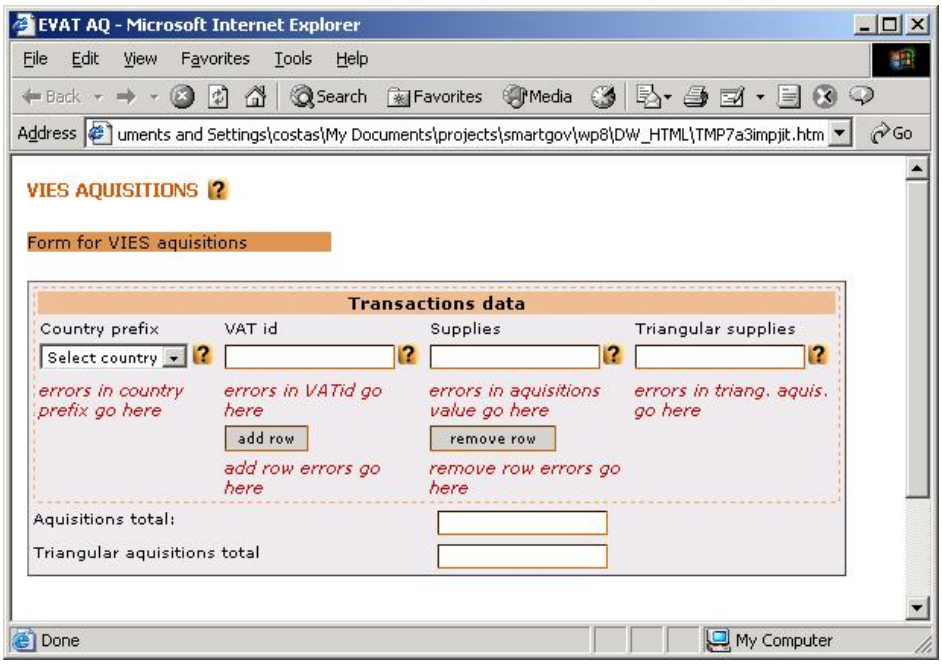


Figure 58 – The SmartGov form rendered in a browser

1. *the TSE label* (the text *Taxable entity's VAT id*) and
2. *the TSE value area* (the box on the right of the label)

Since this TSE will have a pre-populated value, no validation checks will be associated with it and no errors will be emitted for its value; therefore, there is no need to allocate space for an error message.

SmartGov good practice tip 1:

Place labels on the left of the form, values on the right. If many TSEs appear on the same form, use a two-column table placing labels on the left column and input areas on the right. The table helps keeping labels and input areas aligned. Place help anchors, if any, on the right of the respective input area.

If a field is bound to emit validation errors, allocate space either on the right of the input area (expanding the table to a three-column one) or immediately below (or above) the input area.

The *Transaction data* area is actually the space that the user will type values in. In SmartGov terminology, this area hosts a TSE group with four TSEs, namely the country prefix, the VAT id, the value of the supplies and the value of triangular supplies. The group has a generic label (Transactions data) and is organised in a four-column table (one column per TSE), with each column having a column label (the TSE short name). The two following rows of the TSE provide the input areas in which the service user will provide the values (first row) and the space in which relevant errors will be reported (second row). The third and fourth row, respectively, host the controls for adding and removing rows from the group and for displaying errors that may occur upon row addition and deletion (e.g. while trying to remove rows from an empty group).

SmartGov good practice tip 2:

Always use tables when entering repeating groups. Use as many columns as the number of TSEs within the group and place error report areas directly beneath the value input area. If the group contains too many TSEs to fit in a single row, place firstly TSEs in the first row until no more space is left, then insert a new row into which error report areas for the newly placed TSEs will be hosted. Repeat the process by adding row pairs, until no more TSEs are left within the group.

Controls for adding and removing lines should be placed at the bottom of the group.

The two following lines display summary information for the group, allowing the user to view the sum of the declared transactions value and triangular transactions value. Since these fields are automatically calculated, only the label and number appear, with no provision for error reporting areas.

SmartGov good practice tip 3:

Keep group summary data as close as possible to the bottom of the pertinent group. Use descriptive labels for them and, whenever possible, align them with the columns they report summary data on.

Finally, at the bottom-left part of the form, two navigation arrows appear allowing the user to move to the previous form (left arrow) and to the next form (right arrow).

SmartGov good practice tip 4:

Form navigation controls should appear on the bottom of the form, either on the left or on the right.

Once the HTML form has been prepared using the guidelines presented above, the link establishment procedure may commence.

3.7.2 Data export and file installation

The data export and file installation procedure step is performed by the IT staff. During this phase the SmartGov XML repository is queried to retrieve descriptions and identities of SmartGov platform objects. The data retrieved is formatted as needed for use by the DreamWeaver tool and installed in the appropriate location. Data retrieval and formatting are performed using the `dwexport.jar` Java archive, which contains all the appropriate functionality. In order to initiate the export and installation procedure, the following commands should be executed:

```
set CLASSPATH=dwconvert.jar;xmlstore-2.0.0.jar;xmlstoreapi-2.0.0.jar
java XMLToDWConverter repositotyPropFile DWeaverInstallationPath folderName locale
```

Note that in the first command the Java archives (jars) providing the implementation of the export functionality and the XML store are referenced; these may need to be replaced with the full java archive pathname, if they do not reside in the current directory. In the second command, parameters are as follows:

1. `repositoryPropFile`: the settings for the XML repository property file to be queried.
2. `DWeaverInstallationPath`: the location where DreamWeaver MX is installed on the system. Care should be taken if the installation path contains spaces e.g. `C:\Program Files\Macromedia\Dreamweaver MX`, in which case it must be enclosed in double quotes, i.e. entered as `"C:\Program Files\Macromedia\Dreamweaver MX"`
3. `folderName`: The name of the folder into which SmartGov tag content will be placed. The value “SmartGov” is recommended

4. `locale`: the SmartGov Services and Knowledge repository holds multilingual resources for various elements of the SmartGov entities, such as names, descriptions etc. During the export procedure, the locale that will be used during the form design procedure is specified (`en` for English, `el` for Greek `es` for Spanish etc). This should be set to match the preferences of the expert working in the link establishment procedure.

   Please note that the specification of a single locale affects *only* the DreamWeaver MX environment, and *does not* restrict the multilinguality capabilities of the running service.

Alternatively to invoking the export procedure through the command-line, users may use the graphical front-end to accomplish the same task. Figure 59 displays the SmartGov DreamWeaver Integration graphical front-end. The user should provide appropriate values for the form fields, which correspond to the parameters of the command-line version. When all form fields have been entered, the "Export" button should be pressed to initiate the export process. The "Export messages" area displays information regarding the progress of the export process.

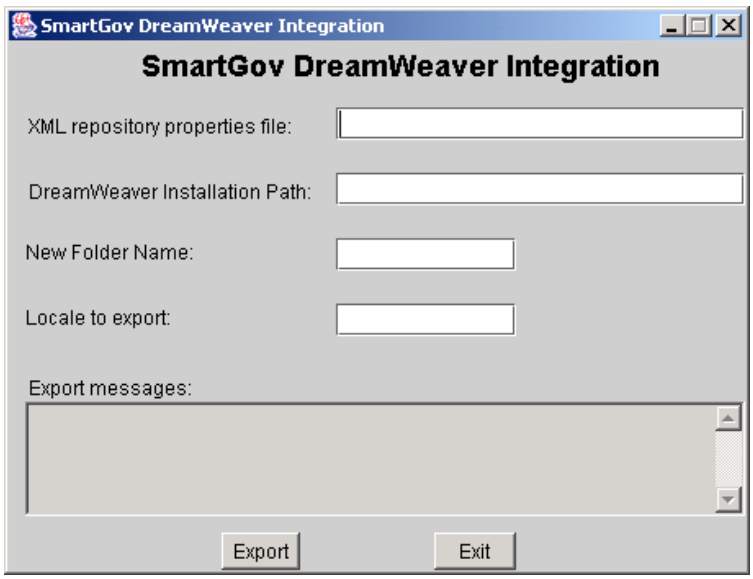


Figure 59 – Graphical front-end for the export procedure

Upon completion of the execution of this command, the tags related to the form design for the service "serviceName" have been installed in DreamWeaver and are ready for use.

The final step required is to enable the usage of SmartGov tags in HTML documents. This can be accomplished through the "Edit/Tag libraries…" menu, selecting the "SmartGov Site Tags" folder from the upper pane, checking the "HTML" control in the lower pane and finally clicking "OK".

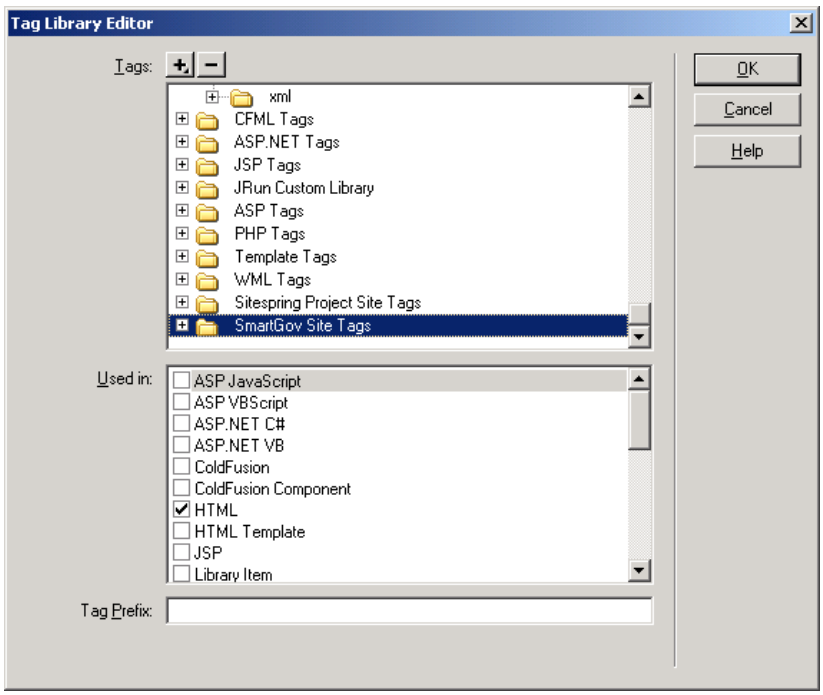


Figure 60 - Enabling the use of SmartGov tags

Notes to system administrators:

1. The export procedure generates tags in a single locale, in order to minimise user confusion caused by a large number of offered selections. If multiple users on the same machine need to work using different locales during the same period, this can be accomplished provided that:
   a. A multi-user OS is installed on the computer (Windows NT, Windows 2000 or Windows XP)
   b. The different users use different accounts to log into the computer.
   c. The files are *not* placed into the DreamWeaver MX installation directory but rather in each *user's personal DreamWeaver MX configuration folder.* For Windows NT systems this folder is usually located at

   C:\WinNT\profiles\<username>\Application Data\Macromedia\Dreamweaver MX\Configuration

   while for Windows 2000 and Windows XP systems this folder is located at

   C:\Documents and Settings\<username>\Application Data\Macromedia\Dreamweaver MX\Configuration

   For more information, please refer to your DreamWeaver MX documentation.
2. The case of the SmartGov tags should not be altered for the linkage procedure to work properly. To ensure that no conversion occurs, please verify that the DreamWeaver MX preferences do not specify forceful tag case conversion. From the Edit menu select "Preferences", select the

"Code format" category and verify that the "Override case of tags" checkbox is clear (this is the default setting).

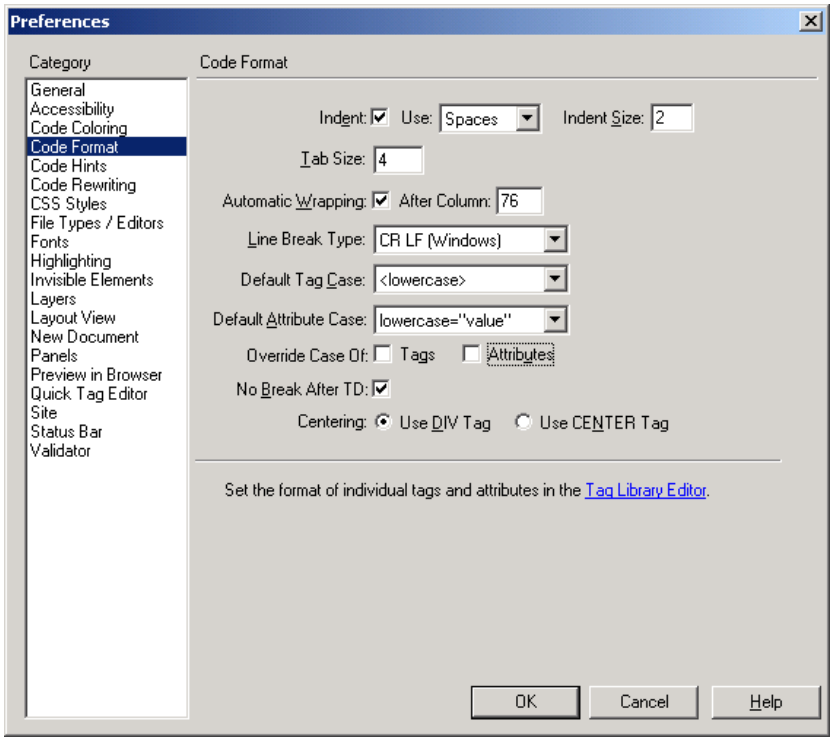


Figure 61 – Code format preferences dialog

3.7.3 Link establishment

Once the "data export and file installation" step has been completed, the DreamWeaver MX tag library will have been enriched with tags corresponding to the SmartGov platform objects. These tags must be placed on the form, *replacing the visual elements placed in the initial form design*, in order to allow the Integrator module to create the application the final service. Deletion of existing visual elements can be done by highlighting individual elements and pressing the "Delete" key (or selecting "Edit/Clear" from the menu). Tag insertion can be performed through the "Insert/Tag" menu of the DreamWeaver MX.

SmartGov good practice tip 5:

The appearance of the form will be distorted in this stage, it is thus advisable to create a copy of the original form design and work with this copy.

Figure 62 presents an example of the enriched DreamWeaver MX "Insert tag" menu. A new top-level folder entitled "SmartGov site tags" is now available, which will contain one entry for each service created using the SmartGov front-

end ("Income tax service" in the example screenshot). Each service folder contains additional subfolders for each form within the service, and each of these form folders will contain additional subfolders for the TSEs and KUs that are linked to this form.

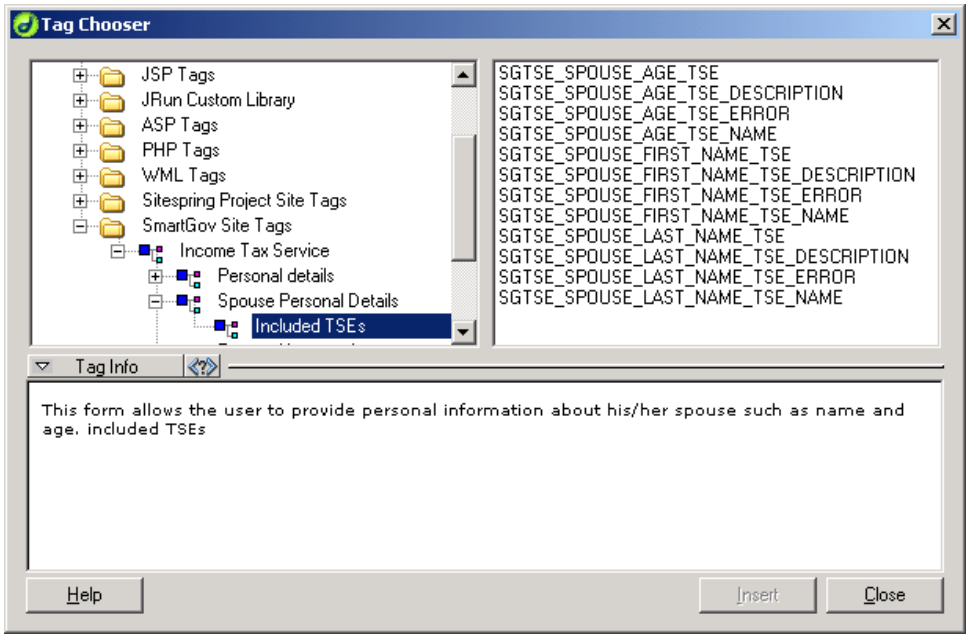


Figure 62 – Enriched "Insert tag" DreamWeaver MX dialog

In the following paragraphs the procedure for inserting tags for each SmartGov entity type is presented.

3.7.3.1 Inserting form-level tags

For any form with id equal to *formId* the following tags are available and may be used:

Tag name	*Description*	*Mandatory?*
SGFORM_formId_BEGIN	Marks the beginning of the form	YES
SGFORM_formId_END	Marks the end of the form	YES
SGFORM_formId_NAME	The name of the form is inserted here	NO
SGFORM_formId_DESCRIPTION	The description of the form is inserted here	NO
SGFORM_formId_ERROR	Error messages produced by the form validation checks	NO

3.7.3.1.1 Inserting the SGFORM_formId_BEGIN tag

In order to insert the SGFORM_formId_BEGIN tag the form designer should click at a form location *before any SmartGov item on the form*, including the form title. The start of the XHTML form document will usually be an appropriate location.

After clicking on the desired location, select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGFORM_formId_BEGIN tag; finally the designer should click on the "Insert" button. The tag choosing procedure is illustrated in Figure 63.

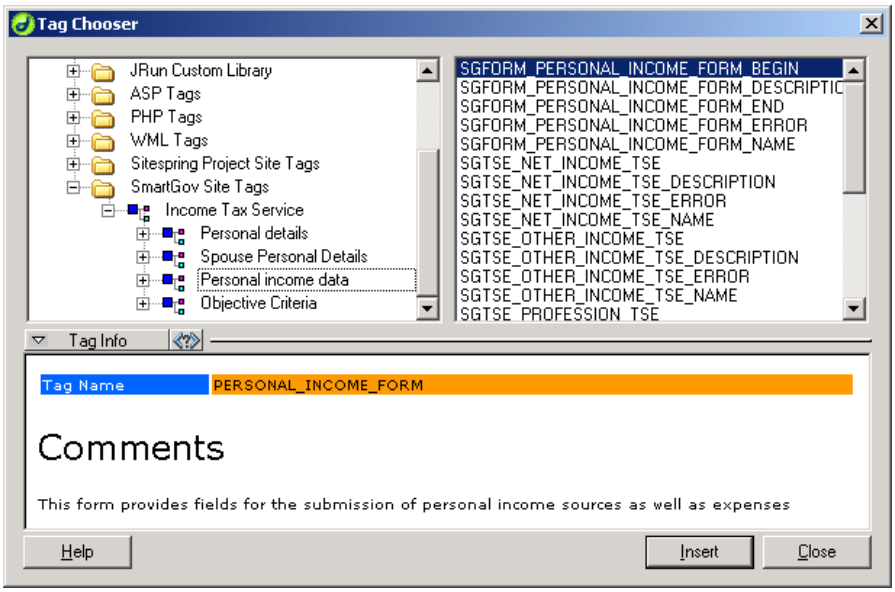


Figure 63 – Inserting the "form begin" tag

No differences in the form appearance should be visible, the tag, however will have been inserted. If the designer wants to verify the tag insertion, she must switch to code view by selecting "View/code" from the DreamWeaver MX menu.

3.7.3.1.2Inserting the SGFORM_formId_END tag

In order to insert the SGFORM_formId_BEGIN tag the form designer should click at a form location *after any SmartGov item on the form*, including the form navigation elements. The end of the XHTML form document will usually be an appropriate location. After clicking on the desired location, select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGFORM_formId_END tag; finally the designer should click on the "Insert" button.

No differences in the form appearance should be visible, the tag, however will have been inserted. If the designer wants to verify the tag insertion, she must switch to code view by selecting "View/code" from the DreamWeaver MX menu.

3.7.3.1.3Inserting the SGFORM_formId_NAME tag

This tag should replace the short description of the form, if such a description appears on it. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the short form description and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGFORM_formId_NAME tag; finally the designer should click on the "Insert" button.

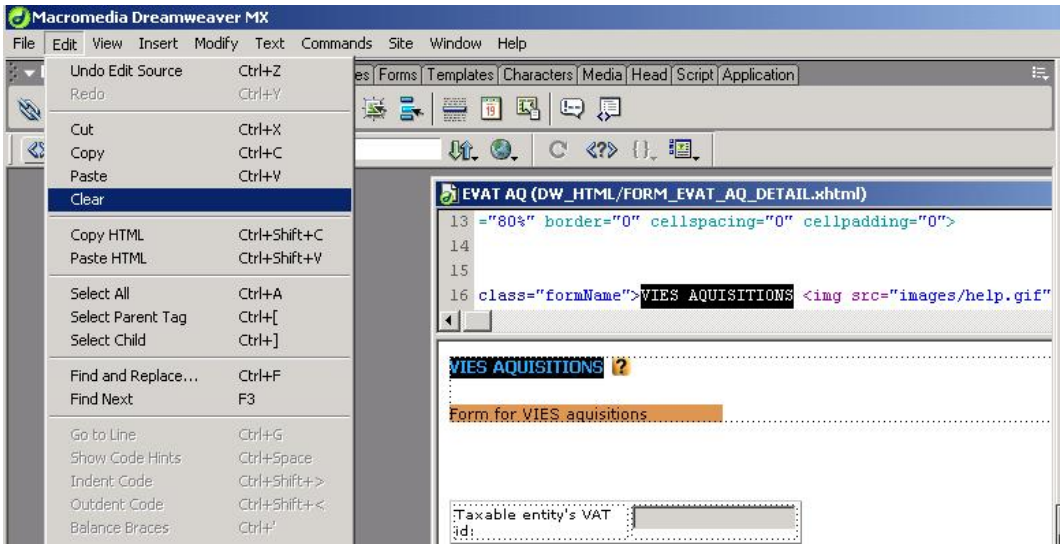


Figure 64 – Deleting the short form title

After this step, *a comment mark indicator* will appear at the place that the form short description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.1.4Inserting the SGFORM_formId_ DESCRIPTION tag

This tag should replace the long description of the form, if such a description appears on it. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the long form description and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGFORM_formId_DESCRIPTION tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the form long description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.1.5Inserting the SGFORM_formId_ERROR tag

This tag should be placed at the location where errors resulting from form validation checks will be displayed. If no validation errors are associated with the form or form elements, this step may be skipped altogether. Firstly, the designer should select any text indicating this space and delete it, as illustrated in Figure 65. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGFORM_formId_ERROR tag; finally the designer should click on the "Insert" button.

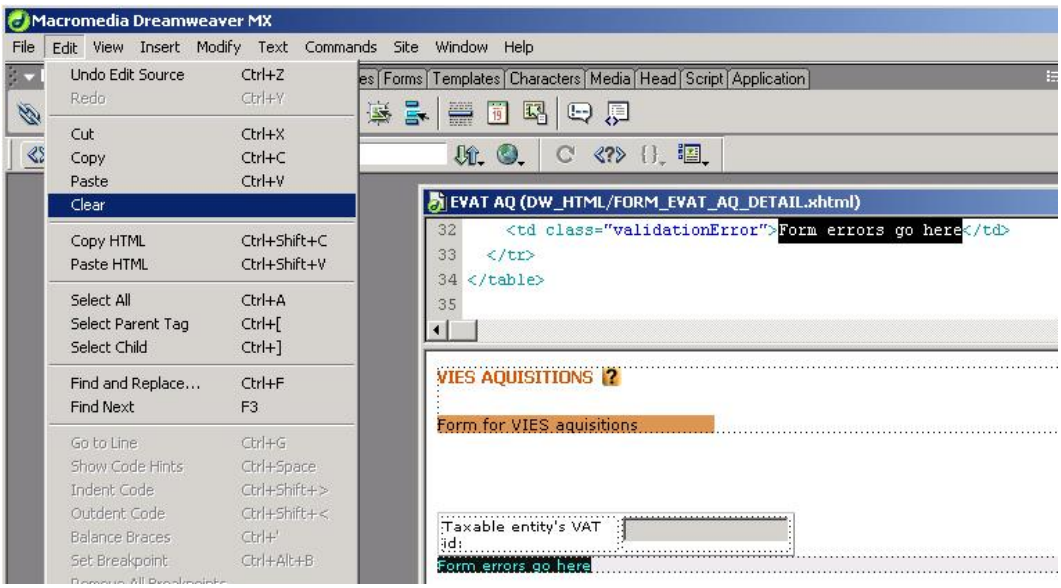


Figure 65 – Deleting the form validation error placeholder text

After this step, a comment mark indicator will appear at the place that the form validation error placeholder text formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module. Please note that during service runtime this space may expand or shrink, depending on the number of validation checks that have failed and the error messages emitted by each validation error.

3.7.3.2 Inserting TSE group-level tags

For any TSE group with id equal to *groupId* the following tags are available and may be used:

Tag name	*Description*	*Manda-tory?*
SGGROUP_groupId_BEGIN	Marks the beginning of the group	YES
SGGROUP_groupId_END	Marks the end of the group	YES
SGGROUP_groupId_NAME	The name of the TSE group is displayed here	NO
SGGROUP_groupId_DESCRIPTION	The description of the TSE group is displayed here	NO
SGGROUP_groupId_ERROR	Error messages produced by the TSE group validation checks are displayed here	NO
SGGROUP_groupId_ADD_BUTTON	The TSE group Add button is displayed here	YES
SGGROUP_groupId_REMOVE_BUTTON	The TSE group remove button is displayed here	YES
SGGROUP_groupId_ADD_BUTTON_ERRORS	Errors from adding rows are inserted here	YES
SGGROUP_groupId_REMOVE_BUTTON_ERRORS	Errors from removing rows are displayed here	YES

3.7.3.2.1 Inserting the SGGROUP_groupId_BEGIN tag

The SGGROUP_groupId_BEGIN tag should be placed *exactly at the beginning* of the visual elements that comprise the group's elements. If a table is used for the group's elements (see SmartGov good practice tip 2 in section 3.7.1), then the first row hosting group elements may be selected by moving the mouse pointer to the left of the row (the pointer becomes an horizontal right arrow) and clicking the left mouse button, as illustrated in Figure 66. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_gropuId_BEGIN tag; finally click on the "Insert" button.

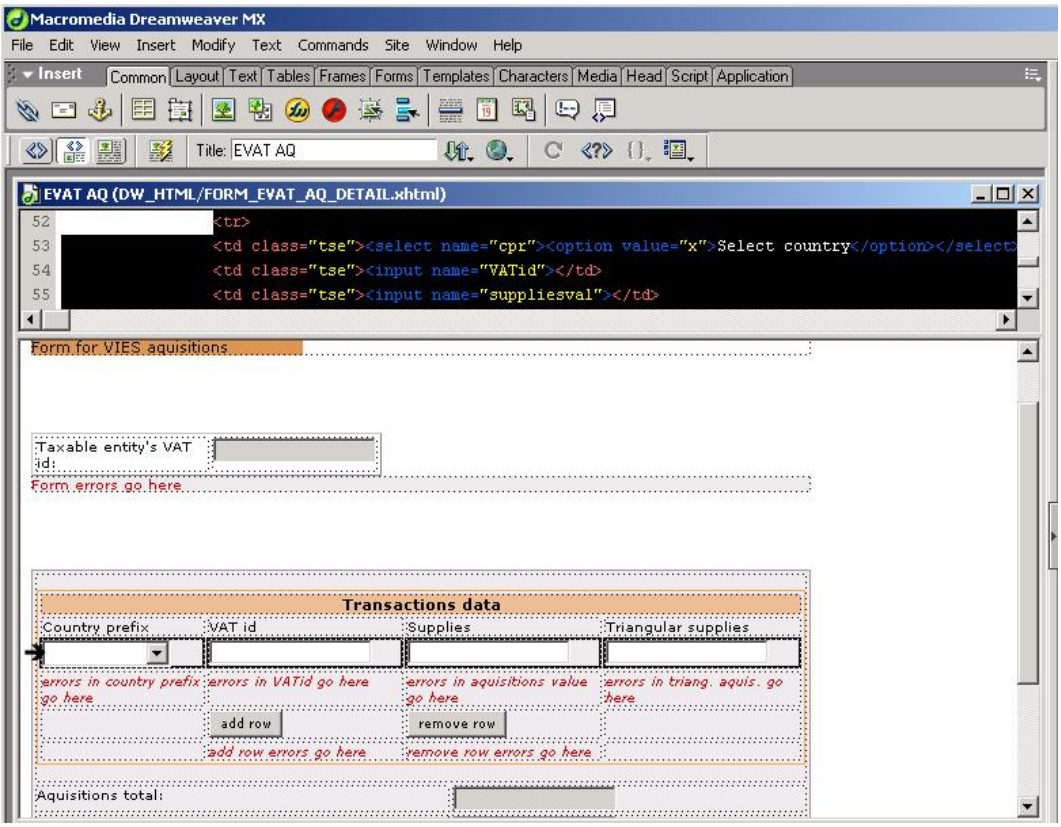


Figure 66 – Selecting the first row hosting TSE group elements

No differences in the form appearance should be visible, the tag, however will have been inserted. If the designer wants to verify the tag insertion, she must switch to code view by selecting "View/code" from the DreamWeaver MX menu.

3.7.3.2.2Inserting the SGGROUP_groupId_END tag

The SGGROUP_groupId_END tag should be placed *exactly at the end* of the visual elements that comprise the group's elements. If a table is used for the group's elements (see SmartGov good practice tip 2 in section 3.7.1), then the row *immediately after the last row hosting group elements* (including the error message placeholders for the group elements) may be selected by moving the mouse pointer to the left of the row (the pointer becomes an horizontal right arrow) and clicking the left mouse button, as illustrated in Figure 67. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_gropuId_END tag; finally click on the "Insert" button.

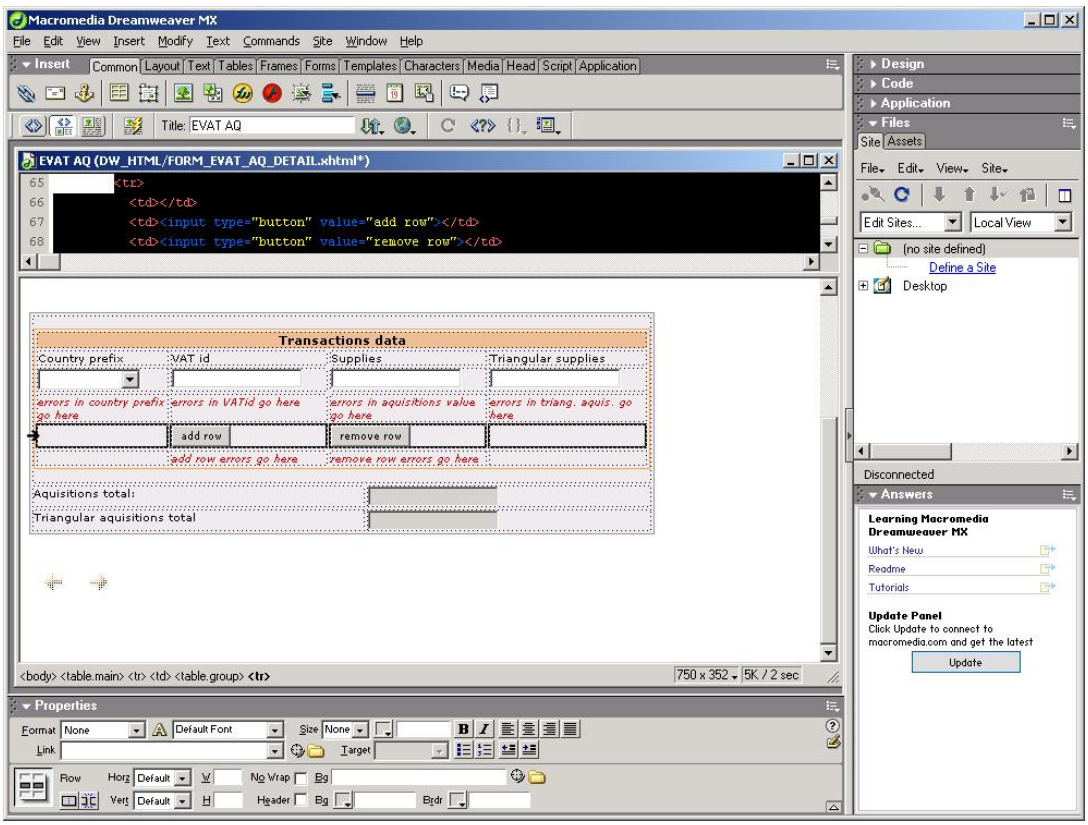


Figure 67 – Selecting the proper row when inserting the SGGROUP_groupId_END tag

No differences in the form appearance should be visible, the tag, however will have been inserted. If the designer wants to verify the tag insertion, she must switch to code view by selecting "View/code" from the DreamWeaver MX menu.

3.7.3.2.3Inserting the SGGROUP_groupId_NAME tag

This tag should replace the short description of the group, if such a description appears on it. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the short group description and delete it, as illustrated in Figure 68. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_NAME tag; finally the designer should click on the "Insert" button.

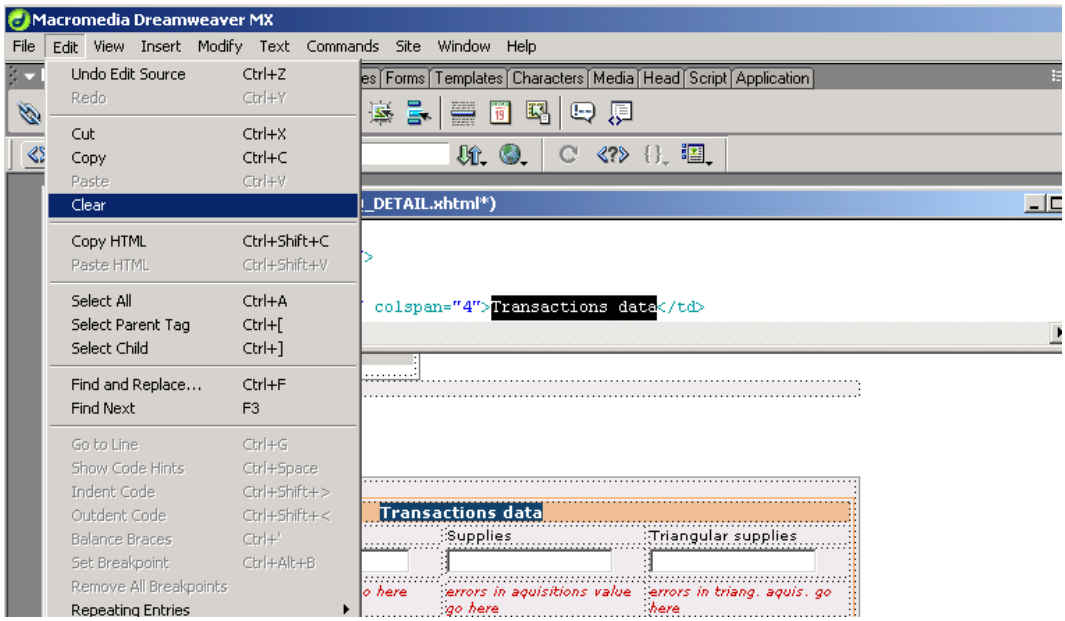


Figure 68 – Deleting the short group description

After this step, a comment mark indicator will appear at the place that the group short description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.2.4Inserting the SGGROUP_formId_ DESCRIPTION tag

This tag should replace the long description of the group, if such a description appears on it. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the long group description and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_DESCRIPTION tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the group long description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.2.5Inserting the SGGROUP_groupId_ERROR tag

This tag should be placed at the location where errors resulting from group validation checks will be displayed. If no validation errors are associated with the group or group elements, this step may be skipped altogether. Firstly, the designer should select any text indicating this space and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate

form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_ERROR tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the group validation error placeholder text formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module. Please note that during service runtime this space may expand or shrink, depending on the number of validation checks that have failed and the error messages emitted by each validation error.

3.7.3.2.6 Inserting the SGGROUP_groupId_ADD_BUTTON tag

This tag should be placed at the location where the "add row" control for the group should appear. Firstly, the designer should select any text or widget indicating this space and delete it, as depicted in Figure 69 (the button labelled "add row" is selected). Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_ADD_BUTTON tag; finally the designer should click on the "Insert" button.

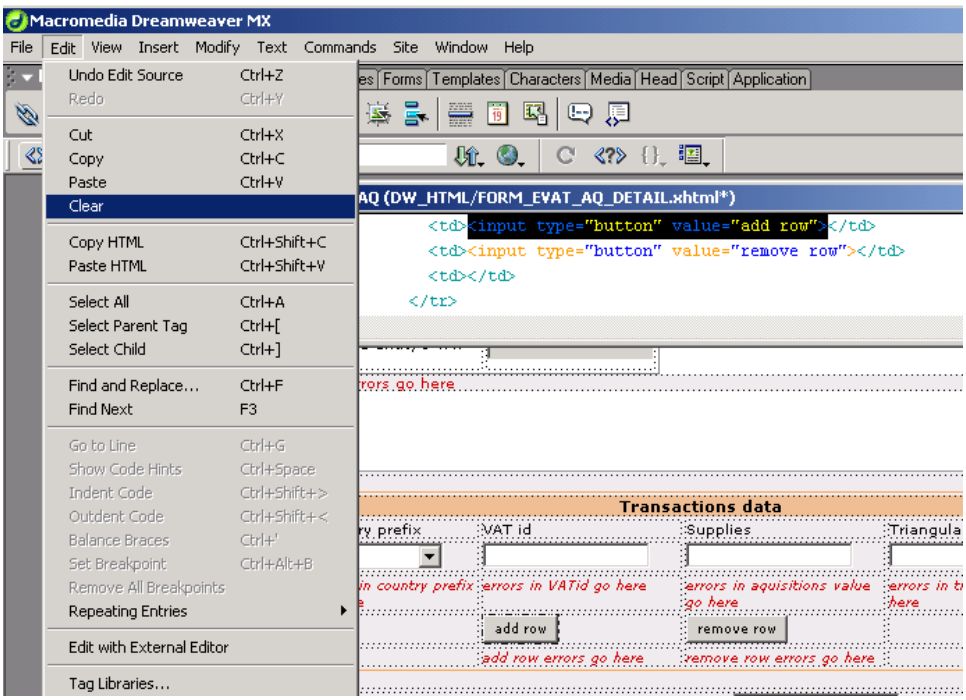


Figure 69 – Deleting the "add row" control placeholder

After this step, a comment mark indicator will appear at the place that the group "add row" control placeholder formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.2.7Inserting the SGGROUP_groupId_REMOVE_BUTTON tag

This tag should be placed at the location where the "remove row" control for the group should appear. Firstly, the designer should select any text indicating this space and delete it, as depicted in Figure 69. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_REMOVE_BUTTON tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the group "remove row" control placeholder formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.2.8Inserting the SGGROUP_groupId_ADD_BUTTON_ERRORS tag

This tag should be placed at the location where errors emitted during "add row" operations for the group should appear (e.g. adding rows to a group having reached its row limit). Firstly, the designer should select any text indicating this space and delete it, as depicted in Figure 70. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_ADD_ERRORS_BUTTON tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the group "add row" error messages placeholder text formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

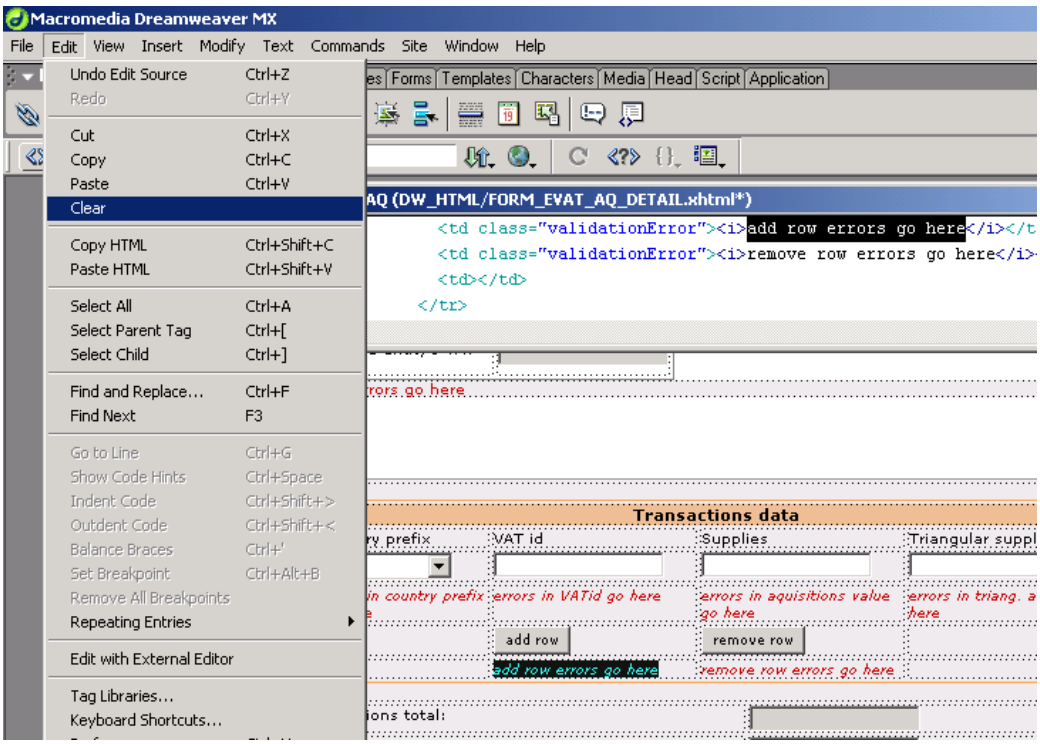


Figure 70 – Deleting the "add row" error messages placeholder

3.7.3.2.9 Inserting the SGGROUP_groupId_REMOVE_BUTTON_ERRORS tag

This tag should be placed at the location where errors emitted during "remove row" operations for the group should appear (e.g. removing rows from a group with no rows in it). Firstly, the designer should select any text indicating this space and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGROUP_groupId_REMOVE_ERRORS_BUTTON tag; finally the designer should click on the "Insert" button.

After this step, a comment mark indicator will appear at the place that the group "remove row" messages placeholder text formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.3 Inserting TSE-level tags

For any TSE with id equal to *tseId* the following tags are available and may be used:

Tag name	*Description*	*Manda-tory?*
SGTSE_tseId	The actual TSE area is displayed here	YES

Tag name	*Description*	*Manda-tory?*
SGTSE_tseId_NAME	The name of the TSE is displayed here	NO
SGTSE_tseId_DESCRIPTION	The description of the TSE is displayed here	NO
SGTSE_tseId_ERROR	Error messages produced by the TSE validation checks are displayed here	NO

3.7.3.3.1 Inserting the SGTSE_tseId tag

This tag should be placed at the location where the actual TSE area will be displayed on the form. The actual widget used for the TSE is selected by the Integrator module, depending on the TSE semantic information. For example, TSEs with Boolean types will be represented via check boxes; TSEs for which the user should select a value among a set of pre-defined ones will be represented as a drop-down list and so on.

Firstly, the designer should select any text or control indicating this space and delete it, as shown in Figure 71. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGGTSE_tseId tag; finally the designer should click on the "Insert" button

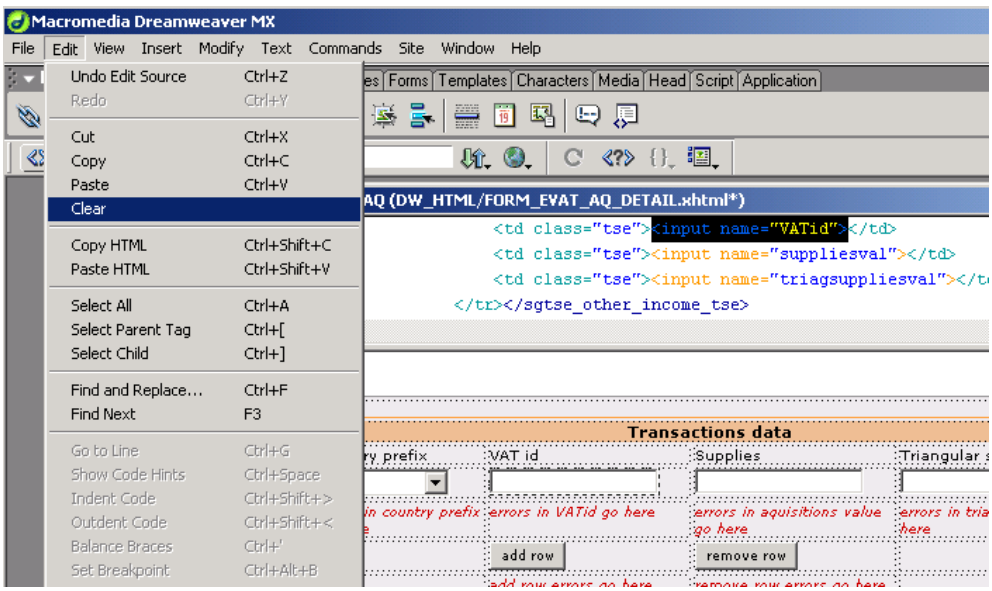


Figure 71 – Deleting the TSE placeholder

For the designer's convenience, a separate sub-folder is provided under each form folder, labelled "Included TSEs", as illustrated in Figure 72. This subfolder

contains only the TSEs appearing on the form, facilitating the selection of the appropriate TSE-related tags.

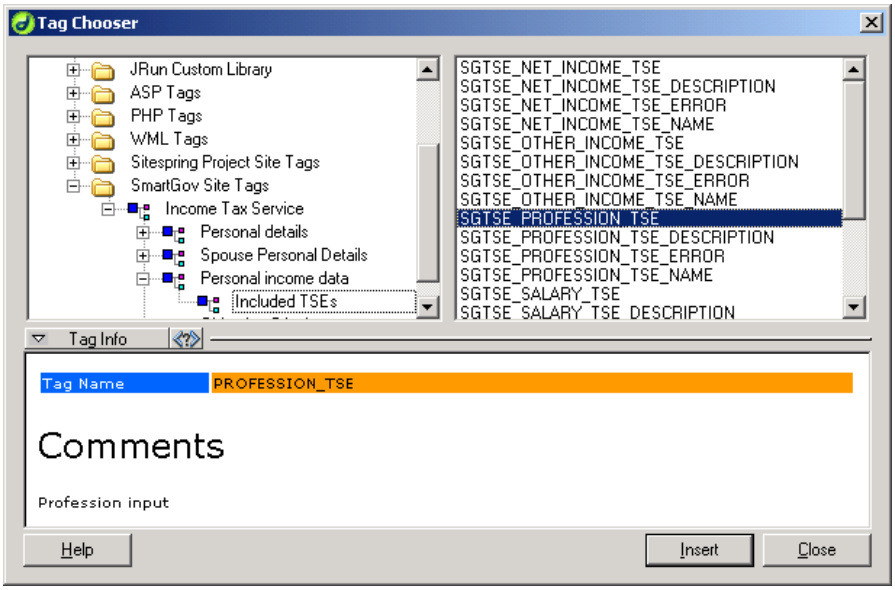


Figure 72 – Included TSEs subfolder

After this step, a comment mark indicator will appear at the place that the TSE control placeholder text formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.3.2Inserting the SGTSE_tseId_NAME tag

This tag should replace the short description of the TSE, if such a description appears on it. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the short TSE description and delete it, as illustrated in Figure 73. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGTSE_tseId_NAME tag; finally the designer should click on the "Insert" button. The designer may also use the convenience "Included TSEs" subfolder.

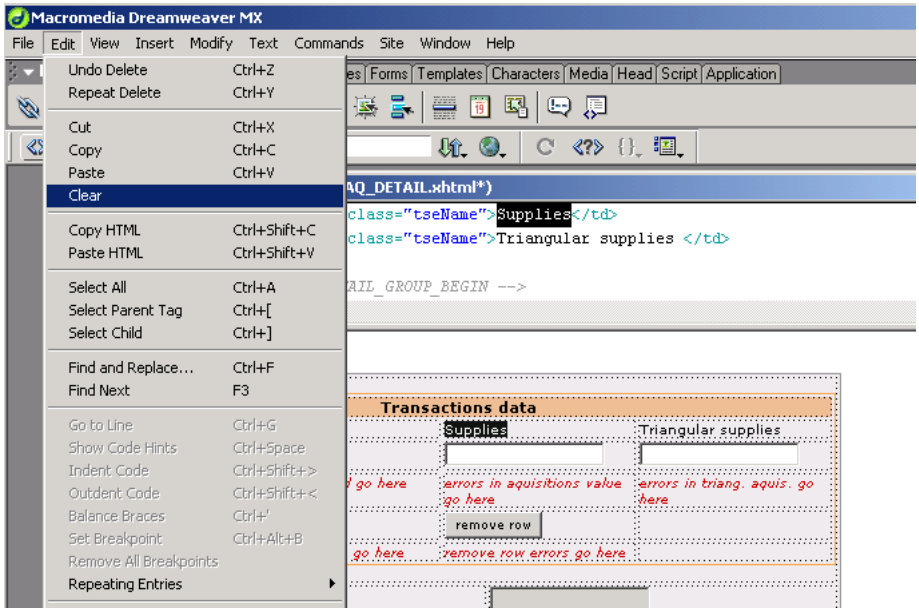


Figure 73 – Deleting the short TSE description

After this step, a comment mark indicator will appear at the place that the TSE short description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.3.3 Inserting the SGTSE_tseId_ DESCRIPTION tag

This tag should replace the long description of the TSE, if such a description appears on the form. If no such description appears, the tag insertion step may be skipped altogether. Firstly, the designer should select the long TSE description and delete it. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGTSE_tseId_DESCRIPTION tag; finally the designer should click on the "Insert" button. The designer may also use the convenience "Included TSEs" subfolder.
After this step, a comment mark indicator will appear at the place that the TSE long description formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.3.4Inserting the SGTSE_tseId_ERROR tag

This tag should be placed at the location where errors emitted from the validations associated with the TSE should be displayed. Note that these validations include "implicit" checks, such as data-type validations (e.g. a numeric TSE is always checked to determine if the user actually entered a numeric value). If implicit or explicit validation checks are associated with the TSE, the tag

insertion step may be skipped altogether. Firstly, the designer should select the long TSE validation checks error text placeholder and delete it as shown in Figure 74. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGTSE_tseId_ERROR tag; finally the designer should click on the "Insert" button. The designer may also use the convenience "Included TSEs" subfolder.

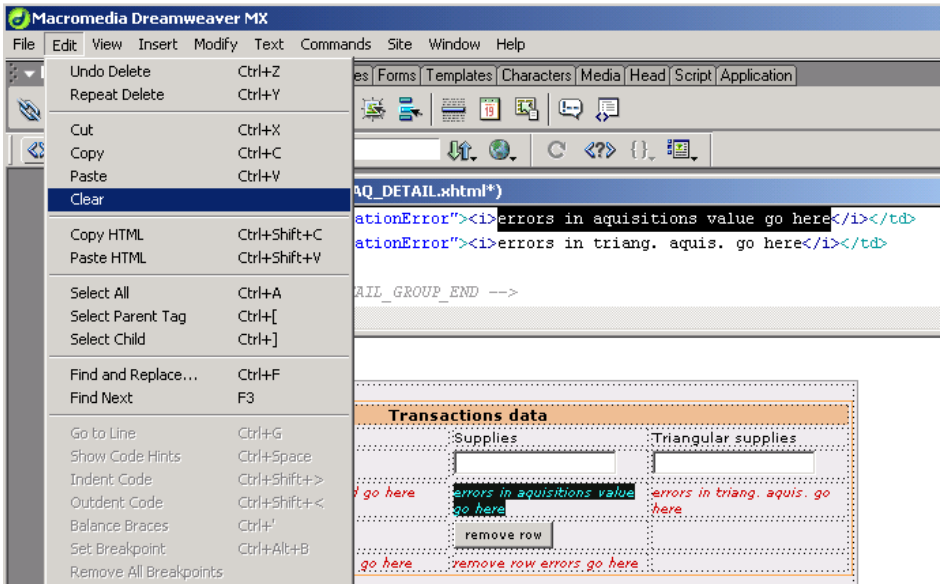


Figure 74 – Removing the TSE error messages placeholder

After this step, a comment mark indicator will appear at the place that the TSE error messages formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.3.4 Inserting KU-level tags

For any KU with id equal to *KUId* the tag SGKU_kuId is available. This tag is optional, and can be placed at the location that help anchors should appear on the form. Firstly, the designer should select the help anchor placeholder and delete it, as depicted in Figure 75. Afterwards, the designer should select "Insert/Tag" from the menu, navigate to the appropriate form sub-folder in the tags hierarchy by first opening the SmartGov site tags folder, then the relevant service and finally selecting the appropriate form node, and from the right pane choose the SGKU_kuId tag; finally the designer should click on the "Insert" button. The designer may also use the convenience "Included KUs" subfolder, which is a direct descendant of the relevant form folder.

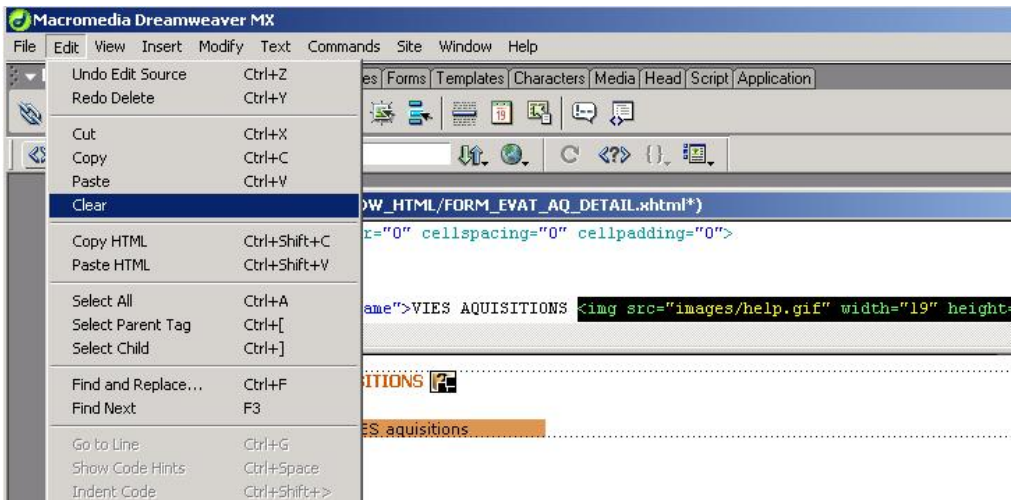


Figure 75 – Deleting the help anchor placeholder

After this step, a comment mark indicator will appear at the place that the help anchor placeholder formerly was; this space is internally occupied by the appropriate tag, and will be filled in by the Integrator module.

3.7.4 Final form appearance

After having completed the link establishment activities, the form will appear with virtually no content on it, as depicted in Figure 76, with only comment mark indicators present. These comment mark placeholders contain all the information necessary for the Integrator module to create the services.

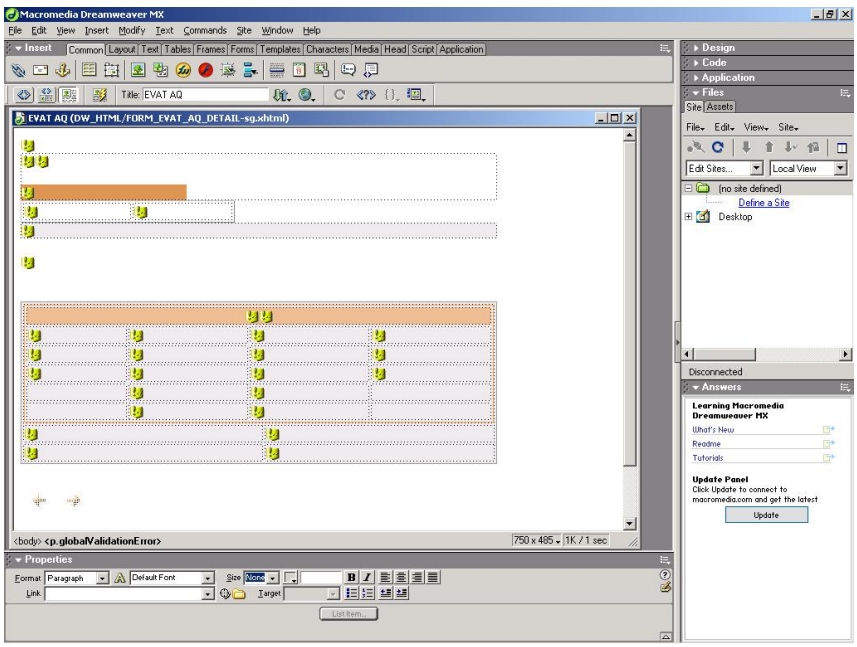


Figure 76 – Form design view after link establishment

The XHTML form is now ready for uploading to the SmartGov platform and for processing by the Integrator module.

4 The SmartGov Integrator tool (ARC)

4.1 Introduction

4.1.1 Summary

The SmartGov Integrator is a functionally complex component. It interacts with various system components directly (SmartGov Agent-SGA, Information Interchange Gateway-IIG, XMLStore) or indirectly (Front-end UI application). For this reason, it is absolutely critical for the component to be set up correctly.

4.1.2 Purpose, Scope and Audience

The purpose of this document is to provide setup instructions for the correct deployment of the Integrator component and serve as a simple usage guide.

The scope of this document is not to explain the internals of the Integrator, how it co-operates with various system components or the technologies used. Instead, this document is meant to serve as a step-by-step installation guide for end-users to avoid possible pitfalls. And get up-and-running a.s.a.p.

This document is targeted towards the members of the SmartGov consortium and whichever third party might be interested in installing the Integrator component.

4.1.3 Typesetting Conventions

Monospace text (e.g., `DataServices`) designates identifiers, such as keys in a properties file. Slanted text (e.g., *./temp/foo*) is used to designate file names and paths. Slanted (e.g., *`yyyy`*) monospace text designates placeholders for user input. Bold text (e.g. **Deployment**) is used for emphasis.

4.2 Requirements

For the Integrator to operate smoothly, two distinct, application server installations are required, in two different hosts: one for service **development** and one for the final **deployment** and operation of the generated service.

However, in cases when this is not feasible, both application server installations may co-exist in the same host (provided that they operate on different ports).

The following tables describe the minimum hardware/software requirements for each of the two hosts:

Hardware	Development	Deployment
CPU	Pentium IV, 1.8 GHz	Pentium IV, 2.8 GHz
RAM	> 512 Mb	> 756 Mb
HDD	**>** 100Mb free space	**>** 100Mb free space

Software	Development	Deployment
OS	Windows 2000, Service Pack 3+	Windows 2000, Service Pack 3+
Servlet Engine	Tomcat 4.1+	Tomcat 4.1+
JDK	Java2 SE 1.4.2+	Java2 SE 1.4.2+

4.3 Environment Setup

The Integrator co-operates closely with the XMLStore during design-time to retrieve service description files. On the other hand, during run-time, the generated service propagates and retrieves documents to the SGA, which in turn propagates changes to the IIG. These three components require a DB to be setup in a host, accessible by the development and deployment hosts.

4.3.1 Setup actions roadmap

To correctly set up the Integrator and all peripheral components, the following actions need to be taken in the following order. Each action item corresponds to a subsequent section of the document.

- **Install Integrator**

  The Integrator is available as a self-installing package. See 4.3.2 for more details.

- **Create SGA/IIG DBs**

  The SGA and the IIG rely heavily on the existence of databases for the temporary storage of outgoing/incoming messages as well as for storing authentication information. For more details, see 4.3.3.

  - **Populate IIG login DB**

    During authentication, user credentials are checked against a back-end DB containing all registered users. To be able to create a running service, one or more user accounts have to be created. For details, see 4.3.4.

  - **Create IIG XML Repository**

  The IIG needs to access a run-time XML Repository to store incoming user documents. The details on how this is done can be found in 4.3.5.

- **Install IIG**

The IIG comes bundled as an installer. Section 4.3.6 details the screens of the installer in relation to the previous steps.

- **Fine-tune installed IIG**

After the IIG has been installed, there are a number of steps that may be needed to enhance its functionality. Section 4.3.7 provides more info.

- **Set up document pre-population**

This step is optional and concerns certain applications where initial values should be displayed for certain fields, originating from a back-end system. Section 4.3.8 details the steps needed to be taken for the creation of a pre-population "DB" for the eVies service, where each user should have his/her own personal details.

- **Create Integrator XML Repository**

The Front-End and the Integrator share a common, design-time service element repository. Section 4.3.9 details the steps required for the creation of the repository so as it is accessible by the Integrator.

- **Populate Integrator XML Repository**

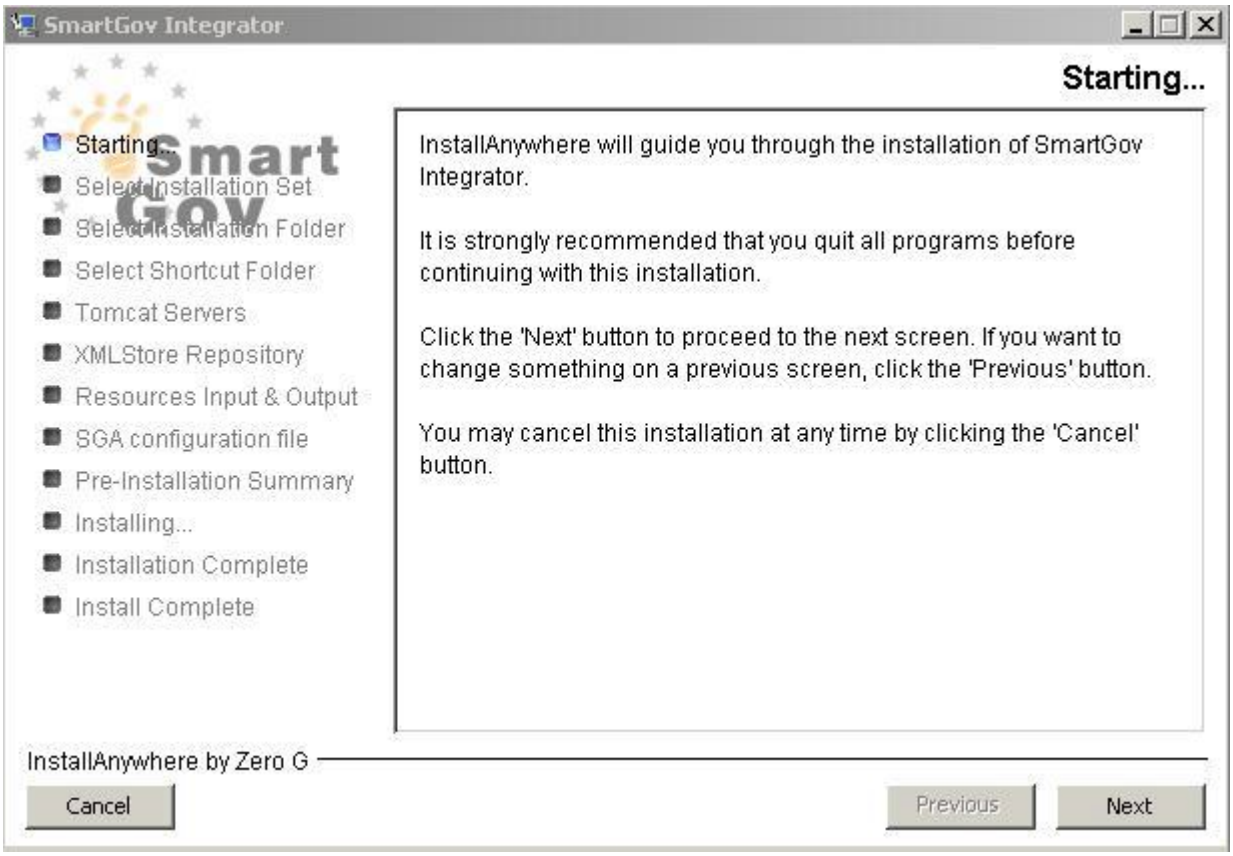
For testing purposes (e.g. when the Front-End is not installed) the design-time repository may need to be populated with service elements. For details, see 4.3.10.

- **Configure the deployment server**

The generated service uses a contained SGA agent to communicate with the IIG. For the SGA to function properly, a set of configuration files has to be updated. Moreover, the Integrator requires a special account on the deployment server. The details on this process are given in section 4.3.11.

4.3.2 Install Integrator

4.3.2.1 Splash screen



Displayed at the beginning of the installation. At this point you should stop all Tomcat processes that may be running on both deployment and development hosts.

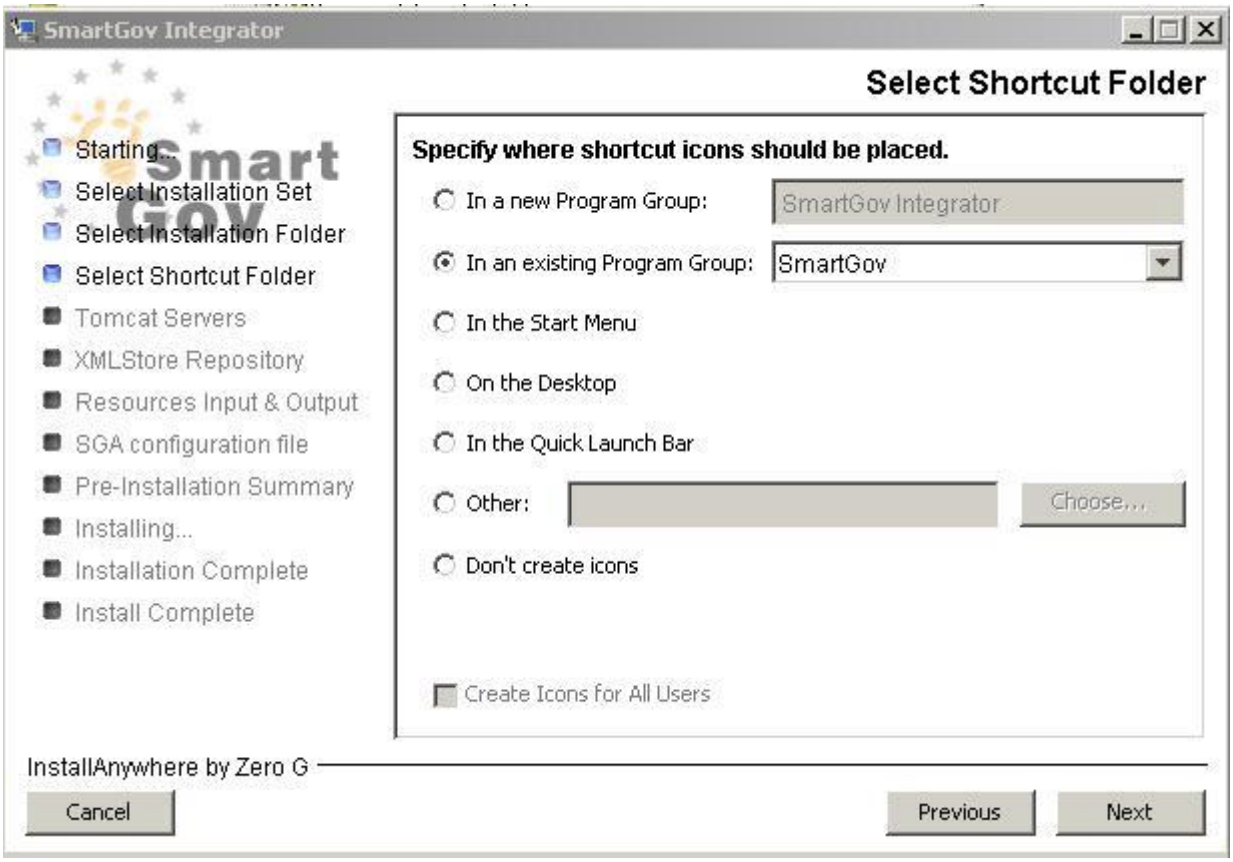
4.3.2.2 Installation type

This screen allows the selection of the installation type. Normal installation selects all features, while Minimal does not install the Document Crawler and XMLStore Manager.

4.3.2.3 Installation directory

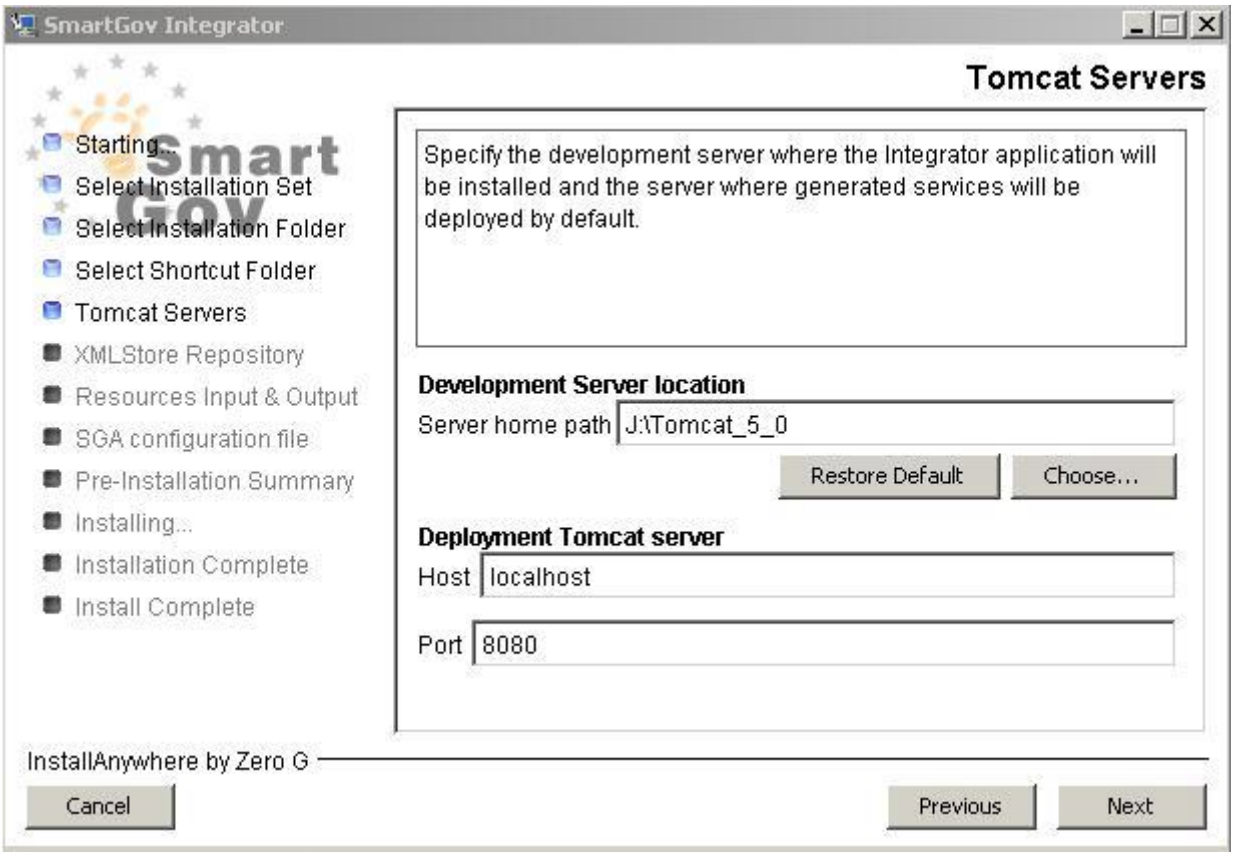
This directory only contains uninstall information about the application. It is of absolutely no importance as the installation process, not the Integrator itself, only uses it.

4.3.2.4 Shortcut group



This screen determines where the application's shortcuts should be placed.

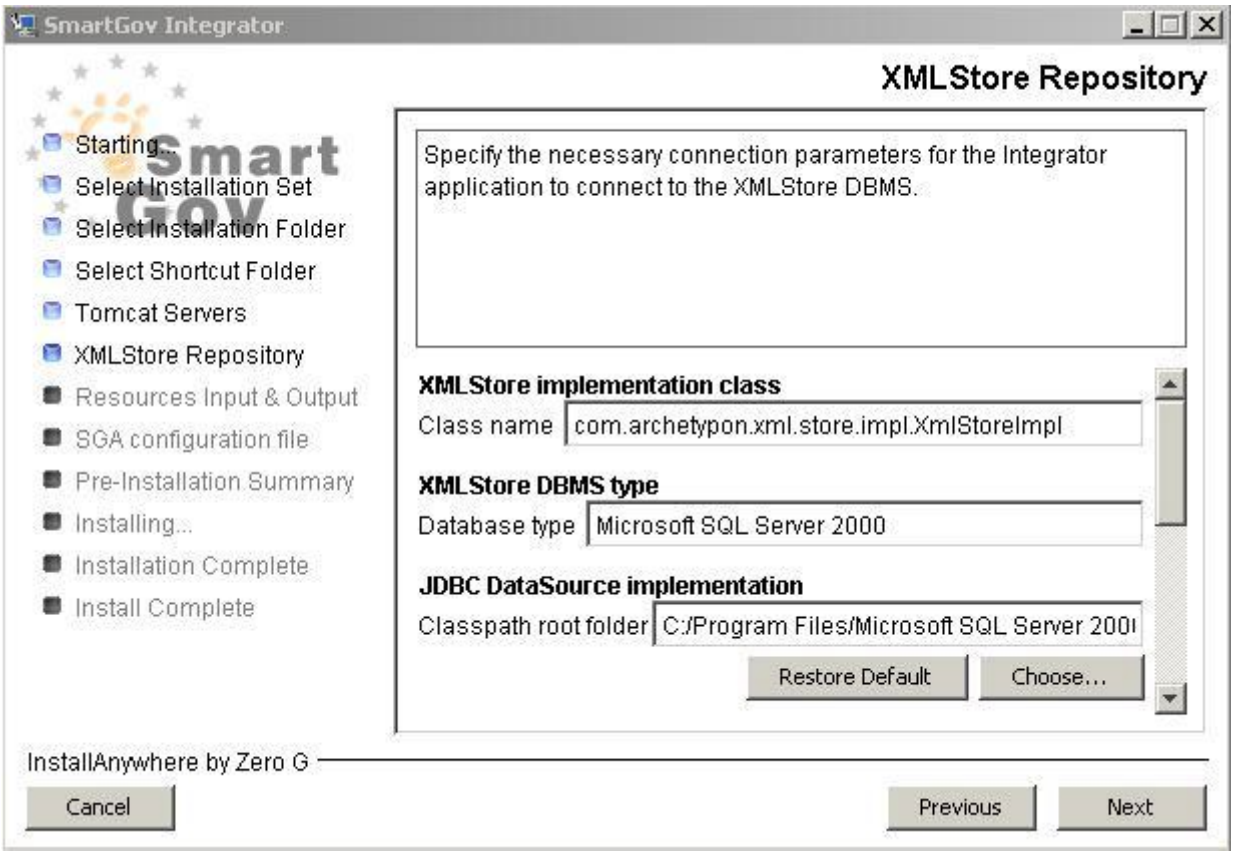
4.3.2.5 Tomcat server configuration



This screen requires the directory where the development Tomcat server is installed. It also requires the host name and port where the deployment Tomcat listens on. It is very important to specify all information correctly, as denoted in the image, for the application to function properly.

It is also a very good idea to have the development server installed in a directory path that **does not contain spaces** (e.g. avoid *c:\Program Files\Tomcat*, *d:\My Server\Tomcat 4*, etc). Tomcat is known to occasionally present erratic behavior when installed in such a directory.

4.3.2.6 XML Repository configuration



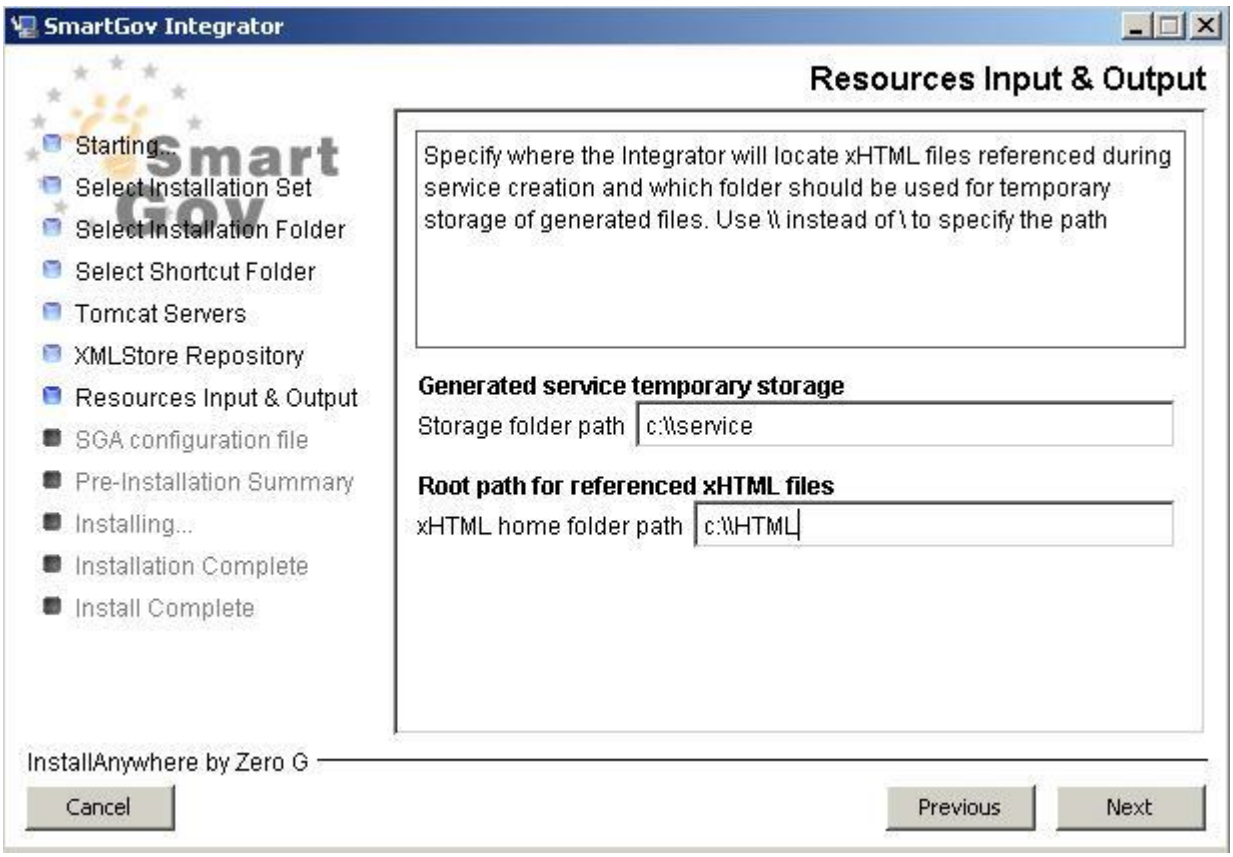
Specifies the connectivity information for the Integrator to locate the service element repository. The information defined in this screen should be identical with that defined later on, in section 4.3.8.

The different fields of the screen are explained below

- *Class name*: Should be left **as is**, unless otherwise noted in the documentation.
- *Database type*: Type of DBMS that the XML Repository is installed on. Only two values are accepted at this time: `Microsoft SQL Server 2000` and `MySQL 4.x`. Putting any other value in this field will require re-installation or manual correction.
- *Classpath root folder*: Browse to select a folder where the classes of the JDBC driver have been extracted. One can only select a folder at this stage and not a JAR. This has been done so because of the Microsoft JDBC driver being delivered in multiple JARs. So, if your driver is delivered as a ZIP/JAR, you will need to extract it in a folder and select this folder in this field. Be careful to **modify** the final path that you will select in this field. Replace all back-slashes (\) with **forward-slashes** (/).

- *Datasource class*: The JDBC Datasource implementation of the driver (e.g. `com.microsoft.jdbcx.sqlserver.SQLServerDataSource` for the official Microsoft SQL Server driver, `com.mysql.jdbc.jdbc2.optional.MysqlDataSource` for the official MySQL driver). Consult your driver's documentation.
- *Server name*: The qualified intranet name of the host that the XML Repository is (or will be) installed.
- *DB name*: The name of the DB that contains (or will contain) the XML Repository.
- *Username*: SQL login for the target host
- *Password*: For the previous login

4.3.2.7 Input / output directories



The first field defines which folder should be used to temporarily store the generated service files. This feature is useful so as to have a backup copy. File separators should be added as shown. The directory will be created at run-time if not present.

The second field defines the directory where the xHTML files used by the service are located. Care should be taken to take into account the relative url defined in the service description files. (E.g. if the xHTMLs are located in *c:\Foo\Html* and

service description files refer to files *Html/XXX.xhtml*, and then you should specify *c:\\Foo* in this dialog). If you are using the Integrator along with the Front-End, this folder should be the **same** as the one where the Front-End saves uploaded xHTML files. *All files placed inside this directory should have an .xhtml file extension*.

4.3.2.8 SGA configuration file

This screen requires the location of the SGA configuration file in the deployment server. The deployed service will use this value to locate it and attempt to initialize the agent. For information on how to setup the SGA configuration files, see 4.3.11. The recommended value for this field is *c:/SmartGov/conf/sga/SGAConfig.txt*.

4.3.2.9 Summary

Contains the summary of the installation parameters.

4.3.3 SGA/IIG DBs

The SGA and the IIG have two internal Pending Actions Queues (PAQs) where they store messages, which for some reason failed to be sent. These PAQs are implemented over a relational DB and are one for incoming (EntraPAQ) and one for outgoing messages (AdelantePAQ), making a total of four different DBs . Moreover, the current version of the IIG implicitly requires a DB to maintain service user logins.

The schema of the first four DBs is exactly the same, so, in a low-traffic environment they can be "merged" into one physical DB. In that case, all different modules will establish connections to the same DB. In addition, since the table names of the login DB are different than those used in the PAQ DBs, this DB may also be merged with the previous. So, to make things short, this step shall create at least one or at most 5 different DBs (as combinations in DB merging may be chosen – e.g. both SGA PAQs in one host, etc)

To create the PAQ DBs, follow these steps:

1. Connect to the DB host that you wish, through the administration client (e.g. Enterprise Manager for MS SQL Server).

2. Create a new DB. Note down its name and its logical function (e.g. DB named `Test1` shall be the SGA PAQ) as this will be used later on.
3. Connect to the new DB and execute against it the proper script found in Appendix D.
4. If you choose to create more than one DB, go to step 2, being careful to select different names.

To create the login DB, follow these steps:

1. Connect to the DB host that you wish, through the administration client (e.g. Enterprise Manager for MS SQL Server).
2. Create a new DB.
3. Connect to the new DB and execute against it the proper script found in Appendix E.

If you choose to create one common DB for all the PAQs and the login information then concatenate the two relevant scripts before executing against the target DB.

4.3.4 Populate IIG login DB

After the IIG login DB has been created, a number of end-user accounts needs to be added to be able to use the generate service later on. Each user account is allowed to access one or more services, identified by their unique id, as defined during the design stage.

Suppose we have two services (`TaxService` and `eVies`) and want to update the DB so as to "capture" the usage scenario depicted in the following table.

Alias	Full Name	Password	Allowed to use…
`Foo`	`John Foo`	`foo`	`TaxService`
`Doe`	`Jack Doe`	`doe`	`TaxService` `eVies`

In that case, the login DB's `SGUserData` table should look like the following image

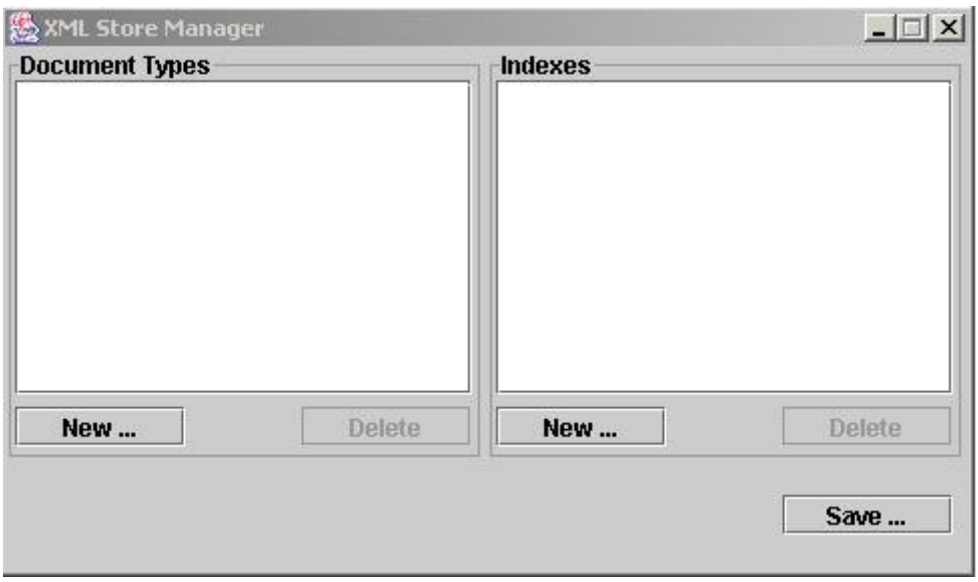
At the same time, the `SGUserServices` table should look like this

4.3.5 Create IIG XML Repository

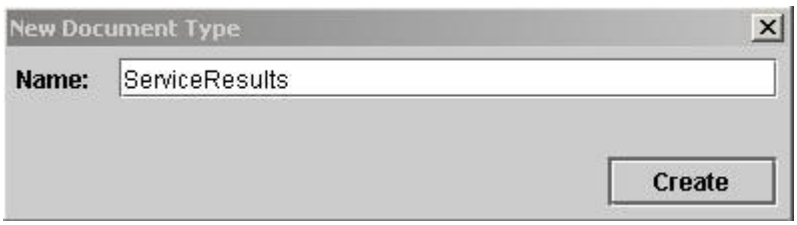
This step supposes that the Integrator has been installed in 4.3.2 selecting Normal setup. In a different case, the features mentioned here are not available.

To create the IIG XML Repository perform the following steps:

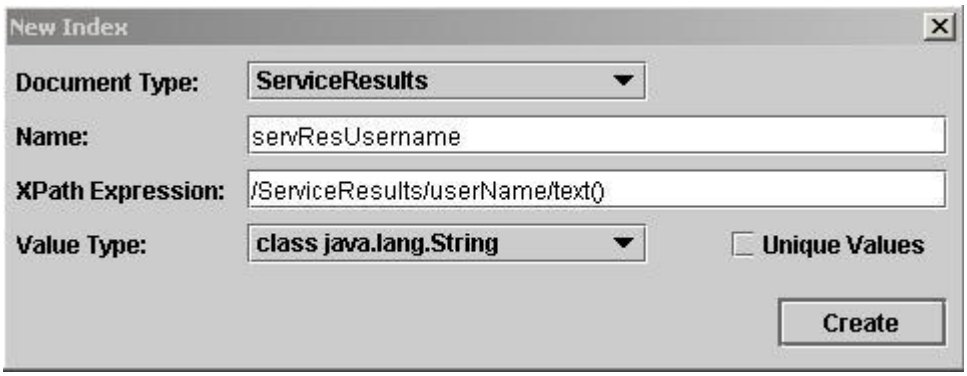
1. Create a DB through the target DBMS' user interface. Only MS SQL Server 2000 and MySql v.4.x are supported. We assume that the target DB is named `IIGXmlStore`, located in an SQL Server host, named `testHost`.
2. From the Windows Start menu, select *`Programs -> SmartGov -> XMLStore Manager`*. The XML Store Manager application is launched.



3. Press the left **New...** button to create a new document type for service description files. Name it `ServiceResults`.



4. Press the right **New...** button to create a new index for the created document type.



For the IIG to work correctly, 3 different indexes need to be created in total in the same manner.

Their details follow (all entries are case-sensitive):

- Name: **servResUsername**
    i. XPath: `/ServiceResults/userName/text()`
    ii. Value type: `string`
    iii. Non-unique
- Name: **servResServiceName**

i. XPath: `/ServiceResults/serviceName/text()`

ii. Value type: `string`

iii. Non-unique

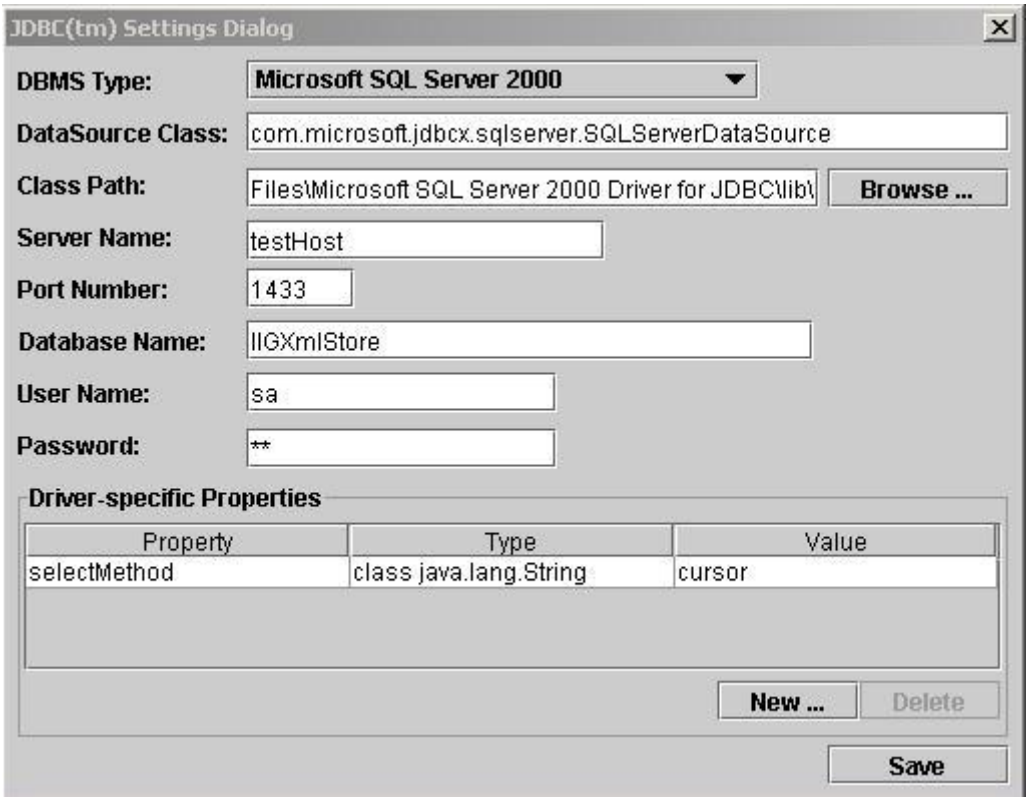
- Name: **servResTimestamp**

i. XPath: `/ServiceResults/timestamp/text()`

ii. Value type: `string`

iii. Non-unique

5. After all indexes are created and listed in the main window (make sure the Document Type list entry is selected), the **Save...** button is pressed.



The information presented in the image matches the suppositions mentioned earlier. The driver property has been added by using the **New...** button. The class path can either be a directory or a JAR file, inside which all driver classes reside.

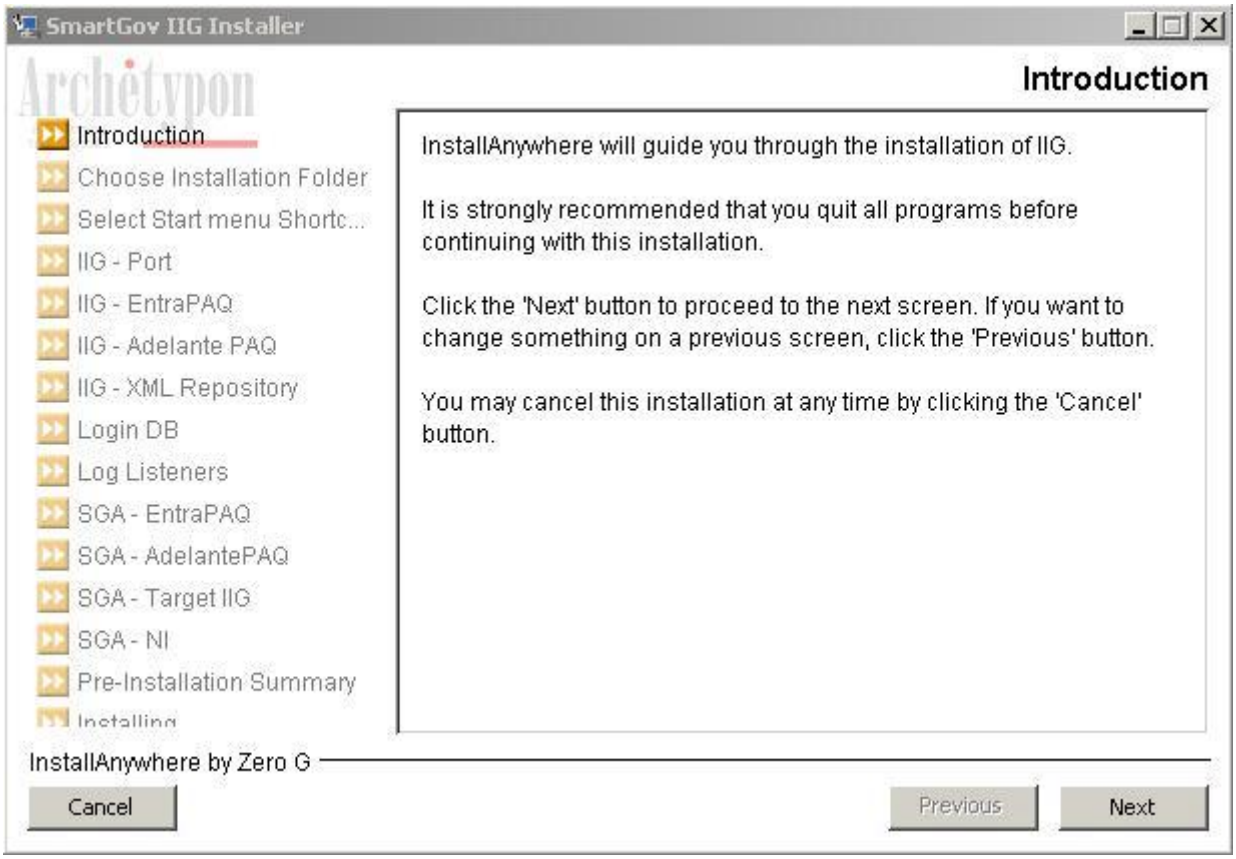
If the target DB host were a MySQL server, then it would be

- DBMS type: `MySQL 4.x`
- DataSource class: `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`
- Port number: `3306`
- Username: `root`
- Password: `<none>`

6. Pressing the **Save...** button creates the following tables: *XpathIndex, servResServiceName, servResUsername, servResTimestamp*. Open the target DB with a DB client application to verify the creation of the new tables.

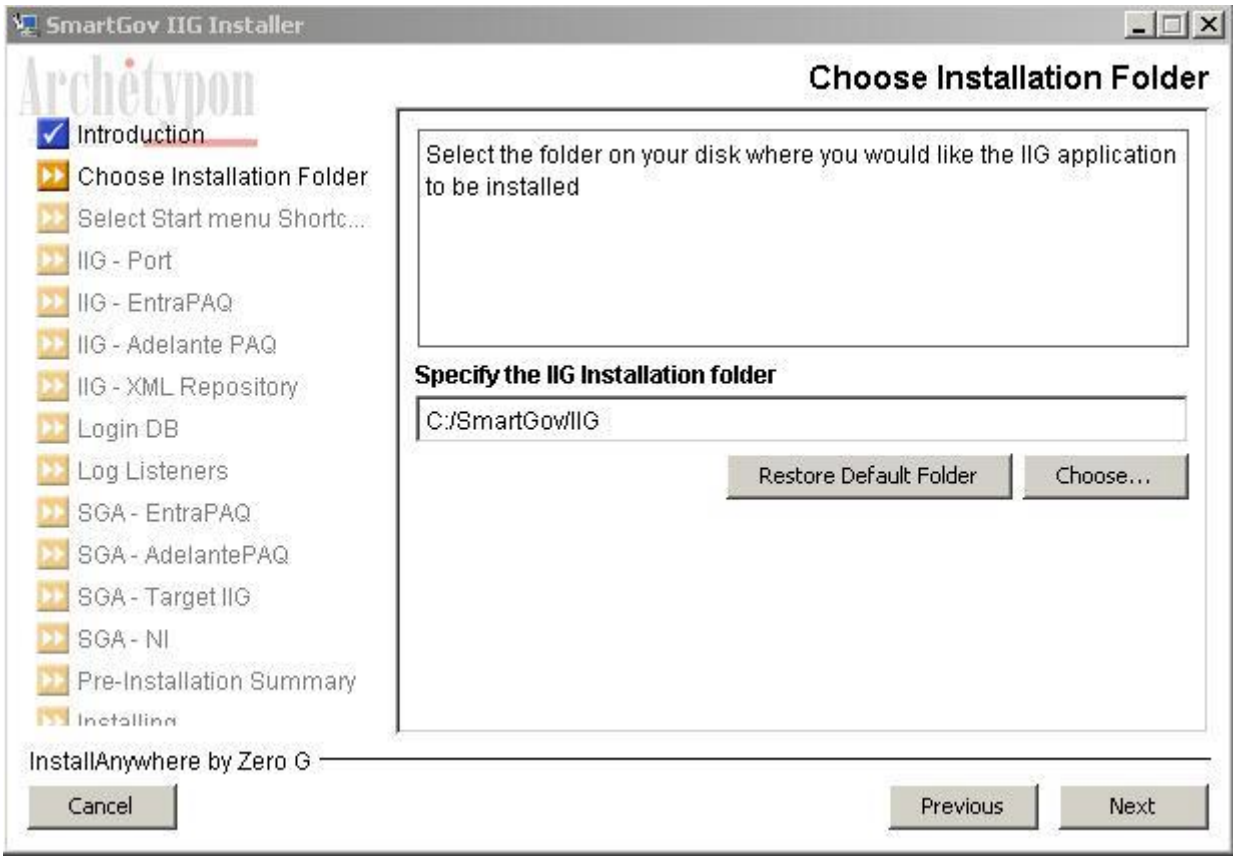
4.3.6 Install IIG

4.3.6.1 Splash screen



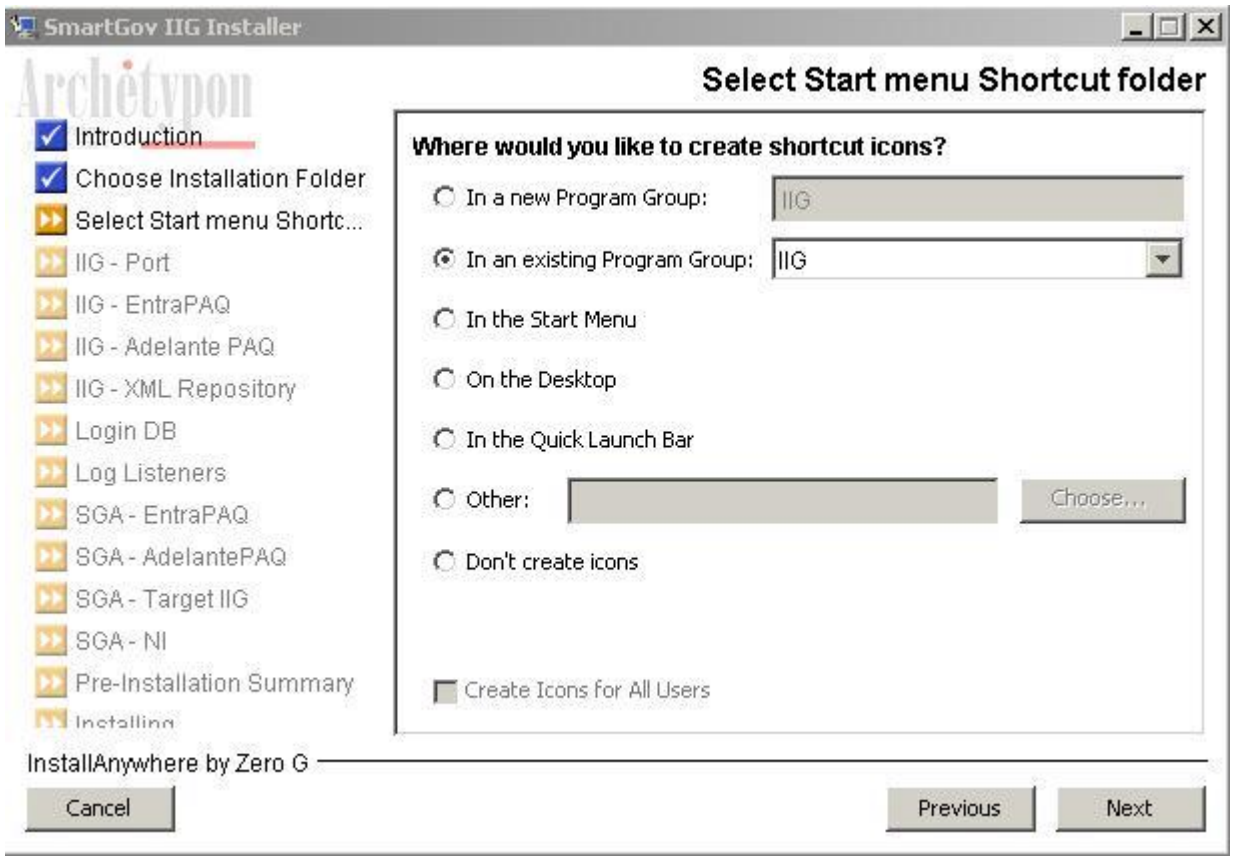
Shown at the beginning of the installation.

4.3.6.2 Installation folder



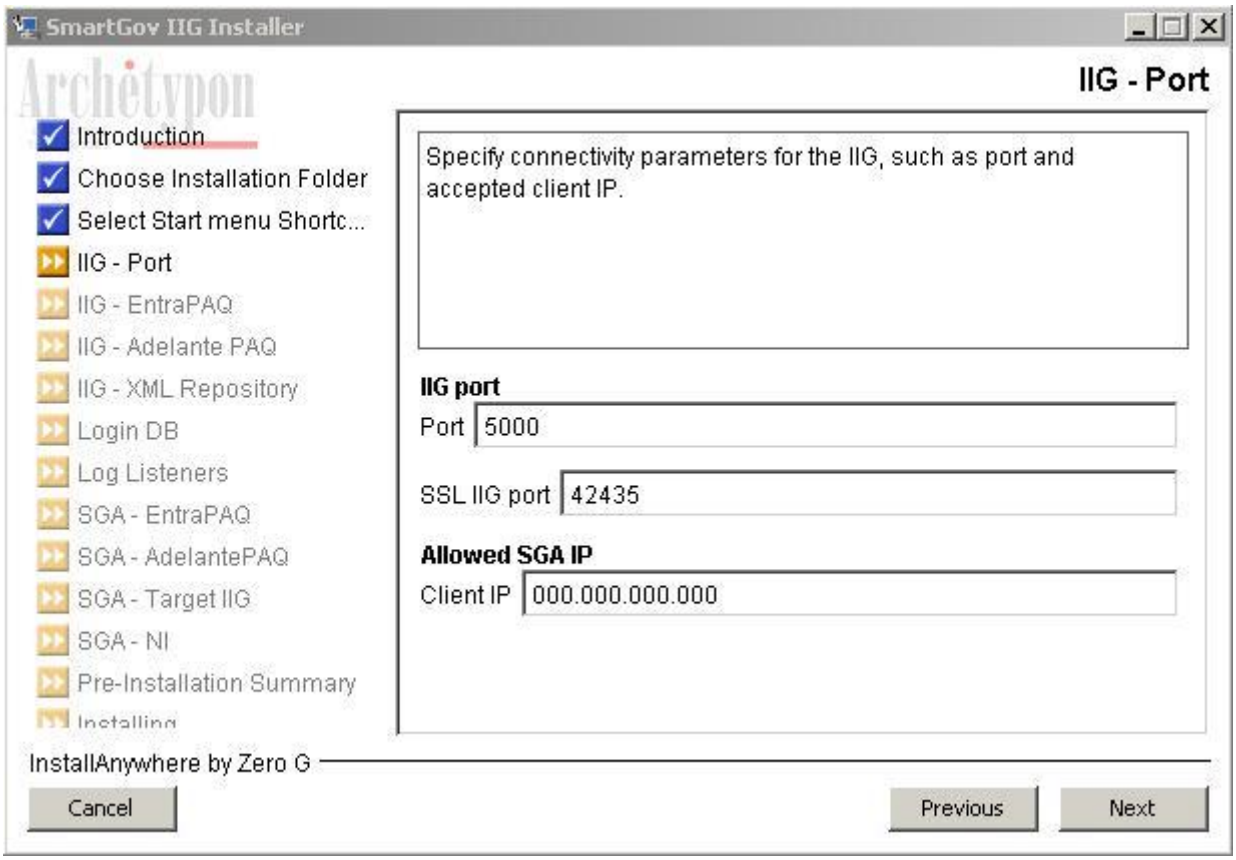
In this screen the destination folder where all program files will be copied is set. The directory path must **not** contain spaces. Moreover, make sure that all back-slashes (\) in the path are changed to **forward-slashes** (/), otherwise the component may not work correctly.

4.3.6.3 Shortcut folder



In this screen the user determines the folder that will contain the component's shortcuts.

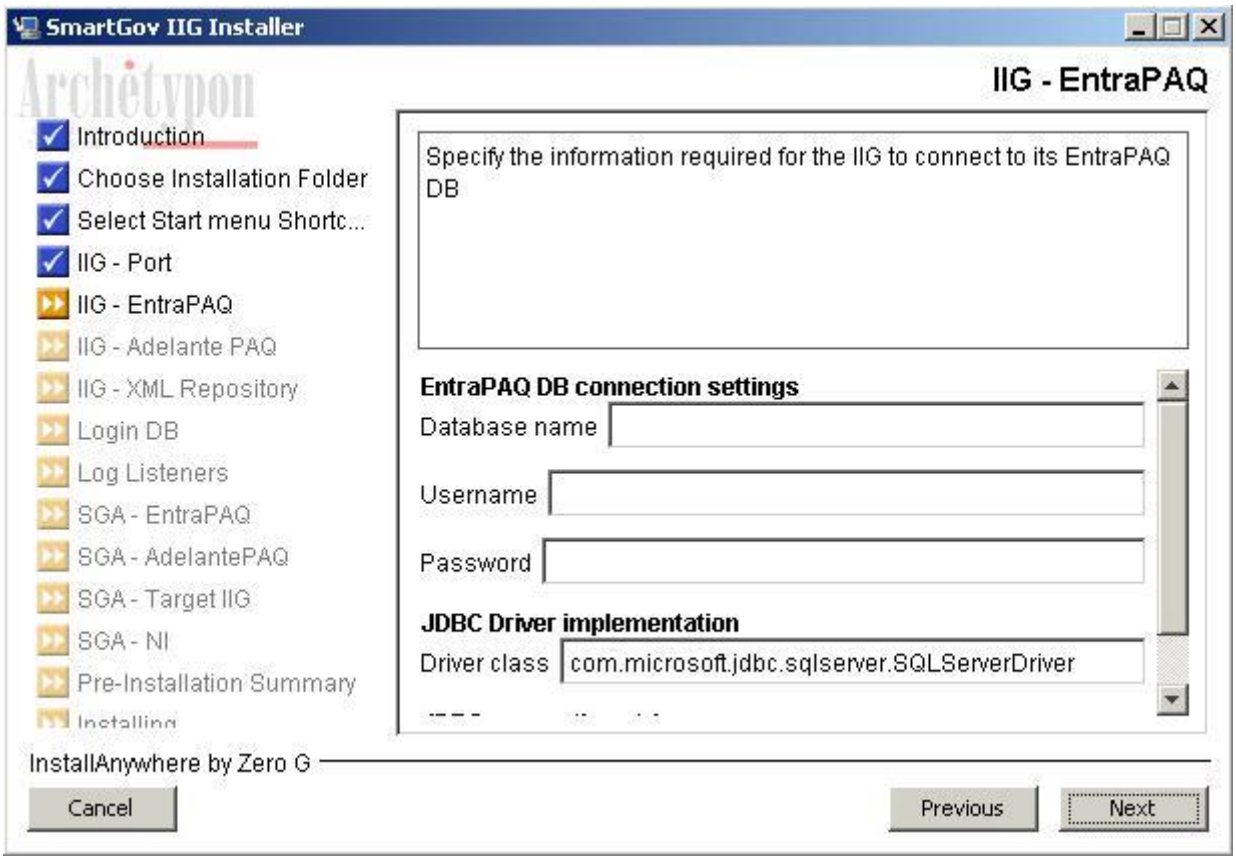
4.3.6.4 IIG ports



This screen specifies the connectivity information for the IIG component. The fields are:

- *Port*: The port where the text-based IIG will listen on
- *SSL IIG port*: the port where the SSL-based IIG will listen on.
- *Client IP*: The allowed client IP. The IIG has a built-in security mechanism that accepts calls from specific IP addresses for each published method call. This field should be filled in with the IP of the service deployment host. Do **not** specify 127.0.0.1 as the IP if you are using localhost: use `Start -> Run -> cmd -> ipconfig` to see the localhost's IP address.

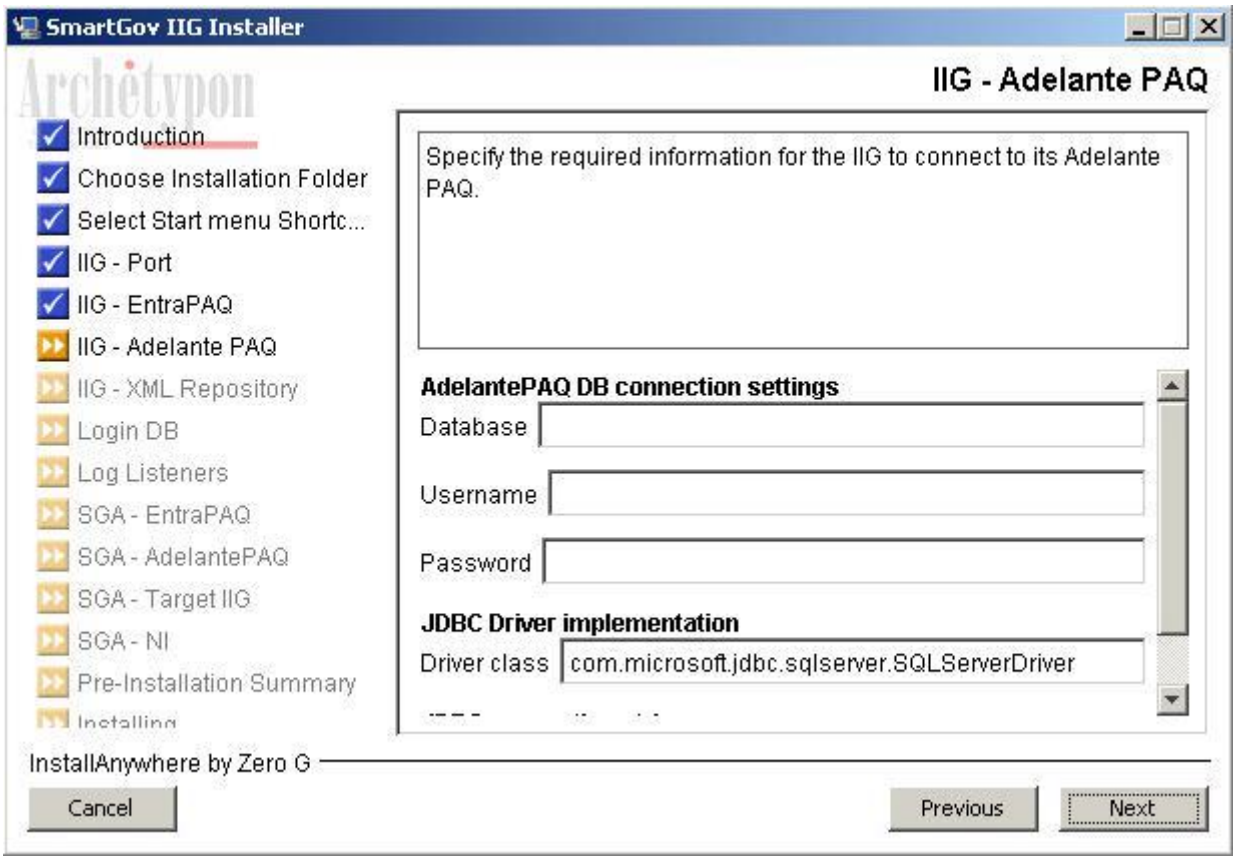
4.3.6.5 IIG EntraPAQ



This information will allow the IIG (both text-based and SSL version) to connect to its EntraPAQ DB. *The information defined in this step should be in line with what was defined in section 4.3.3*. The fields are:

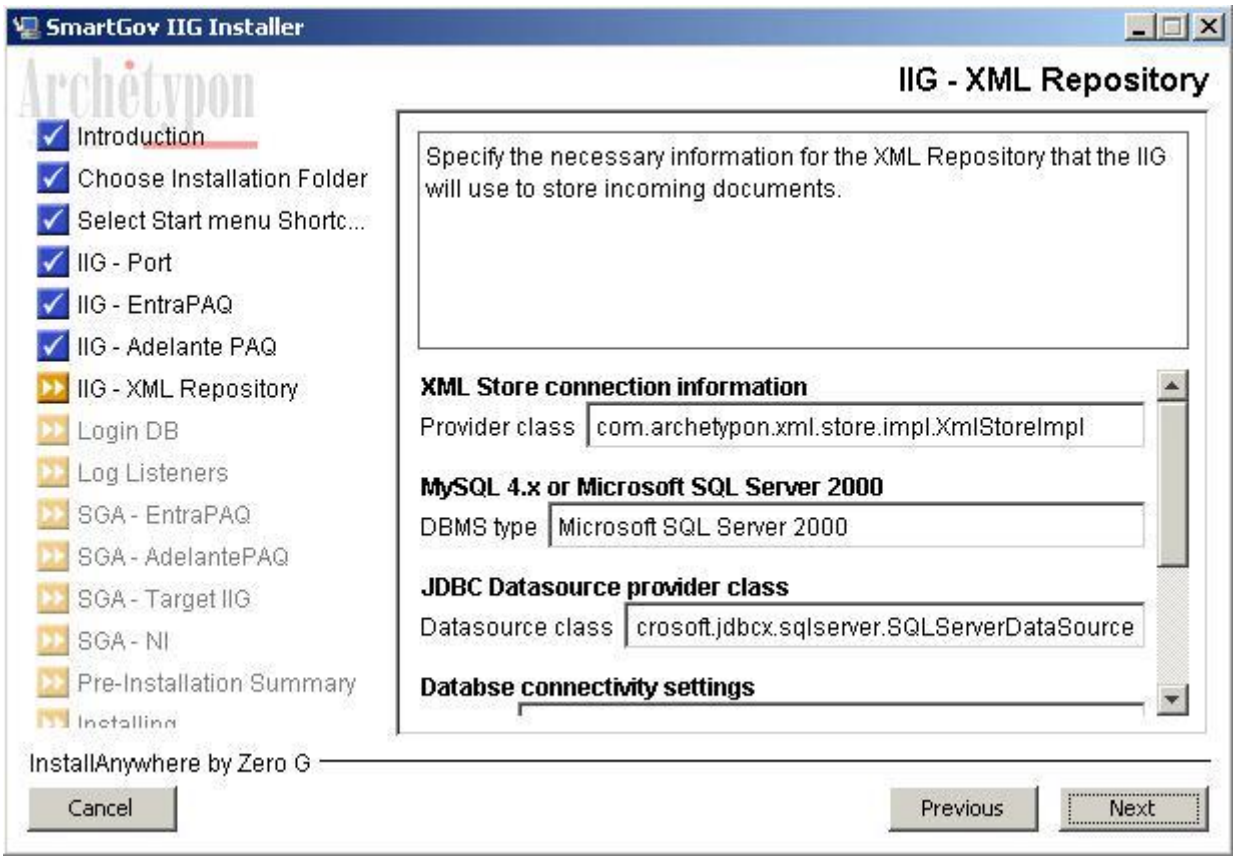
- *Database name*: the name of the physical DB that will host the IIG EntraPAQ
- *Username*: The SQL login used to connect to the DB
- *Password*: the password of the SQL login
- *Driver class*: The JDBC driver implementation class. This may be `com.microsoft.jdbc.sqlserver.SQLServerDriver` for the official SQL Server driver (pre-selected), `org.gjt.mm.mysql.Driver` for the MySQL driver, etc. Consult your driver's documentation.
- *Connect string*: The JDBC connection string to use while connecting to the DB. The value is `jdbc:microsoft:sqlserver://change_the_host_name:1433;SelectMethod=cursor;DatabaseName=` for the SQL Server driver (pre-selected), `jdbc:mysql://change_the_host_name/` for the MySQL driver, etc. In the previous strings, only the host name needs to be changed. Consult your driver's documentation.

4.3.6.6 IIG AdelantePAQ



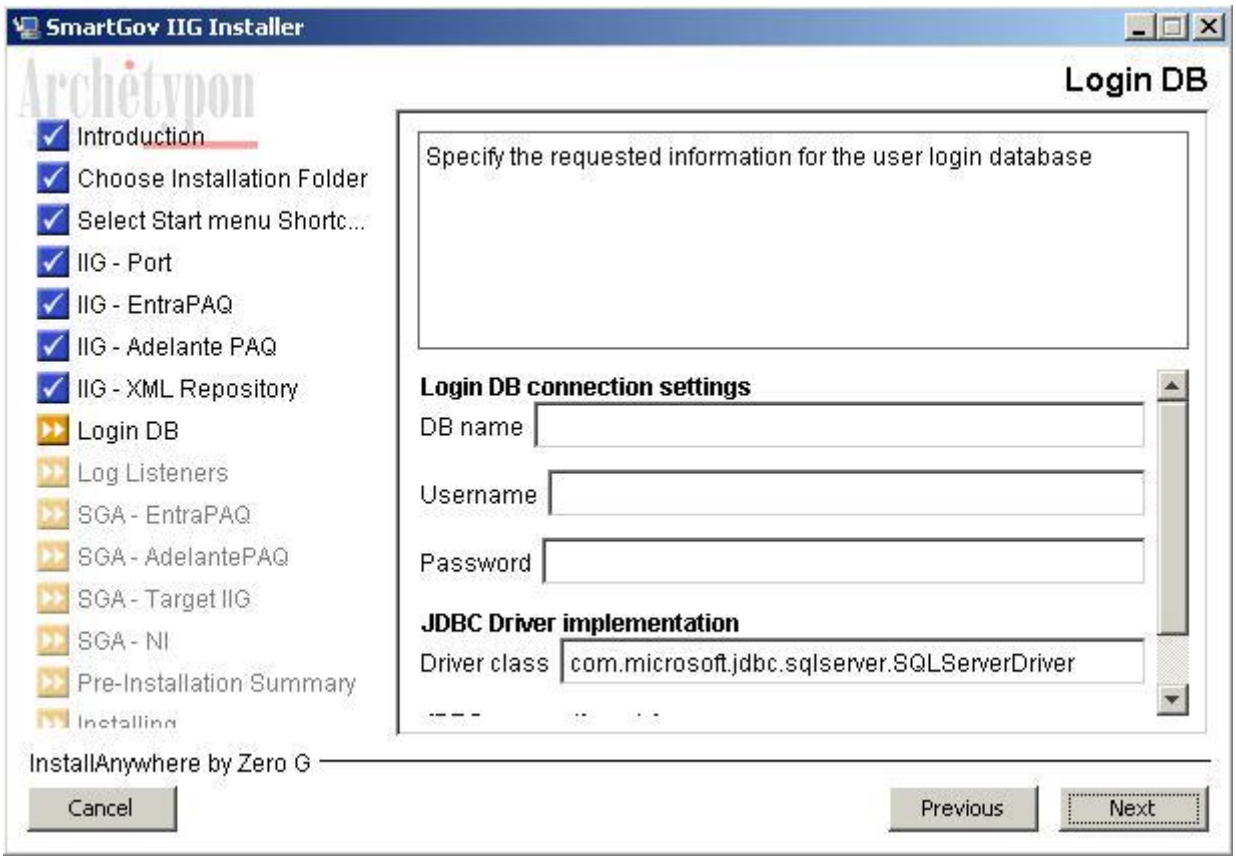
This information will allow the IIG to connect to its AdelantePAQ DB. *The information defined in this step should be in line with what was defined in section 4.3.3*. The fields have the same meaning as in 4.3.6.5.

4.3.6.7 IIG XML Repository



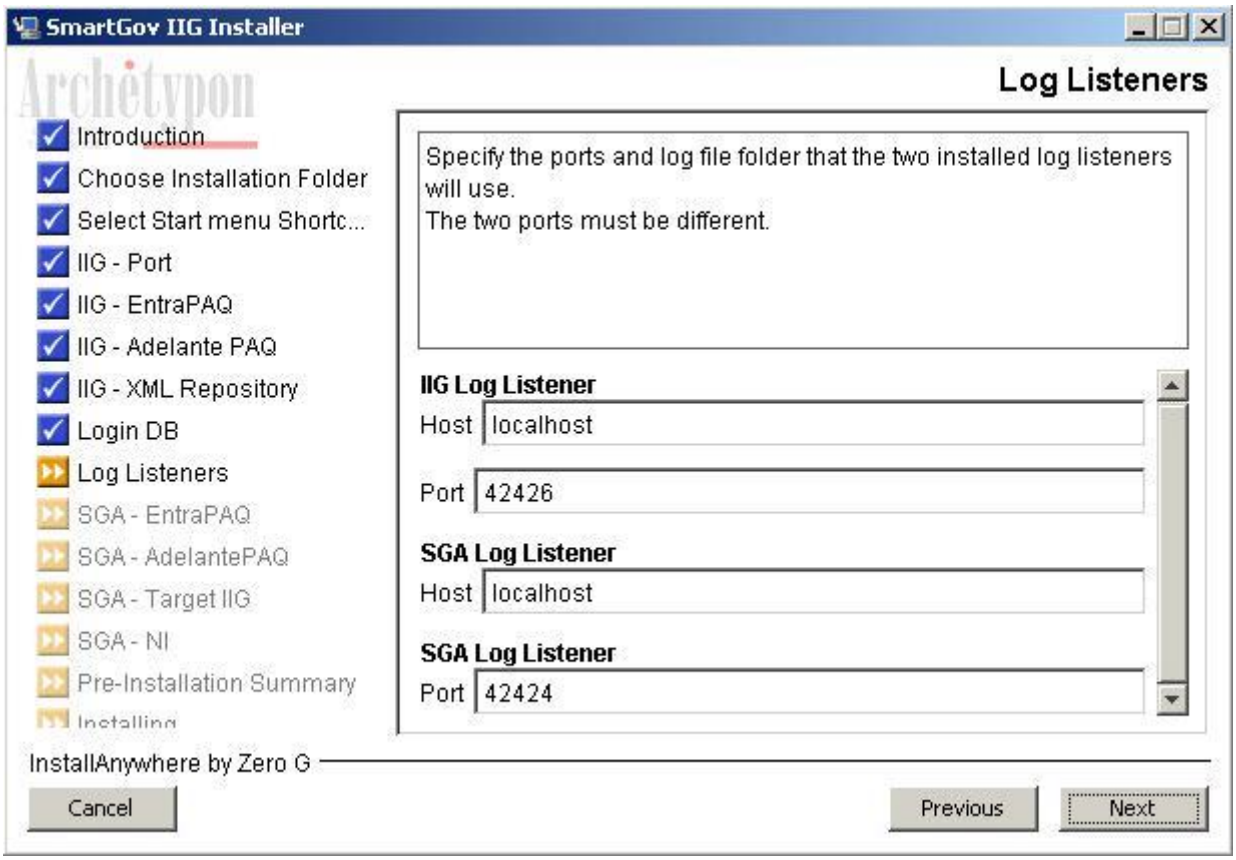
This information will allow the IIG to connect to its local XML Repository. *The information defined in this step should be in line with what was defined in section 4.3.5*. The fields have the same meaning as in section 4.3.2.6.

4.3.6.8 IIG login DB



This screen contains connectivity information for the DB containing service user logins. *The information defined in this step should be in line with what was defined in section 4.3.3*. The fields have the same meaning as in 4.3.6.5.

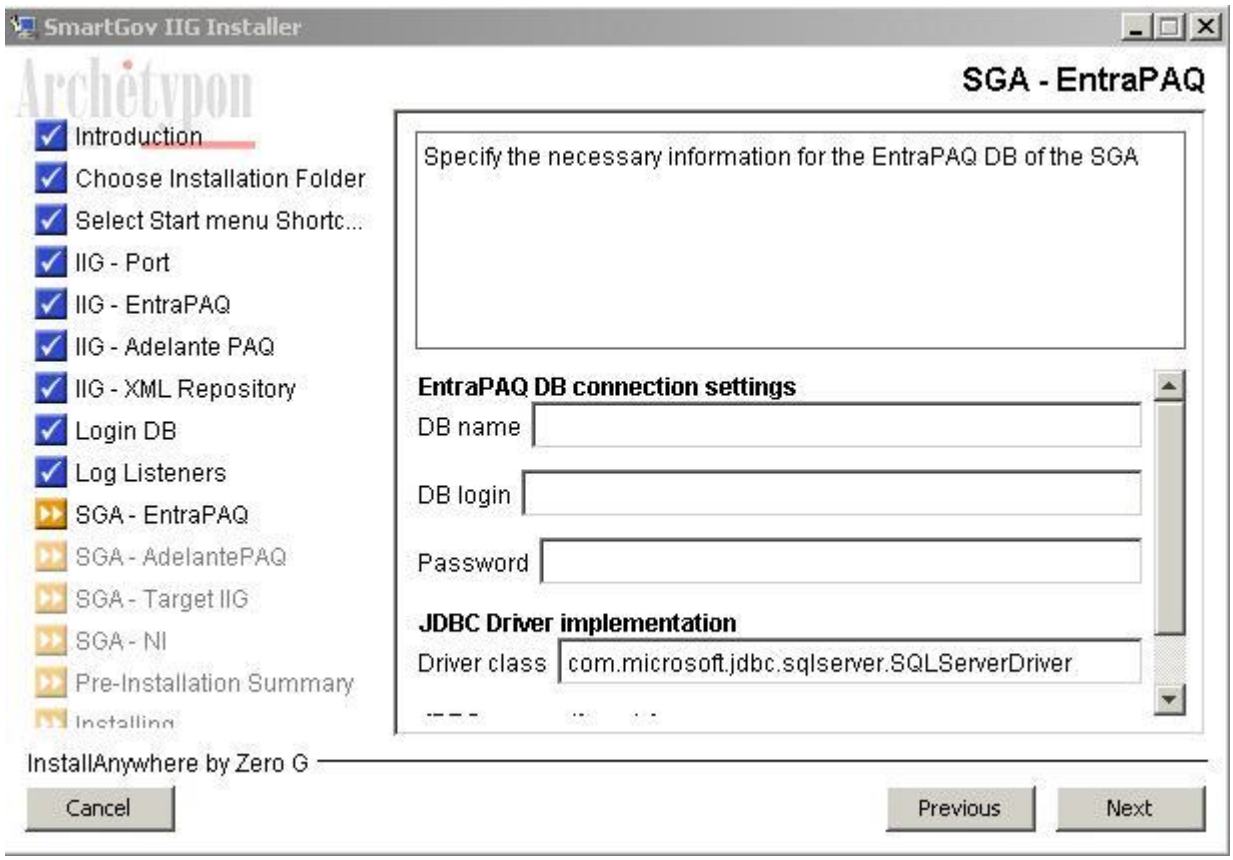
4.3.6.9 Log listeners



This screen allows the user to define the log listeners that will be used to log error messages generated by the IIG and the agent. The fields are:

- *IIG listener host*: Do not change this value, unless you are going to use a log listener other than the one installed by default. In a different case, specify the name of the log listener host.
- *IIG listener port*: Do not change this value, unless you are going to use a log listener other than the one installed by default. In a different case, specify the port on which the other log listener listens on.
- *SGA listener host*: If you are not going to use the bundled, test SGA client, ignore this value.
- *SGA listener port*: If you are going to use this host for the SGA log listener, specify this port accordingly. This field should be in line with the information specified in section 4.3.11.

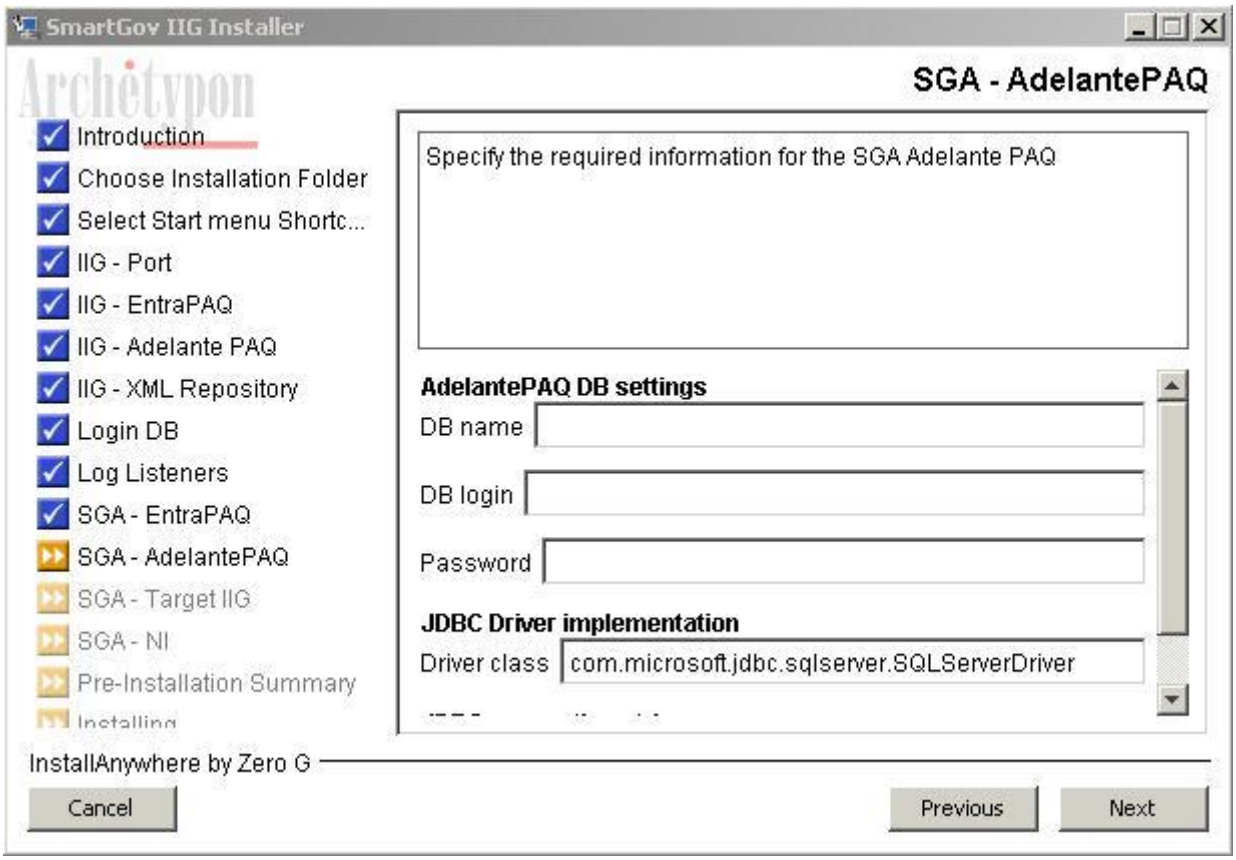
4.3.6.10 SGA EntraPAQ



This information will allow the local, test SGA client to connect to its EntraPAQ DB. *The information defined in this step should be in line with what was defined in section 4.3.3*. The fields have the same meaning as in 4.3.6.5.

Ignore this screen, unless you are going to use the local, test SGA client.

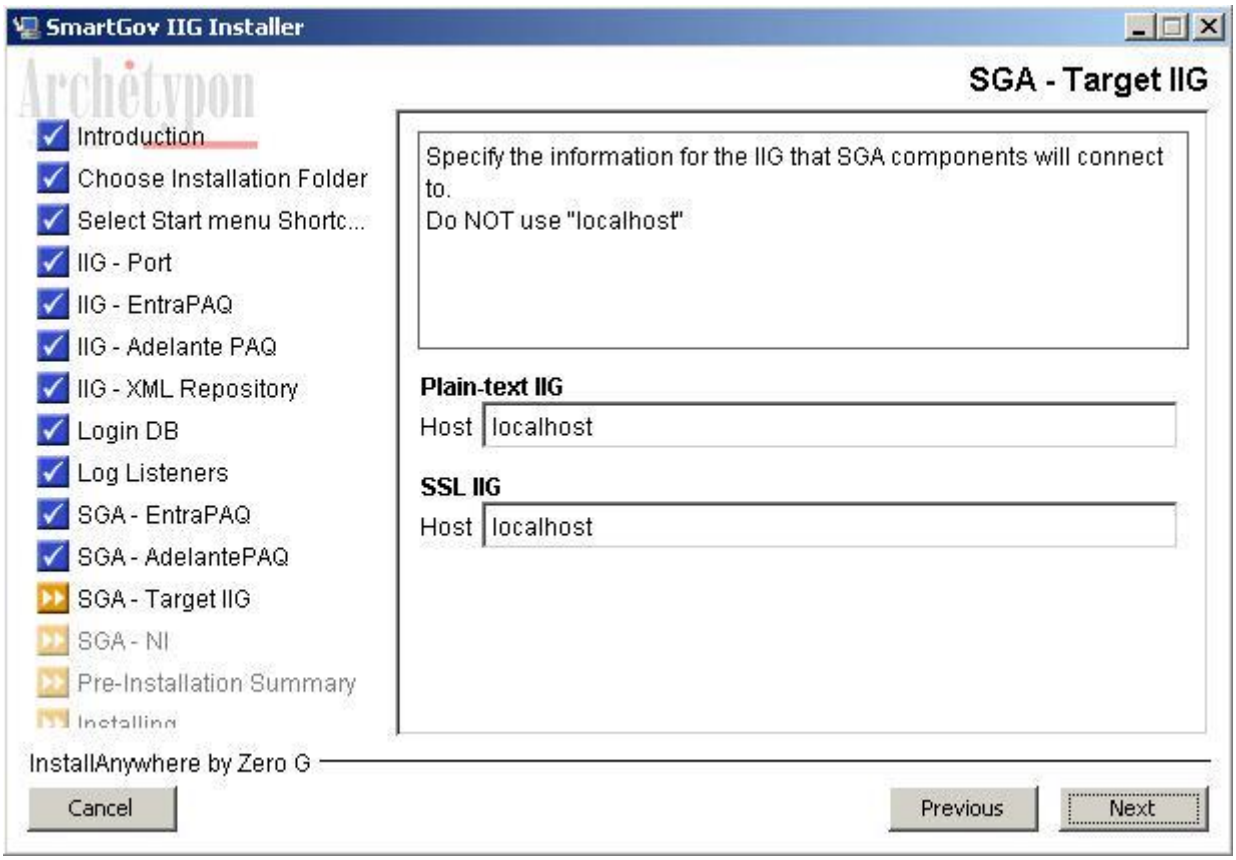
4.3.6.11 SGA AdelantePAQ



This information will allow the local, test SGA client to connect to its AdelantePAQ DB. *The information defined in this step should be in line with what was defined in section 4.3.3*. The fields have the same meaning as in 4.3.6.5.

Ignore this screen, unless you are going to use the local, test SGA client.

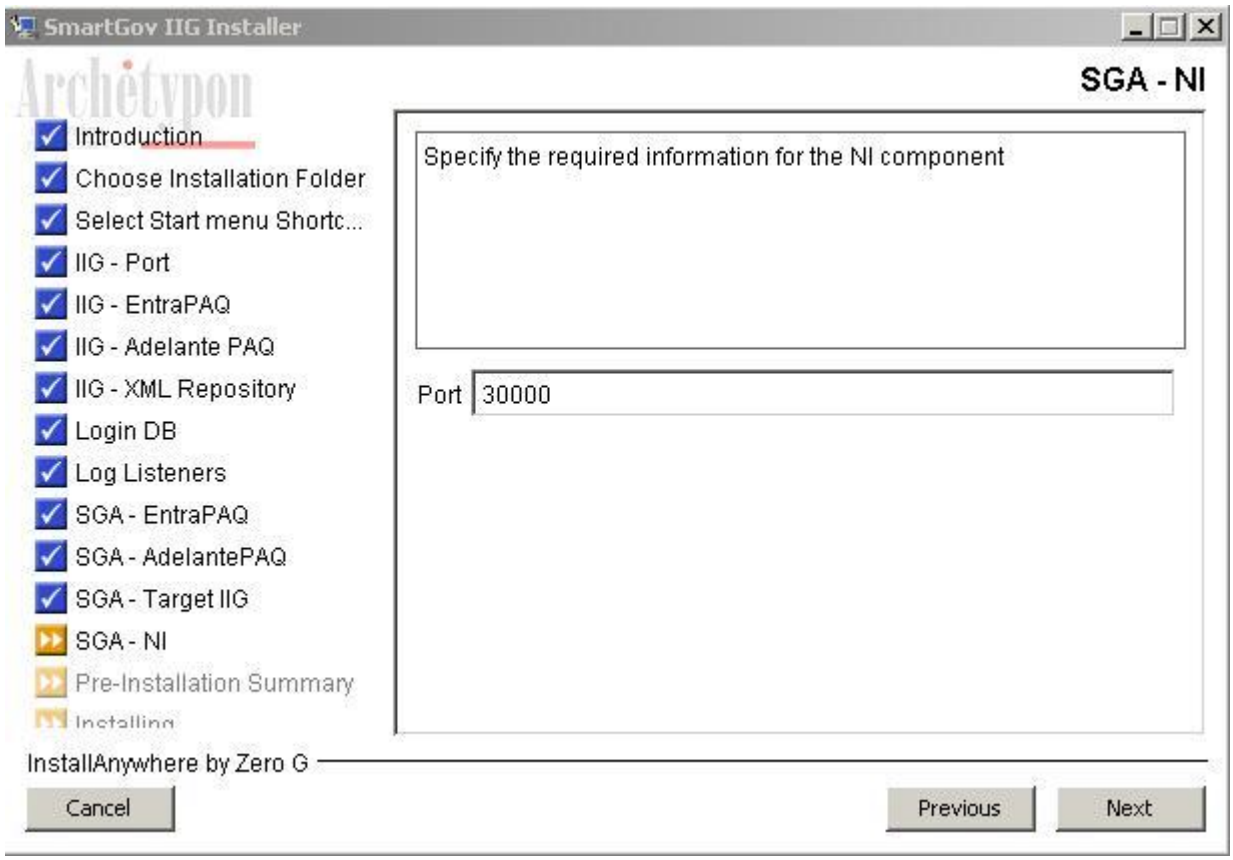
4.3.6.12 Target IIG



This information will allow the local, test SGA client to connect to the two different IIGs.

Ignore this screen, unless you are going to use the local, test SGA client.

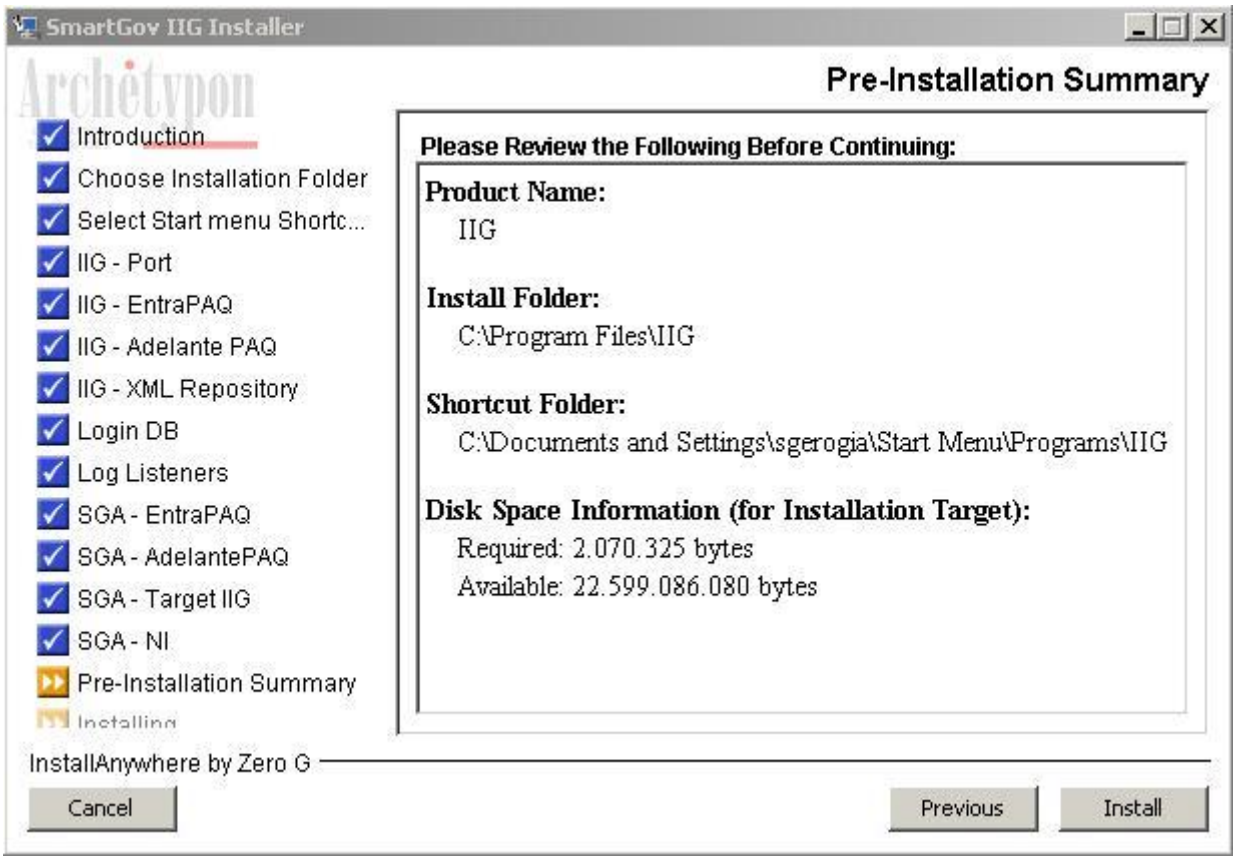
4.3.6.13 SGA NI



This information specifies the port where the SGA NI component will listen on.

Ignore this screen, unless instructed otherwise.

4.3.6.14 Summary



The final screen before the installation begins.

4.3.7 Use / Fine-tune installed IIG

To launch the IIG go to the Start menu and select *Programs -> IIG -> Start IIG Servers*. This will launch

- the IIG server
- the SSL IIG server
- the IIG dispatcher
- the IIG log listener

A successful launch results in 4 different console windows being added in the desktop.

To launch the SGA servers, go to the Start menu and select *Programs -> IIG -> Start SGA Servers*. This will launch

- the SGA log listener
- the SGA dispatcher
- the SGA NI

A successful launch will result in 3 different console windows being added to the desktop. However, if steps 4.3.6.10 to 4.3.6.13 have been ignored, only one

console window should remain open at the end, the log listener. This is also normal.

If you want to re-create the IIG DBs, you will have to follow the instructions found in section 4.3.3. To easily locate the DB creation scripts, from the Start menu, select *Programs -> IIG -> Auxiliary -> SQL scripts*. This directory contains all the SQL scripts to generate the DBs, as included in Appendix D and Appendix E.

The delivered IIG component, by default, supports Microsoft SQL Server 2000 for the deployment of the PAQ and login DBs. If you want to use a different DBMS, then you will need to

- specify the necessary information in screens 4.3.6.5, 4.3.6.6, 4.3.6.8, 4.3.6.10 and 4.3.6.11
- copy the DBMS' specific driver in the appropriate folder. The JARs/ZIPs/classes should be copied in the folder accessible by the Start menu shortcut *Programs -> IIG -> Auxiliary -> Place additional JARs here*.
- Close all IIG console windows and re-launch them from the Start menu shortcut.

The included SSL IIG server is not usable as it misses the necessary security certificates. Before using it, you will have to create them. To do so, select from the Start menu *Programs -> IIG -> Auxiliary -> Create SSL certificates*. You will have to restart the SSL IIG process.

The IIG servers (both text-based and SSL), as installed by the setup program, only allow one SGA agent to access them, located in the IP address specified in 4.3.6.4. If you have more than one deployment hosts and you want all of them to share the same IIG, you will have to explicitly declare the additional IP addresses. This can be done via Start menu shortcut *Programs -> IIG -> Auxiliary -> Allow additional SGAs here*. Edit the XML file with an editor (even WordPad will do), creating copies of the different `IIGCredentials` elements for each new IP address and save the file. You will have to restart the IIG processes for the changes to take effect.

4.3.8 Set up document pre-population

After the login DB has been created in step 4.3.3, a set of XML files should be created to cater for pre-population of document fields, upon end-user login. The idea is the following: when a user logs in to the service for the first time (and, hence, no submitted documents exist), the back-end IIG creates an empty document with certain values pre-filled, such as name, telephone, etc. The

mechanism to do so is quite simple: the IIG looks for properly named XML files in its working directory.

To be able to pre-populate the form with each user's personal data, you will have to do the following

- For each user defined in table `SGUserData` in section 4.3.4, create a file named *username*.xml, inside folder *<IIG install dir>\defaultXml*. Here, *username* denotes the same value as the one added in column username in table `SGUserData`.
- Edit the file with Notepad or an XML editor (e.g. XMLSpy) and insert the XML excerpt found in Appendix F. Replace all XXX placeholders with the appropriate values and save the file. If there are values that have characters non-Latin (e.g. Greek) the file **must** be saved as UTF-8. All date values must be saved in the **DD/MM/YYYY** format. XML editors do so by default. If using Notepad, you should select from the menu *`File -> Save As -> Encoding -> UTF-8`*.

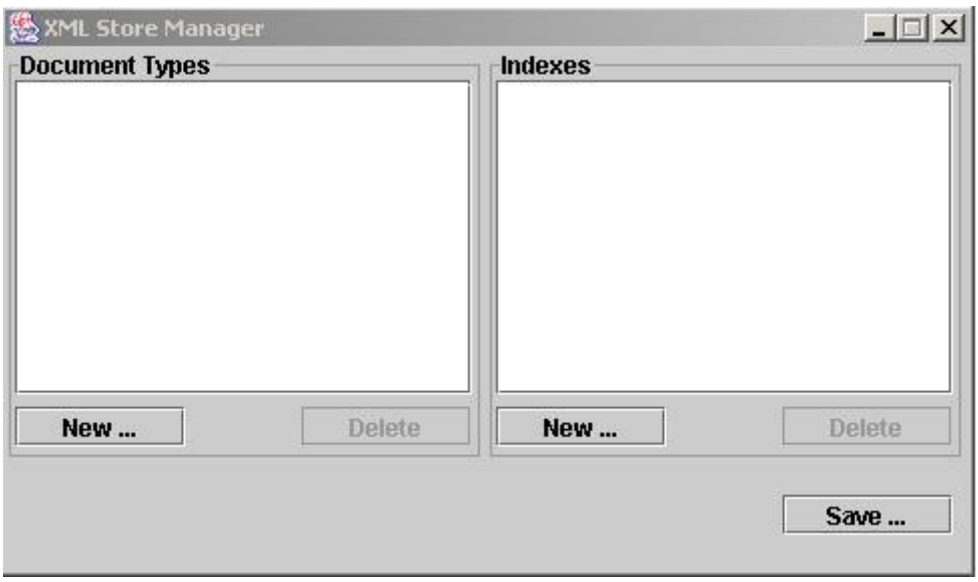
Important note: The XML structure found in Appendix F is only suitable for the eVies service. A different service requires a different document. Moreover, you cannot have two services that require automatic initial pre-filling using the same IIG server. If you feel that these issues are important, send an email to SmartGov@archetypon.gr for more information.

4.3.9 Create Integrator XML Repository

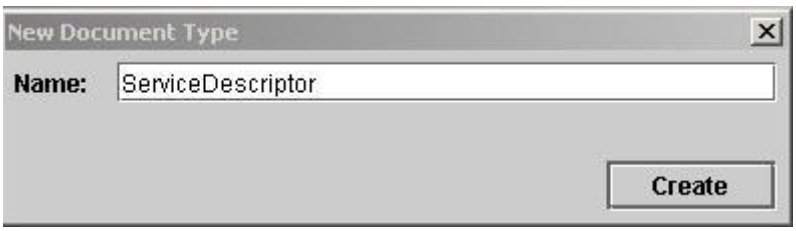
The Integrator, when installed selecting the *Normal setup* option, comes bundled with a visual management tool to assist in the creation of the back-end relational DB.

To create the XML Repository DB perform the following steps

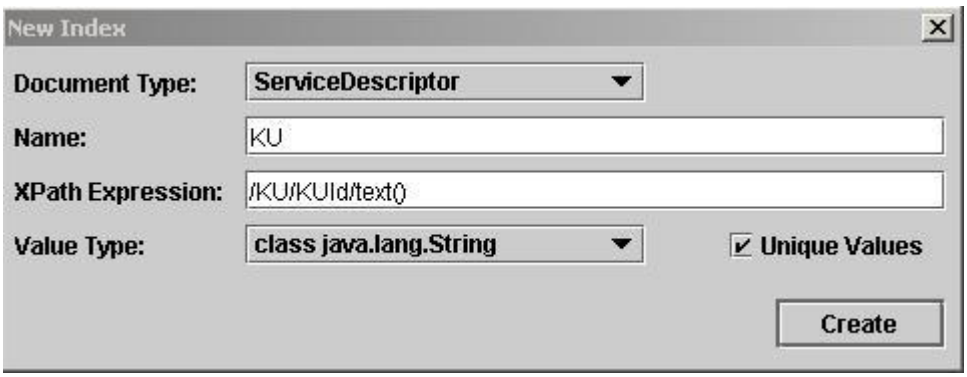
1. Create a DB through the target DBMS' user interface. Only MS SQL Server 2000 and MySql v.4.x are supported.
2. Execute the XML Repository Manager application from the Start menu, selecting *`Programs -> SmartGov -> XmlStore Manager`*.



3. Press the left **New...** button to create a new document type for service description files. Name it `ServiceDescriptor`.



4. Press the right **New...** button to create a new index for the created document type.



For the Integrator to work correctly, five different indexes need to be created in total in the same manner.

Their details follow (case-sensitive, the Name might be different):

- Name: **KU**
    i. XPath: `/KU/KUId/text()`
    ii. Value type: `string`
    iii. Unique
- Name: **Form**

i. XPath: `/form/formId/text()`

ii. Value type: `string`

iii. Unique

- Name: **InstantiatedTSE**

i. XPath: `/instantiatedTSE/instantiatedTSEId/text()`

ii. Value type: `string`

iii. Unique

- Name: **InstantiatedTSEGroup**

i. XPath: `/instantiatedTSEGroup/instantiatedTSEGroupId/text()`

ii. Value type: `string`

iii. Unique

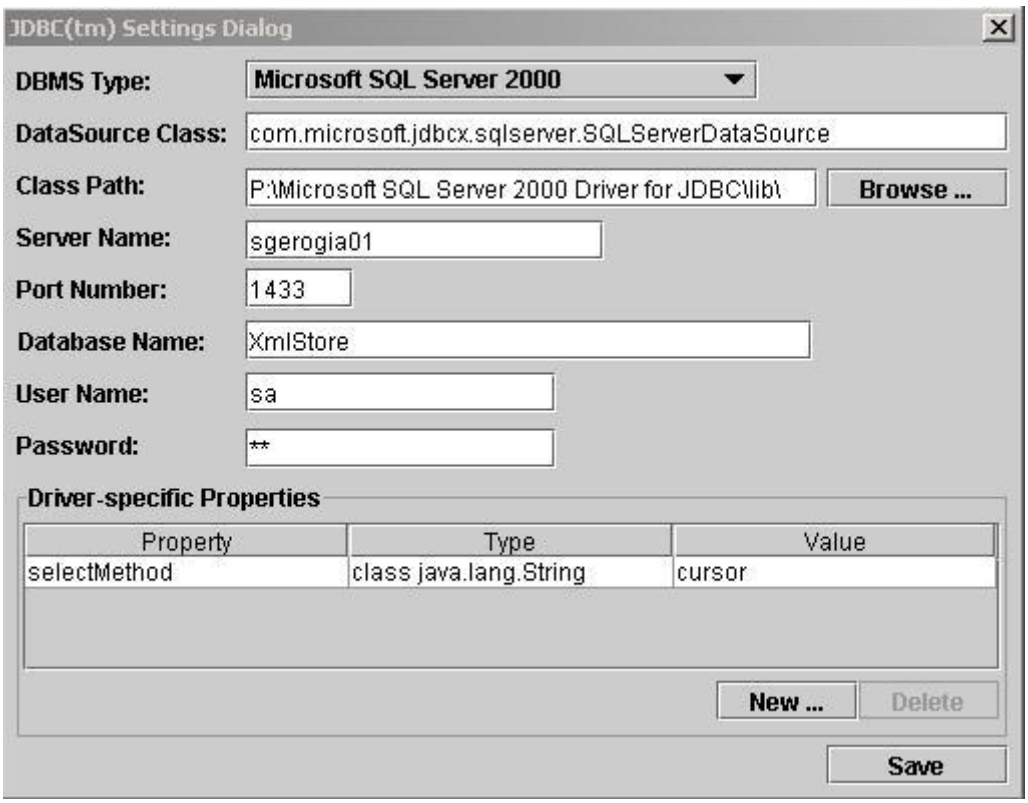
- Name: **TS**

i. XPath: `/TS/TSId/text()`

ii. Value type: `string`

iii. Unique

5. After all indexes are created and listed in the main window (make sure the Document Type list entry is selected), the **Save...** button is pressed.



The driver property has been added by using the **New...** button[1]. The class path can either be a directory or a JAR file, inside which all driver classes reside.

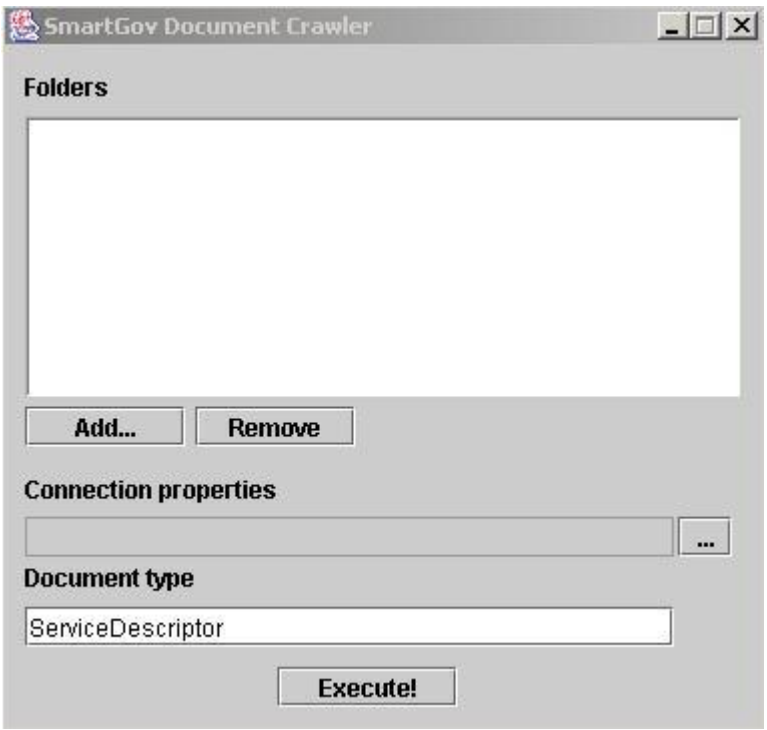
6. Pressing the **Save...** button creates the tables. Open the target DB with a DB client application to verify the creation of the new tables.

4.3.10 Populate Integrator XML Repository

The Document Crawler is a visual utility that crawls through any number of folders and stores all encountered files inside a specified XML Repository. Directory crawling is **not** recursive and all non-valid XML files are ignored. To be able to use this tool, you must select Normal setup during Integrator installation.

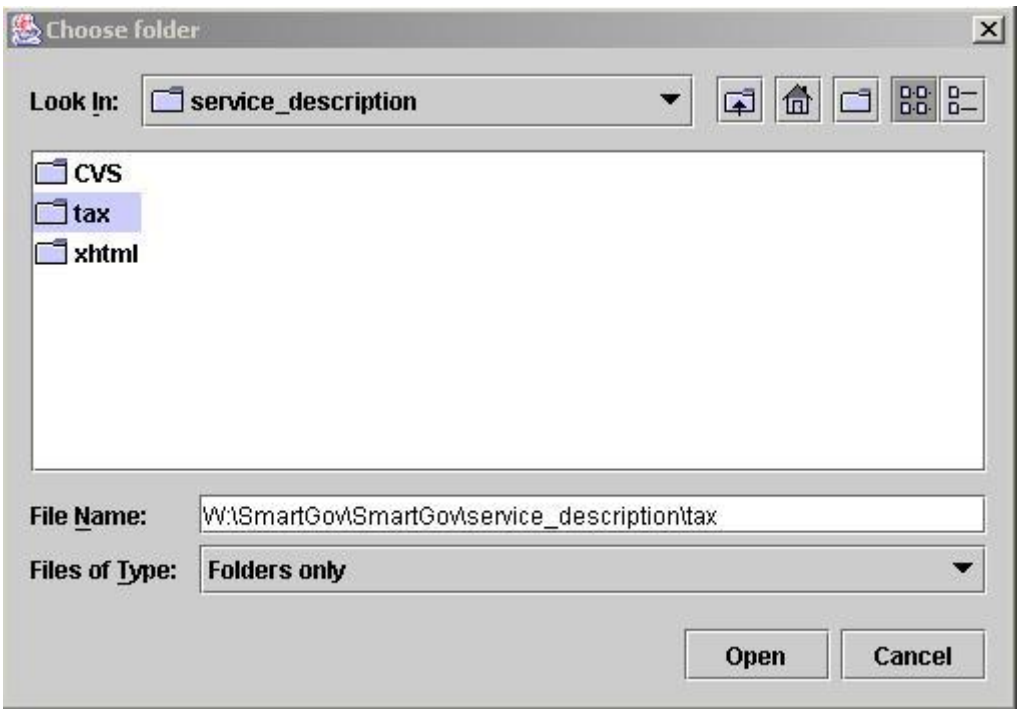
To add files to the XML Repository

1. Launch the Document Crawler from the Start menu, in *Programs -> SmartGov -> Document Crawler*.

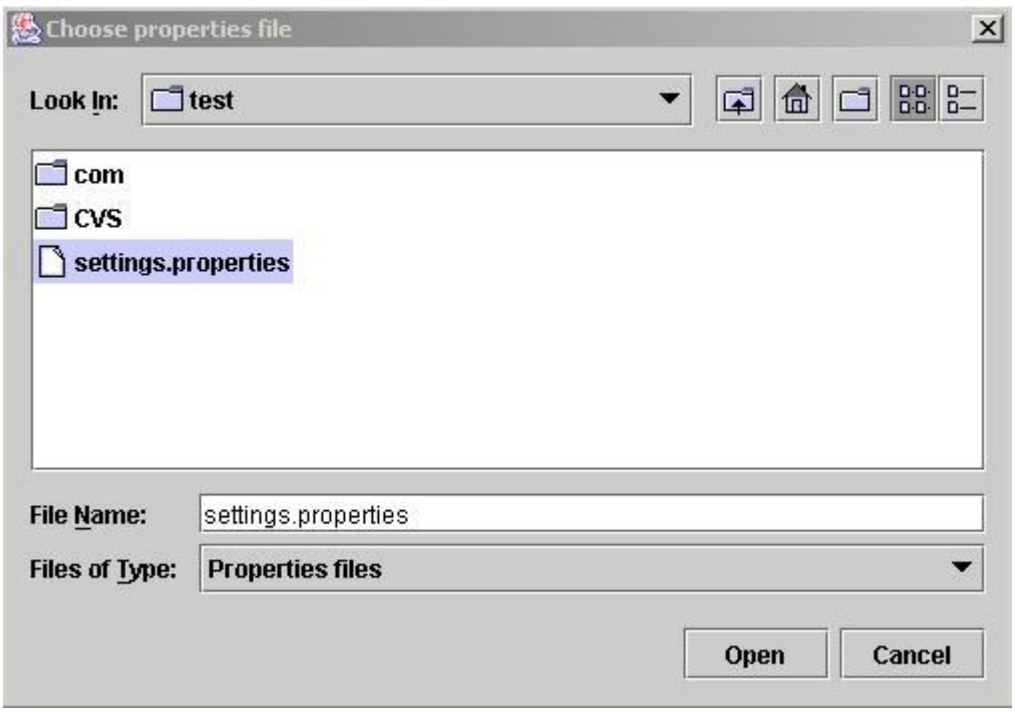


2. Pressing the **Add..** button a file dialog appears, allowing the user to select any folder.

[1] This particular property MUST be set as shown in the image in the case of SQL Server using the Microsoft JDBC driver.



3. This process can be repeated any number of times to include all directories that hold service description files. To remove a directory from the list, select it and press **Remove**.
4. To be able to connect to the XML Repository, we need to specify a connection properties file. Pressing the ellipsis button (**...**) allows us to locate it.

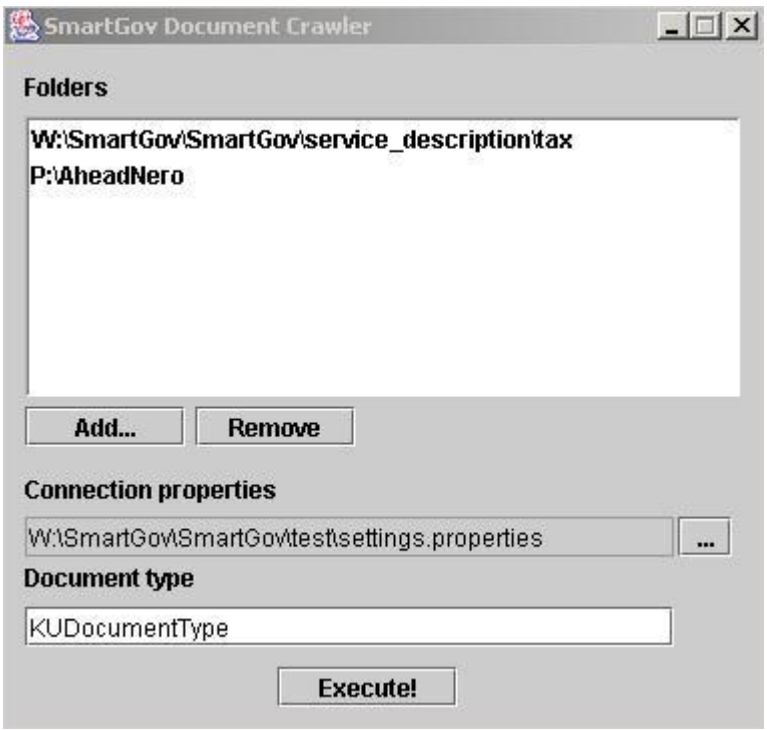


It is a good idea to re-use the connection properties file created during the Integrator installation. The path to this file is *<deployment_Tomcat_dir>\webapps\Integrator\scripts\integrator.properties*.

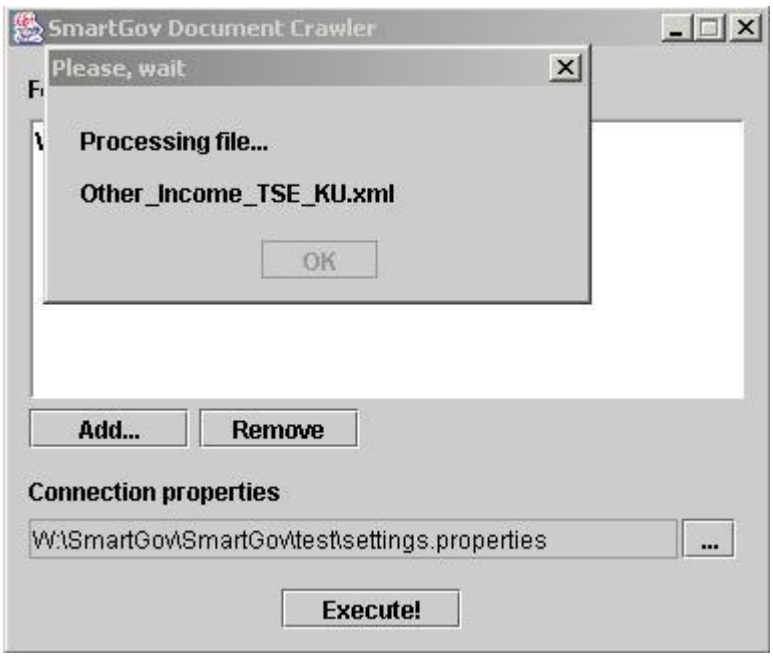
5. Finally, you will need to specify the document type that the added documents will belong to, by editing the last field. The value entered here will have to be the same as the one entered in step 3, section 4.3.9. After all parameters have been set, we can press the **Execute!** Button.

   It should be noted here that all documents "belonging" to the specified document type already in the repository will be **removed**. This allows you to "update" existing files in the repository en masse.

   It is a good idea to use the tool to populate the repository with files belonging to the same service each time, i.e. do not specify directories containing XML files belonging to many services. This is necessary because each time the tool is used, all documents belonging to the specified document type are deleted. To be able to update the description files for a service, you should specify the document type these files have been saved as. If descriptor files belonging to a different service are saved under the same document type, then they will be removed as well, which is probably wrong.



6. During execution, the tool provides feedback on which files are being processed.



4.3.11 Configure the deployment server

During Tomcat installation at a certain point it asks which JVM to use. Avoid specifying the JRE proposed by default (usually located in *c:\Program Files\…*). Instead browse for the installation directory of the JDK v.1.4+. The installation directory of the JDK should **not** contain spaces.

After installing Tomcat, go to *My Computer -> <right click> -> Properties -> Advanced -> Environment Variables*. Make sure that the following variables are added in the `System Variables` pane

- `JAVA_HOME`: The value of the variable is the installation directory of the JDK.
- `CATALINA_HOME`: The value of the variable is the installation directory of Tomcat.

After performing these changes, restart Tomcat (the executable or the service).

The Integrator employs the remote deployment feature found in Tomcat v.4.1.18+. to do so, and to avoid compromising the deployment server's security, the Integrator assumes that there is a `smartGov` login configured on the target server with `manager` privileges. To create this login on the deployment host:

- Locate and edit the Tomcat user configuration file. This is located in <deployment Tomcat install dir>\conf\tomcat-users.xml.
- Add the following entry right before the </tomcat-users> XML tag. Save and restart Tomcat.

```
<user username="smartGov" password="smartGov" roles="manager"/>
```

The Integrator installer copies a ZIP file containing a ready-to-use set of configuration files for the service SGA. This ZIP file should be copied and extracted to the deployment server.

The ZIP file can be accessed via the Start menu, shortcut *Programs -> SmartGov -> SGA Configuration files*. Extract the contents of the file in the deployment host(s) under folder *c:\SmartGov*.

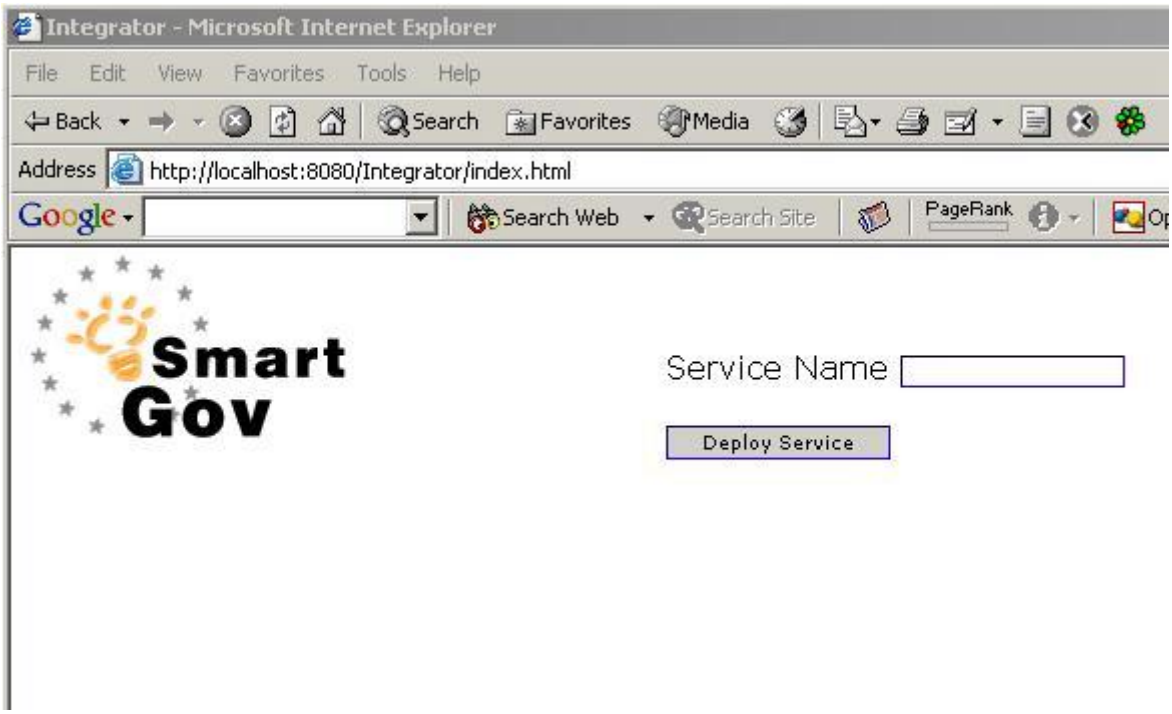
If you want to…

- **extract the configuration files into a different directory**: Extract the file into any directory. This path should be specified in the Integrator installer, as shown in section 4.3.2.8. Update the paths in file *<extraction_dir>\sga\conf\SGAConfig.txt*.
- **change the host and port where the IIG servers listen on**: Edit file *<extraction_dir>\sga\conf\SGAIIGConf.xml*. Change the ports to reflect the values specified in section 4.3.6.4. change the host IP to be that where the IIG process are executed. Be careful to **only** specify an IP, not a machine name.
- **change the SGA EntraPAQ DB to be used**: Edit file *<extraction_dir>\sga\conf\EntraPAQConfig.txt*. Change the values as necessary. The semantics are the same as explained in section 4.3.6.5.
- **change the SGA AdelantePAQ DB to be used**: Edit file *<extraction_dir>\sga\conf\AdelantePAQConfig.txt*. Change values as necessary. The semantics are the same as in 4.3.6.5.
- **change the SGA log listener to send error messages to**: Edit file *<extraction_dir>\sga\conf\SGLogConfig.txt*. Change values to be in line with those specified in 4.3.6.9.

4.4 Usage Guide

The Integrator component is accessible as a web-based application.

The URL to launch the Integrator is `http://development:port/Integrator`, where `development` is the name of the development host and `port` is the port where the development Tomcat server is listening.



Usage of the tool is really simple, as the only required parameter from the user is the unique id of the service, which the Integrator shall process and deploy.

Pressing the **Deploy Service** button, the Integrator process is launched and its different sub-tasks are executed. Feedback is provided on-screen in the lower part of the screen. Normal output is colored light blue, while error messages are colored light brown.

Error messages are an indication for technical personnel to correct errors is service definition files (misspellings, missing files, DB failures…). A successful build and deployment is indicated with a **BUILD SUCCESSFUL** final message.

5 Communication services: SmartGov Agents. Installation, configuration and usage

The SmartGov platform includes facilities for communication between the service delivery platform and the organisational information systems. These facilities are an indispensable part of the SmartGov platform and must be installed and configured for the service delivery platform to operate successfully. In this section, the procedures for installing and configuring the communication services are documented and details on the operation of the services are provided.

It should be noted that the users of the SmartGov services will be the IT staff and system administrators of the PAs, thus the documentation provided is necessarily technical and detailed.

5.1 Prerequisites

Before proceeding to install the SmartGov Agents bundle, it should be verified that:

1. The Java runtime environment version 1.4.1 or higher is installed on the system
2. The PATH variable includes the directory in which the java executable resides (typically, the bin directory under the JRE installation directory).

   This can be ascertained by issuing the following command in a shell window:

   `java –version`

   If an error message indicating that the `java` command cannot be found is printed or the version reported is older than 1.4.1, then the system administrators should proceed in installing the appropriate JRE, or configuring the PATH variable.

The administrators should also verify that the appropriate JDBC drivers for the DBMS that will host various necessary tables are installed in the system and the DBMS runtime libraries, if required, are present and registered to the execution environment (e.g. in the `LD_LIBRARY_PATH` variable for Solaris systems).

5.2 Bundle contents and installation

The communication services bundle includes the following files:

1. `SGAgent.jar`. This file contains the code implementing the SmartGov communication services, both for the service delivery environment and the organisational information system. The file may be placed anywhere in the file system, however it must be added to the locations searched by the

Java Runtime Environment (JRE) for class loading. These locations are commonly listed in the CLASSPATH environment variable, or specified via the `-cp` or `-classpath` flag to the Java virtual machine, as illustrated in the following examples:

Windows Platform

```
mkdir c:\smartgov
mkdir c:\smartgov\sga
copy SGAgent.jar c:\smartgov\sga
java -cp C:\smartgov\sga\SGAgent.jar gr.uoa.di.SGANI.SGANIFactory
         c:\smartgov\SGANIPropertyFile.txt
```

or

```
mkdir c:\smartgov
mkdir c:\smartgov\sga
copy SGAgent.jar c:\smartgov\sga
set CLASSPATH c:\smartgov\sga\SGAgent.jar
java gr.uoa.di.SGANI.SGANIFactory c:\smartgov\SGANIPropertyFile.txt
```

Unix Platform

```
mkdir /usr/smartgov
mkdir /usr/smartgov/sga
cp SGAgent.jar /usr/smartgov/sga
java -cp /usr/smartgov/sga/SGAgent.jar gr.uoa.di.SGANI.SGANIFactory
        /usr/smartgov/sga/SGANIPropertyFile.txt
```

or

```
mkdir /usr/smartgov
mkdir /usr/smartgov/sga
cp SGAgent.jar /usr/smartgov/sga
CLASSPATH=/usr/smartgov/sga/SGAgent.jar; export CLASSPATH
java gr.uoa.di.SGANI.SGANIFactory /usr/smartgov/sga/SGANIPropertyFile.txt
```

(The line `CLASSPATH=...` in the latter case applies for the Bourne, Korn and Bash shells; for C-Shell and tcsh, the line

`setenv CLASSPATH /usr/smartgov/sga/SGAgent.jar`

should be used instead)

The CLASSPATH variable may also be set in a machine-wide fashion. For Microsoft Windows this can be accomplished by right-clicking on `My Computer` and selecting `Properties`, then selecting `Advanced` and `Environment Variables` and, finally, adding or modifying the CLASSPATH variable. For Unix environments, the OS vendor's instruction for setting platform-wide variables should be consulted.

2. `SGAconf.zip`. This file contains the XML configuration files, the document type definition (DTD) files and the component property files required for

the operation of the communication services. The file SGAconf.zip is a zip archive, which must initially be unzipped. The unzip process will create two top-level directories named "windows" and "unix", for use in the respective environments (unix subsumes linux). Practically, any configuration may be used in either environment; the two separate folders are provided for convenience, to minimise the required editing. Configuration files in the windows directory assume that they will be placed under c:\smartgov while configuration files in the unix directory assume that they will be installed in /usr/smartgov. Each of the directories contain a subdirectory conf, which in turn contain two subdirectories namely sga and iig. The directory conf/iig contains all the files needed for operation of the Information Interchange Gateway (the component attached to the organisational information system), while the directory conf/sga contains all the files needed for operation of the SmartGov agent (the component running on the service delivery environment). For more information on the configuration files, see sections "Configuration, Property And DTD files" and "Package Documentation".

3. SQLscripts.zip. This zip archive contains scripts that initialise the databases used by the communication services. Three scripts are included in the archive, namely sgaora.sql, sgamysql.sql and sgamssql.sql to be used with Oracle, MySQL and MS SQL Server, respectively. For more details see section "Database Setup".
4. SGsyssvc.jar. This Java archive contains the implementations of the services required for the operation of the service delivery environment. The zip file contents must be extracted to some file system location and the IIGCommMethConfFile.xml configuration file should be edited accordingly to point to the actual location of the extracted files. For more information on configuring the system services, see section "SmartGov system services".
5. scripts.zip. This directory contains batch scripts for starting the SmartGov communication services and for generating the appropriate certificates to be used with the Secure Socket Layer services. Of course, certificates provided by Certificate Authorities may be used with the Secure Socket Layer, however organisations may create their own, self-signed certificates for their installations, in order to minimise costs. For more information on the SSL certificates, see section gr.uoa.di.SSLIIGServer Package.

5.3 Configuration, Property And DTD files

A number of property files are necessary for providing the value of parameters required by the various modules of the SmartGov platform. These property files are described in detail in the package documentation section. A list is provided here, along with a summary of their contents and where there should be provided as a parameter.

5.3.1 Property files

Property file	Bundle filename	Comments
SGLogger property file	`conf/sga/SGLogConfig.txt` `conf/iig/SGLogConfig.txt`	The SGLogger property file defines the address details that the SGLogger entities may use to contact the SGLogListener. Two files are provided, to be used in the service delivery environment (SGA) and the organizational information system (IIG) respectively. The location is passed as a parameter to the *newSGLogger* method of the *SGLoggerFactory* class of the *gr.uoa.di.SGLogging* package. If the `SGUtil.logMessage` method is used, the system property `SGLogger.propertyFile` should be set to point to the location of the property file. For more information on the SGLogger property file, see section "gr.uoa.di.SGLogging Package".

Property file	Bundle filename	Comments
SGLogListener property file	`conf/sga/SGLogListenerConf.txt` `conf/iig/SGLogListenerConf.txt`	The SGLogListener property file defines parameters used by the SGLogListener, such as the port to listen to and the log file destination. Two files are provided, to be used in the service delivery environment (SGA) and the organizational information system (IIG) respectively. Its location is provided as a parameter when the *gr.uoa.di.SGLogListener.SGLogListenerFactory* class is executed. For more information on the SGLogListener property file, see section "gr.uoa.di.SGLogListener Package".
SGA-NI property file	`conf/sga/SGANIConfig.txt`	The SGA-NI property file defines parameters used by the SGA Notification Interceptor, such as the port to listen to and the location of the EntraPAQ configuration file. Its location is provided as a parameter when the *gr.uoa.di.SGANI.SGANIFactory* class is executed. For more information on the SGA-NI property file, see section "gr.uoa.di.SGANI Package".
EntraPAQ property file	`conf/sga/EntraPAQConfig.txt` `conf/iig/EntraPAQConfig.txt`	This file contains details for accessing the queue of tasks built due to reception of notification events (SGA side) or non real-time requests (IIG side). Its location is provided as a property (`SGA.EntraPAQ.propertyFile=<property file spec>`) in the SGA-NI property file (SGA side) For more information on the EntraPAQ property file, see sections "gr.uoa.di.SGANI Package" and gr.uoa.di.IIGMyP package.

Property file	Bundle filename	Comments
IIG-NI property file	`conf/sga/IIGNIConfig.txt`	This file contains configuration details for the IIG notification initiator in the form of pointers to other configuration files. Its location is provided as a parameter to the *newIIGNI* method of the *gr.uoa.di.IIGNI.IIGNIFactory* class. For more information on the IIG-NI property file, see section "gr.uoa.di.IIGNI Package".
Dispatcher property file	`conf/sga/dispatcherConfig.txt` `conf/iig/dispatcherConfig.txt`	This file contains details on the operation of the pending actions queue dispatcher. Two files are provided, to be used in the service delivery environment (SGA) and the organizational information system (IIG) respectively. Its location is provided as a parameter when the *gr.uoa.di.dispatcher.dispatcher* class is executed (SGA side) or the *gr.uoa.di.dispatcherIIG.dispatcherIIG* class is executed (IIG side). For more information on the dispatcher property file, see sections "gr.uoa.di.dispatcherIIG Package" and "gr.uoa.di.dispatcher Package".
SGA property file	`conf/sga/sgaconfig.txt`	The main configuration file for the SmartGov agent. Its location is passed as parameter to the *newSGAgent* method of the *gr.uoa.di.SGA.SGAgentFactory* class. For more information on the SGA property file, see section "gr.uoa.di.SGA Package".

Property file	Bundle filename	Comments
Adelante PAQ property file	`conf/sga/AdelantePAQConfig.txt` `conf/iig/AdelantePAQConfig.txt`	This file contains details for accessing the pending outgoing requests queue. Two files are provided, to be used in the service delivery environment (SGA) and the organizational information system (IIG) respectively. Its location is provided as a property in the SGA property file (`SGA.AdelantePAQConfFile=<configuration file spec>`). For more information on the Adelante PAQ property file, see sections "gr.uoa.di.SGA Package" and "gr.uoa.di.IIGNI Package".
IIG MYP property file	`conf/iig/iigMyPconfig.txt`	The main configuration file for the Information Interchange Gateway. Its location is passed as a command-line parameter to the execution of the *gr.uoa.di.IIGServer.IIGServer* and *gr.uoa.di.SSLIIGServer.SSLIIGServer* classes. For more information on the IIG MYP property file, see section "gr.uoa.di.IIGMyP package".
Database store configuration file	`conf/sga/DatabaseStoreConf.txt`	This file contains information regarding the software drivers used for connecting to the database when using a *store to database* method for servicing a request at the SGA.
SEP database store configuration file	`conf/iig/SEPDatabaseStoreConf.txt`	This file contains information regarding the software drivers used for connecting to the database when using a *store to database* method for servicing a request at the IIG. For more information on the SEP database store configuration file, see section "gr.uoa.di.SEPDatabaseStore Package".

5.3.2 Configuration files

Configuration file	Bundle filename	Comments
SGA-NI configuration file	`conf/sga/SGANI_Conf.xml`	An XML document containing the name and description of a method associated with each notification the SGA-NI is configured to receive. Its location is provided as a property in the SGA-NI property file (`SGA.SGAServicesConfFile=<config file spec>`). For more information on the SGA-NI configuration file, see section "gr.uoa.di.SGANI Package".
SGA Services configuration file	`conf/sga/SGAServicesConf.xml`	An XML document that binds the service names that an SGA can serve, with corresponding symbolic names for IIG and symbolic names for the communication methods to be used for the communication between SGA and IIG. Its location is provided as a property in the SGA property file (`SGA.SGAServicesConfFile=<conf file spec>`). For more information on the SGA services configuration file, see section "gr.uoa.di.SGA Package".
SGA – IIG configuration file	`conf/sga/SGAIIGConfFile.xml`	An XML document file that binds the symbolic name for the IIG with all physical level information required for initiating communication with designated IIG. Its location is provided as a property in the SGA property file (`SGA.SGAIIGConfFile=<config file spec>`). For more information on the SGA IIG configuration file, see section "gr.uoa.di.SGA Package".

Configuration file	**Bundle filename**	**Comments**
SGA Communication Methods configuration file	`conf/sga/SGACommMethConfFile.xml`	An XML document that binds the symbolic name for the communication methods with all the physical level information required for implementing each method. The file location is provided as a property in the SGA property file (`SGA.SGACommMethConfFile=<config file spec>`). For more information on the SGA communication methods configuration file, see section "gr.uoa.di.SGA Package".
IIG-NI configuration file	`conf/iig/IIGNIConf.xml`	An XML document containing the names of notifications and their associated communication information for contacting the SGA-NI. Its location is provided as a property in the IIG-NI property file (`IIG.NI.confFile=<config file spec>`). For more information on the IIG-NI configuration file, see section "gr.uoa.di.IIGNI Package".
IIG services configuration file	`conf/iig/IIGServicesConfFile.xml`	An XML document that binds the service names that an IIG can serve, with corresponding symbolic names for the communication methods and the separate external processes. Its location is provided as a property in the IIG property file (`IIGMyP.IIGServicesConfFile=<conf file spec>`). For more information on the IIG services configuration file, see section "gr.uoa.di.IIGMyP package".

Configuration file	Bundle filename	Comments
IIG communication methods configuration file	conf/iig/IIGCommMethConfFile.xml	An XML document that binds the symbolic name for the communication methods with all the physical level information required for implementing each method. The file location is provided as a property in the IIG property file (IIGMyP.IIGCommMethConfFile=<config file spec>).>). For more information on the IIG communication methods configuration file, see section "gr.uoa.di.IIGMyP package".
IIG SEP configuration file	conf/iig/IIGSEPConfFile.xml	An XML document that binds the symbolic name for the separate external processes with all the physical level information required for implementing SEP. The file location is provided as a property in the IIG property file (IIGMyP.IIGSEPConfFile=<config file spec>). For more information on the IIG SEP configuration file, see section "gr.uoa.di.IIGMyP package".
IIG security file	conf/iig/IIGSecurity.xml	A file that associates request senders with the credentials they must present to the IIG, in order to be authenticated. The file location is provided as a property in the IIG property file (IIGMyP.IIGSecurityFile=<config file spec>). For more information on the IIG security configuration file, see section "gr.uoa.di.IIGMyP package".
IIG SSL server information file	conf/iig/SSLserverInfo.xml	A file used by the secure socket layer-based version of the IIG. This file provides information on the keystore, the supported cipher suites and other parameters needed by the secure socket layer. The location of this file is provided as a command-line parameter when the class

Configuration file	Bundle filename	Comments
		gr.uoa.di.SSLIIGServer.SSLIIGServer. For more information on the IIG security configuration file, see section "gr.uoa.di.SSLIIGServer Package".

5.3.3 DTD files

The DTD files are used for validating the content structure of configuration files or messages that are received. Their contents should not be edited, however their presence is required in specific directories. These requirements are listed in the following table.

DTD file	Bundle filename	Comments
SGA-NI configuration DTD	`conf/sga/SGANI_Conf.dtd`	DTD document that validates the SGA-NI configuration file. It should be located in the same directory with the SGA-NI configuration file. The `<!DOCTYPE …` line in the SGA-NI configuration file should only reference the DTD file name, and *not* the full path specification.
SGA Services DTD	`conf/sga/SGAServicesConfFile.dtd`	DTD document that validates the SGA services configuration file It should be located in the same directory with the SGA Services configuration file. The `<!DOCTYPE …` line in the SGA services configuration file should only reference the DTD file name, and *not* the full path specification.
SGA Communication Methods DTD	`conf/sga/SGACommMethConf.dtd`	DTD document that validates the SGA Communication Methods configuration file. It should be located in the same directory with the SGA Communication Methods configuration file The `<!DOCTYPE …` line in the SGA Communication Methods configuration file should only reference the DTD file name, and *not* the full path specification.

DTD file	Bundle filename	Comments
SGA – IIG DTD	`conf/sga/SGAIIGConfFile.dtd`	DTD document that validates the SGA-IIG configuration file. It should be located in the same directory with the SGA –IIG configuration file The `<!DOCTYPE …` line in the SGA - IIG configuration file should only reference the DTD file name, and *not* the full path specification.
IIG-NI DTD	`conf/iig/IIGNIConf.dtd`	DTD document that validates the IIG-NI configuration file. It should be located in the same directory with the IIG-NI configuration file. The `<!DOCTYPE …` line in the IIG NI configuration file should only reference the DTD file name, and *not* the full path specification.
IIG Services DTD	`conf/iig/IIGServicesConfFile.dtd`	DTD document that validates the IIG services configuration file It should be located in the same directory with the IIG Services configuration file. The `<!DOCTYPE …` line in the IIG services configuration file should only reference the DTD file name, and *not* the full path specification
IIG Communication Methods DTD	`conf/iig/IIGCommMethConfFile.dtd`	DTD document that validates the IIG Communication Methods configuration file. It should be located in the same directory with the IIG Communication Methods configuration file The `<!DOCTYPE …` line in the IIG Communication Methods configuration file should only reference the DTD file name, and *not* the full path specification.
IIG SEP DTD	`conf/iig/IIGSEPConfFile.dtd`	DTD document that validates the IIG Separate External Processes (SEP) configuration file. It should be located in the same directory with the IIG SEP configuration file The `<!DOCTYPE …` line in the IIG SEP configuration file should only reference the DTD file name, and *not* the full path specification.

DTD file	Bundle filename	Comments
IIG security DTD	`conf/iig/IIGSecurity.dtd`	DTD document that validates the IIG Security configuration file. It should be located in the same directory with the IIG Security configuration file The `<!DOCTYPE …` line in the IIG Security configuration file should only reference the DTD file name, and *not* the full path specification.
IIG SSL server information DTD	`conf/iig/SSLserverInfo.dtd`	DTD document that validates the IIG SSL server information configuration file. It should be located in the same directory with the IIG SSL server information configuration file The `<!DOCTYPE …` line in the IIG SSL server information configuration file should only reference the DTD file name, and *not* the full path specification.
XML Packet DTD	`conf/iig/XMLPacket.dtd`	DTD document that validates the request messages received by the IIG. It may be located anywhere in the file system, and its location is designated by the IIGMyP.XMLPacketPath property in the IIG property file. If an application is built that creates and sends request messages to the IIG without using the SmartGov Agent library, it should arrange so that in the `<!DOCTYPE …` line in the request packets only the DTD file name is referenced, and *not* any absolute location.

5.4 Database Setup

The communication services use in several cases a database for persistent storage. A database is used in both the SGA and IIG part of the SG Agent and it should be made available to ensure its correct operation. The communication services may be used with any DBMS, provided that the appropriate JDBC driver is installed; in certain cases, adaptations need to be made to cater for DBMS particularities: for instance, in MySQL 3.x `varchar`-type fields are limited to 255 characters and the schema must be modified to use BLOB-type columns when columns of more length are required.

After installing the DBMS, the following actions should be performed:

1. a DBMS user, e.g. `smartgov`, should be created.
2. for DBMSs not supporting the "per user schema" model (e.g. MySQL) the creation of a separate database is highly recommended. Full rights to this database should be granted to the user created in step (1).
3. The tables needed by the SmartGov Agent bundle should be created in the user's schema or database. This can be accomplished by executing the appropriate SQL batch under the credentials of the user created in step (1). The distribution includes three sample SQL batches, namely `sgaora.sql`, `sgamysql.sql` and `sgamssql.sql` to be used with Oracle, MySQL and MS SQL Server, respectively. The typical procedure for creating the database objects is as follows:
    a. If Oracle is used, the command
    ```
    sqlplus username/password < sgaora.sql
    ```
    should be issued, where `username` and `password` are the credentials for the user created in step (1).
    b. If MySQL is used, the command
    ```
    mysql -uusername –ppassword dbname < sgaora.sql
    ```
    should be issued, where `username` and `password` are the credentials for the user created in step (1) and `dbname` is the name of the database created in step (2).
    c. If SQL Server is used then the *Query Analyzer* tool should be launched and the connection to the DBMS should be established by entering the user credentials in the "Connect to SQL server" dialog box. Then the `File/Open` menu should be selected and the `sgamssql.sql` should be selected. Finally the `Query/Execute` menu should be selected to complete the task.

Note that in a typical SmartGov platform installation, two separate database installations are expected to be operating, one for the service delivery environment and one for the organisational information system. Though it is possible to use one database installation for both the service delivery environment and the organisational information system, this is *not recommended*, due to security issues that may arise. If however only one database installation is available, two different users and/or databases should be created, and the configuration files should be modified accordingly to provide the appropriate connection information.

Finally, the appropriate JDBC driver should be installed in the standard extensions directory of the Java Runtime Environment – typically the `lib/ext` subdirectory under the JRE installation directory).

5.5 Package Documentation

5.5.1 gr.uoa.di.SGLogging Package

The gr.uoa.di.SGLogging package provides facilities for message logging in the SmartGov environment. Messages may be logged for various purposes, including the notification of a system operator about an event requiring imminent attention, monitoring of the activities taking place in the context of the platform, debugging activities etc. Certain events are logged automatically by SmartGov platform components (e.g. request initiation, receive of a reply), whereas applications may use these facilities to log arbitrary messages.

Applications (including SmartGov platform components) willing to exploit the logging facilities should use the provided API to fulfill this task. Additionally, within the SmartGov platform, an SGLogListener process (also provided with the implementation) should be running, which is responsible for collecting the logging requests and placing the respective messages in a persistent store.

A logging request includes (a) the message to be logged (free text) and (b) a *severity code*, indicating the importance and/or the expected reactions to the message. The SmartGov platform allows for six levels of severity, as listed in the following table.

Table 1. Severity code numbers, names and corresponding messages.

Severity code name	**Severity code number**	**Text to be displayed in the log file**	**Use**
SG_LOG_EMERG	1	EMERGENCY	An event requiring

			immediate attention
SG_LOG_ALERT	2	ALERT	An event requiring attention
SG_LOG_ERROR	3	ERROR	An error condition
SG_LOG_WARNING	4	WARNING	A warning message
SG_LOG_INFO	5	INFO	An informational message
SG_LOG_DEBUG	6	DEBUG	IT staff use for debugging purposes

When a message is placed in the persistent store, it is complemented with the following additional information:

1. The timestamp that the message was received and stored (date and time)
2. A textual description of the severity designation
3. The actual message

An example of a log message is the following:

```
03-07-2003 13:58:49 ERROR IIG-NI: Notification with name wrongNotif could not be sent to SGA-NI.
```

5.5.1.1 Using the logging facilities

In order to use the logging facilities, an application should *import* the *SGLogger*, *SGLoggerFactory* and *SGLoggerException* classes contained in the package, which implement all the necessary functionality. In more detail, the application willing to log messages should perform the following steps:

1. Use the *newSGLogger* method of the *SGLoggerFactory* class to create a new object of type *SGLogger*. Only one instance of the *SGLogger* class is allowed to exist at any given time in the context of an application.
2. Invoke the *logMessage* method of the *SGLogger* object, in order to actually log the message. The *logMessage* method accepts two parameters, corresponding to the severity designation and the actual message, for example:

   ```
   SGL.logMessage(SGLogger.SG_LOG_EMERG, "Message to be logged");
   ```

While performing any of the steps listed above, certain error conditions may arise; in these cases, an exception of type *SGLoggerException* is thrown, which can be caught and appropriately handled by the application.

Since the SmartGov logging facility follows the client-server model, it is important for the applications to be able to locate the SGLogListener entity, which intercepts all logging requests and arranges for their persistent storage. The information necessary to locate the SGLogListener entity should be provided in a property file,

and the location of the file is passed as a parameter to the newSGLogger method. This property file contains two lines, designating the communication details with the SGLogListener. The one is the name of the host where the SGLogListener is located and the other the port at which the SGLogListener is listening for logging requests. The property file must have the following form:

```
#SGLogger property file
SGA.Logger.port=<port>
SGA.Logger.host=<host>
```

For example:

```
#SGLogger property file
SGA.Logger.port=30000
SGA.Logger.host=hydra.mm.di.uoa.gr
```

5.5.1.2 Example

A sample application using the `gr.uoa.di.SGLogging` package is listed below:

```
public class testLogger {
    /** Creates a new instance of testLogger */
    public testLogger() { }
     public static void main(String[] args) {
          if (args.length != 1 ) {
             System.out.println("testLoggerUsage: testLogger  <propertyFile> ");
           }
           else{
                try{
               /** Creates an instance of the SGLogger, giving as argument the
                    *the property file.*/
                    SGLogger SGL = SGLoggerFactory.newSGLogger(args[0]);
                      /*Log the event*/
                    SGL.logMessage(SGL.SG_LOG_EMERG, "Message to be logged");
                    System.err.println("Exited OK.");
                }
                catch(SGLoggerException SGLEx){
                    System.err.println("Cannot create logger");
                    System.exit(1);
                }
        }
    }
}
```

In order to facilitate the usage of the *SGLogger*, a static *logMessage* method is provided in the `gr.uoa.di.SGUtil.SGUtil` class. The *logMessage* method accepts two parameters, corresponding to the severity designation and the actual message, for example:

```
SGUtil.logMessage(SGLogger.SG_LOG_EMERG, "Message to be logged");
```

The `SGUtil.logMessage` method arranges for the creation of the appropriate *SGLogger* instance (if necessary) and then uses this instance to log the messages.

Both the `gr.uoa.di.SGUtil` and `gr.uoa.di.SGLogger` packages should be imported for the call to this method to work.

The `SGUtil.logMessage` method needs to have information regarding the connection details of the SGLogListener process. This information should be provided by means of a property file (as is the case with the SGLogger package) and the system property `SGLogger.propertyFile` should be set to point to the location of this file. If the property is not set, the `SGUtil.logMessage` function ignores calls for logging.

The `SGLogger.propertyFile` system property may be set programmatically from within a Java program using the `System.setProperty` method:

```
System.setProperty("SGLogger.propertyFile", "/path/to/property/file");
```

or by using the `-D` flag to the Java Virtual Machine when the Java program is being run:

```
java -DSGLogger.propertyFile=/path/to/property/file myprog.java
```

5.5.2 gr.uoa.di.SGLogListener Package

The `gr.uoa.di.SGLogListener` package implements the process that is responsible for collecting the logging requests and placing the messages to be logged in a persistent store. Logging requests originate from the SGLogger, whose functionality is described in another section of this document.

The SGLogListener is started by executing the class SGLogListenerFactory, providing as parameter the location of the SGLogListener property file.

```
java gr.uoa.di.SGLogListener.SGLogListenerFactory <property file>
```

For example,

```
java gr.uoa.di.SGLogListener.SGLogListenerFactory c:\SG\SGLListPrF.txt
```

The SGLogListener then waits for logging requests to arrive (to a specified port). These requests are received in the form of text messages, which the SGLogListener writes to a file.

The path to the file that the SGLogListener will write the log messages to, and the port that will be monitored for incoming logging requests should be specified in a property file, and the location of the file is provided as a parameter when the SGLogListenerFactory is executed. More specifically, the property file contains three lines, designating:

1. the TCP/IP port where the SGLogListener is listening for logging requests (`SGA.LogListener.port` property)
2. the backlog of the listener server socket, which is the maximum queue length for incoming connection requests (`SGA.LogListener.backlog` property)
3. the file were the messages will be stored (`SGA.LogListener.destinationFile` property)

The format of the property file is as follows:

```
#SGLogListener property file
SGA.LogListener.port=<port>
SGA.LogListener.backlog=<backlog>
SGA.LogListener.destinationFile=<destination specification >
```

For example:

```
#SGLogListener property file
SGA.LogListener.port=30000
SGA.LogListener.backlog=50
SGA.LogListener.destinationFile=c:\\smartgov\\logs\\LogDest.txt
```

5.5.3 gr.uoa.di.SGANI Package

The `gr.uoa.di.SGANI` package provides facilities for intercepting and storing notifications in the SmartGov SGA-EPAQ (incoming queue for the service delivery platform). The *SGA Notifications Interceptor (SGA-NI)* is an autonomous program that continuously runs on the SmartGov service delivery platform and listens for notifications signifying that an external to the platform event has taken place. The SGA-NI responds to these notifications by placing a suitable entry in the SGA-EPAQ, which will be handled by the SGA-PAQUED. The notifications are sent to SGA-NI by the IIG Notification Initiator (IIG-NI).

SGA-NI performs the following actions:

1. Listen to a specified port for incoming notifications
2. Upon receiving a notification, look up in the SGA-NI XML configuration file the information concerning the method associated with the notification.
3. Store the method information in the SGAEPAQ.

The SGA-NI is started by executing the class SGANIFactory, providing as parameter the location of the SGA-NI property file, described in the following paragraphs.

```
java gr.uoa.di.SGANI.SGANIFactory <property file>
```

For example,

```
java gr.uoa.di.SGANI.SGANIFactory c:\smartgov\cfg\\SGLANIPropertyFile.txt
```

The SGA-NI creates a server socket, and listens to a port, also specified in the property file, for incoming notifications. These notifications are received in the form of text messages. Upon receiving a notification, the SGA-NI looks it up in a configuration file. This configuration file contains the information about the methods associated with each notification and its location (file specification including directory and file name) is defined in the property file.

The method information extracted from the configuration file is stored in the SGA-EPAQ. The SGA-EPAQ is implemented as a database table. In order to facilitate the storage and retrieval of data in the database, the `gr.uoa.di.EntraPAQ` package is used, which is described in another part of this document. The information for connecting to the database (connection credentials, network information etc) is provided by means of a separate property file; the location of this property file is specified as a property within the SGA-NI property file. The contents of the SGA-EPAQ property file are explained below.

SGA-NI uses the SGLogger, which is described in another part of this document, to log information on the following events:

1. receiving a notification
2. failure to store the notification in the database, due to communication failure with the IIG-Notification Initiator
3. failure while retrieving the method information from the XML configuration file
4. database connection failure.

Since SGA-NI uses SGLogger facilities, it needs to have access to a property file with the information necessary for connecting to the SGLogListener. The location of this property file is specified as a property within the SGA-NI property file.

5.5.3.1 The SGANI configuration file

The SGA-NI configuration file is an XML document which contains the description in XML format of the methods associated with a notification.

This file contains the following information for each notification:

1. The name of the notification
2. The method description, which contains the following items
    a. The command path, i.e. the complete file specification of the OS-level command that will be executed as a response to the reception of the notification
    b. The working directory, i.e. the OS directory that will be set as "current" before the execution of the command is started

c. The command parameters, i.e. command-line arguments that will be passed to the command.
d. The input file, i.e. a file containing data that will be read by the command as input
e. The output file, i.e. a file into which the command's output will be stored.
f. The error file, i.e. a file into which error messages emitted by the command (if any) will be stored.
g. The environmental variable list, which consists of pairs of variable names and values. This item is optional. We note here that for Java programs, in particular, environment variables may be suppressed by the Java runtime environment, depending on the JRE version, since Java designers have characterised the environment variable mechanism as "non-portable". For communicating parameters to Java programs, the property file approach is recommended.

SGA-NI configuration files are validated against a DTD document, which must be located in the same directory with the SGA-NI configuration file. If an installation uses multiple SGA-NI configuration files, which are stored in different directories, then a copy of the DTD document should be placed in each of these directories. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD XML Schema describing the SGA-NI XML configuration file -->
<!ELEMENT root (notificationInfo*)>
<!ELEMENT notificationInfo (notificationName, method)>
<!ELEMENT notificationName (#PCDATA)>
<!ELEMENT method (commandPath, workingDirectory, parameters, inputFile, outputFile, errorFile, envVariable*)>
<!ELEMENT commandPath (#PCDATA)>
<!ELEMENT workingDirectory (#PCDATA)>
<!ELEMENT parameters (#PCDATA)>
<!ELEMENT inputFile (#PCDATA)>
<!ELEMENT outputFile (#PCDATA)>
<!ELEMENT errorFile (#PCDATA)>
<!ELEMENT envVariable (envVariableName, envVariableValue)>
<!ELEMENT envVariableName (#PCDATA)>
<!ELEMENT envVariableValue (#PCDATA)>
```

An SGA-NI configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an SGA-NI XML configuration file is illustrated bellow.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root SYSTEM "SGANI_Conf.dtd">
<root>
    <notificationInfo>
        <notificationName>dataReady</notificationName>
        <method>
            <commandPath>java testProgram</commandPath>
            <workingDirectory>Z:\\smartgov\\wp\\wp06\\api\\sga</workingDirectory>
            <parameters/>
            <inputFile>c:\\smartgov\\test\\testPinput.txt</inputFile>
            <outputFile>Z:\\smartgov\\test\\testPoutput.txt</outputFile>
            <errorFile>Z:\\smartgov\\test\\testPerror.txt</errorFile>
        </method>
    </notificationInfo>
    <notificationInfo>
        <notificationName>dataLost</notificationName>
        <method>
            <commandPath>java testProgram2</commandPath>
            <workingDirectory>c:\\smartgov\\test </workingDirectory>
            <parameters/>
            <inputFile>c:\\smartgov\\test\\testPAinput.txt</inputFile>
            <outputFile>c:\\smartgov\\test\\testPAoutput.txt</outputFile>
            <errorFile>c:\\smartgov\\test\\testPAerror.txt </errorFile>
        </method>
    </notificationInfo>
</root>
```

5.5.3.2 The SGANI property file

The SGA-NI property file is provided as a parameter when the SGANIFactory is executed and contains the information necessary for SGA-NI to operate. This information is stored in five properties, which are the following:

- The port where the SGA-NI listens for incoming notifications.
- The backlog of the SGA-NI server socket, which is the maximum queue length for incoming connection requests.
- The file name of the SGA-NI configuration file (described above). A full file specification may be provided, including the directory and the filename of the configuration file.
- The property file of the SGA-EPAQ, which contains information for the database connection.
- The property file of the SGLogger, providing information on contacting the logging facilities for event information logging

The property file must have the following form:

```
SGA.NI.port=<port>
SGA.NI.backlog=<backlog>
SGA.NI.confFile=<configuration file specification >
SGA.EntraPAQ.propertyFile=<property file specification >
SGLogger.propertyFile=<property file specification >
```

For example:

```
# Property file for the SGA NI
SGA.NI.port=30000
SGA.NI.backlog=50
SGA.NI.confFile=c:\\smartgov\\cfg\\SGANI\\SGANI_Conf.xml
SGA.EntraPAQ.propertyFile=c:\\smartgov\\cfg\\entraPAQ\\EntraPAQConfig.txt
SGLogger.propertyFile=Z:\\smartgov\\cfg\\SGLogging\\SGLogConfigSGA.txt
```

5.5.3.3 The Entra PAQ property file

The EntraPAQ (SGA-EPAQ) property file is provided as a property in the SGA-NI property file and contains the necessary information for connecting with the database where the Entra PAQ is stored. It contains the following four properties:

- The user name for connecting with the database where the Entra PAQ is stored.
- The name of the database where the Entra PAQ is stored.
- The password for connecting with the database where the Entra PAQ is stored.
- The driver for connecting with the database where the Entra PAQ is stored.
- The connection string for connecting with the database where the Entra PAQ is stored.

The property file must have the following form:

```
SGA.EntraPAQ.username=<username>
SGA.EntraPAQ.database=<database name>
SGA.EntraPAQ.password=<password>
SGA.EntraPAQ.driver=<driver class name>
SGA.EntraPAQ.connectString=<connection string>
```

For example

```
# Property file for the SGA EntraPAQ
SGA.EntraPAQ.username=smartgov
SGA.EntraPAQ.database=smartgov
SGA.EntraPAQ.password=sg123
SGA.EntraPAQ.driver=oracle.jdbc.driver.OracleDriver
SGA.EntraPAQ.connectString=jdbc:oracle:oci8:@
```

5.5.3.4 Extending the SGA-NI

Currently, the SGA-NI uses the TCP/IP communication protocol in order to receive notifications from the IIG-NI. It creates a ServerSocket and listens to a port for incoming notifications. If the IIG-NI uses another communication protocol, the SGANI should be modified appropriately in order to be able to receive the notifications. In order to implement such an extension, the programming team should:

1. replace the TCP/IP-oriented properties in the SGA-NI configuration file (`SGA.NI.port` and `SGA.NI.backlog`) with properties appropriate for the protocol that will be supported.
2. Modify the code that looks up the specific properties in the property file so that it accesses the properties specified in step (1).
3. load the libraries that support the selected protocol and arrange so that these libraries are used instead of the TCP/IP libraries.

5.5.4 gr.uoa.di.IIGNI Package

The `gr.uoa.di.IIG-NI` package provides facilities for sending notifications from the IIG to the SmartGov SGA-NI. Notifications may be sent for various purposes, including the completion of a back-end batch process, the request for a specific action to be taken at the side of the service delivery environment etc.

Applications (including SmartGov platform components) willing to exploit the notification facilities should use the provided API to fulfill this task. Additionally, within the SmartGov platform, an SGA Notification Interceptor (SGA-NI) process (provided with the implementation and described in the previous paragraphs) should be running, which is responsible for receiving the notifications and placing them in a persistent store, the SGA EPAQ.

5.5.4.1 Using the IIG Notification Initiator

In order to use the IIG Notification Initiator, an application should *import* the *IIGNI*, *IGNIFactory* and *IIGNIException* classes contained in the package, which

implement all the necessary functionality. In more detail, the application willing to send notifications should perform the following steps:

1. Use the *newIIGNI* method of the *IIGNIFactory* class to create a new object of type *IIGNI*. Only one instance of the *IIGNI* class is allowed to exist at any given time in the context of an application.
2. Invoke the *IIGToSGAgentNotification* method of the *IIGNI* object, in order to actually send the notification. The *IIGToSGAgentNotification* method accepts one parameter, the name of the notification to be sent, for example:

```
NI.IIGToSGAgentNotification("dataReady");
```

While performing any of the steps listed above, certain error conditions may arise; in these cases, an exception of type *IIGNIException* is thrown, which can be caught and appropriately handled by the application. Such an exception is thrown when:

1. the notification name does not correspond to a valid notification contained in the IIGNI XML configuration file or
2. if there were problems with the communication with SGA-NI.

In the latter case, the notification is stored in the IIG AdelantePAQ (IIG-APAQ: outgoing queue for the organizational or third-party information system) and the IIG dispatcher will attempt to resend this notification.

All the events concerning the notification status are logged using the SGLogger, described in an earlier paragraph. The SGLogger logs the event of a notification being sent and the errors that may arise during this process. Since IIG-NI uses SGLogger facilities, it needs to have access to a property file with the information necessary for connecting to the SGLogListener. The location of this property file is specified as a property within the IIG-NI property file. If the SGLogger property file is not provided, or communication with the logger is not possible, the IIG-NI will operate successfully but events will not be logged.

Since the IIG-NI sends the notifications to the SGA-NI, it is important for the applications to be able to locate the SGA-NI entity, which will receive the notifications and will arrange for the persistent storage of their corresponding methods. The information necessary to locate the SGA-NI entity that notifications will be sent to should be provided in a configuration file, whose location is specified within the IIG-NI property file. Furthermore, the locations of the IIG AdelantePAQ and SGLogger property files are provided within the IIG-NI property file.

The location of the IIG-NI property file is passed as a parameter to the newIIGNI method. The IIGNI property file contains the following three lines:

- The file name of the IIGNI configuration file.
- The property file of the IIG-APAQ.
- The property file of the SGLogger.

The property file must have the following format:

```
IIG.NI.confFile=<configuration file specification>
IIG.AdelantePAQ.propertyFile=<property file specification>
SGLogger.propertyFile=<property file specification>
```

The property file may also contain comment lines, beginning with the hash (#) symbol. For example:

```
# Property file for the SG IIGNI
IIG.NI.confFile=c:\\smartgov\\cfg\\IIGNI\\IIGNIConf.xml
IIG.AdelantePAQ.propertyFile=c:\\smartgov\\cfg\\adelantePAQIIG\\APAQIIGConfig.txt
SGLogger.propertyFile=c:\\smartgov\\cfg\\SGLogging\\SGLogConfigIIG.txt
```

5.5.4.2 The IIG-NI configuration file

The IIGNI configuration file is an XML document which contains the communication information associated with a notification in XML format.

This file contains the following information for each notification:

1. The name of the notification
2. The communication information for contacting the SGA-NI listener. In the case of the TCP/IP protocol, this information consists of the host name that the SGA-NI listener is run on and port that the SGA-NI listener listens to.

IIG-NI configuration files are validated against a DTD document, which must be located in the same directory with the IIG-NI configuration file. If an installation uses multiple IIG-NI configuration files, which are stored in different directories, then a copy of the DTD document should be placed in each of these directories. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--The DTD of the IIG-NI configuration file-->
<!ELEMENT root (notificationInfo*)>
<!ELEMENT notificationInfo (notificationName, comType, comData)>
<!ELEMENT notificationName (#PCDATA)>
<!ELEMENT comType (#PCDATA)>
<!ELEMENT comData (tcpIpCom)>
<!ELEMENT tcpIpCom (address, port)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT port (#PCDATA)>
```

An IIG-NI configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an IIG-NI XML configuration file is illustrated bellow.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root SYSTEM "IIGNIConf.dtd">
<root>
    <notificationInfo>
        <notificationName>dataReady</notificationName>
        <comType>tcpIp</comType>
        <comData>
            <tcpIpCom>
                <address>hydra.mm.di.uoa.gr</address>
                <port>30000</port>
            </tcpIpCom>
        </comData>
    </notificationInfo>
    <notificationInfo>
        <notificationName>dataLost</notificationName>
        <comType>tcpIp</comType>
        <comData>
            <tcpIpCom>
                <address>hydra.mm.di.uoa.gr</address>
                <port>30000</port>
            </tcpIpCom>
        </comData>
    </notificationInfo>
    <notificationInfo>
</root>
```

5.5.4.3 Extending the IIG-NI

Currently the IIGNI uses the TCP/IP communication protocol in order to send notifications to the SGA-NI. All the implementation details for the actual message transfer are encapsulated into the class IIGNITcpIpHandler, contained in the `gr.uoa.di.IIGNI` package. This class may be modified (or copied, renamed and modified) in order to use other communication protocols. The modifications should be the following:

1. The communication information should be stored in an appropriate XML file, of the general format described in the following section.
2. The class IIGNITcpIpHandler should be modified accordingly:
   a. The following class attributes should be modified to reflect the new communication information.

```
/**The communication type. */
static final String comType = "communicationType";

/**This array contains the names of the elements containing the communication information. */
static final String[] comElements ={"property1", "property2", "property3"};

/**This variable contains the number of comInf elements, in this case 2*/
static final int comInfNum=<number of properties>;
```

For example, for the TCP/IP:

```
/**The communication type. In this case it is a constant of type String with value tcpIp*/
static final String comType = "tcpIp";

/**This array contains the names of the elements containing the communication information. */
static final String[] comElements ={"address", "port"} ;

/**This variable contains the number of comInf elements, in this case 2*/
static final int comInfNum=2;
```

b. The sendMessage() method should be modified accordingly in order to be able to send messages with the specified protocol. The libraries implementing the target protocol should be also imported.
c. The DTD corresponding to the XML configuration file should be modified accordingly, to describe the elements required for the new protocol.

5.5.4.4 General format of the IIG-NI configuration file

The IIG-NI XML configuration file contains the following information for each notification:

1. The name of the notification
2. The communication information for the SGANI listener

The general format of the DTD for this document is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--The DTD of the IIG-NI configuration file-->
<!ELEMENT root (notificationInfo*)>
<!ELEMENT notificationInfo (notificationName, comType, comData)>
<!ELEMENT notificationName (#PCDATA)>
<!ELEMENT comType (#PCDATA)>
<!ELEMENT comData (tcpIpCom)>
<!ELEMENT newCom (property1, property2)>
<!ELEMENT property1 (#PCDATA)>
<!ELEMENT property2 (#PCDATA)>
```

The IIGNI XML configuration file should contain the name of the DTD. The DTD should be in the same directory with the document. An example of an IIGNI XML configuration file is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root SYSTEM "IIGNIConf.dtd">
<root>
    <notificationInfo>
        <notificationName>notification1</notificationName>
        <comType>communicationType</comType>
        <comData>
            <newCom>
                <property1>property1 value</property1>
                <property2>property2 value</property2>
            </newCom>
        </comData>
    </notificationInfo>
    <notificationInfo>
        <notificationName>dataLost</notificationName>
        <comType>communicationType</comType>
        <comData>
            <newCom>
                <property1>property1 value</property1>
                <property2>property2 value</property2>
            </newCom>
      </notificationInfo>
</root>
```

5.5.4.5 The SGA Adelante PAQ property file

The Adelante PAQ property (SGA-APAQ) file is provided as a property in the IIG-NI property file and contains the necessary information for connecting with the database where the Adelante PAQ is stored. It contains the following five properties:

- The user name for connecting with the database where the Adelante PAQ is stored.
- The name of the database where the Adelante PAQ is stored.
- The password for connecting with the database where the Adelante PAQ is stored.
- The driver for connecting with the database where the Adelante PAQ is stored.
- The connection string for connecting with the database where the Adelante PAQ is stored.

The property file must have the following form:

```
IIG.AdelantePAQ.username=<username>
IIG.AdelantePAQ.database=<database name>
IIG.AdelantePAQ.password=<password>
IIG.AdelantePAQ.driver=<driver class name>
IIG.AdelantePAQ.connectString=<connection string>
```

For example,

```
# Property file for the SGA EntraPAQ
IIG.AdelantePAQ.username=smartgov
IIG.AdelantePAQ.database=smartgov
IIG.AdelantePAQ.password=sg123!
IIG.AdelantePAQ.driver=oracle.jdbc.driver.OracleDriver
IIG.AdelantePAQ.connectString=jdbc:oracle:oci8:@
```

5.5.5 gr.uoa.di.dispatcherIIG Package

The `gr.uoa.di.dispatcherIIG` package provides facilities for using the *IIG Pending Actions Queue Dispatcher (IIG-PAQUED).* The IIG-PAQUED is an autonomous program that periodically scrutinises the IIG Entra and Adelante pending actions queues on the SmartGov service delivery platform, extracts actions that can be carried out, and initiates their execution.

The IIG-PAQUED is started by executing the class `gr.uoa.di.dispatcherIIG.dispatcherIIG`. A property file should be provided as a parameter for the execution of the IIG-PAQUED; the contents of the property file are described in the following paragraphs. The IIG-PAQUED periodically

retrieves all records from the Entra (incoming) and Adelante (outgoing) PAQ and processes them, accordingly, executing the method described in each PAQ entry.
The periodicity of the scheduler scrutinising the PAQ will be determined by the IT staff supporting the actual runtime environment, taking into account any peculiarities and constraints placed by the actual working systems. The period of invocation may be variant, ranging from minutes to days, depending upon the processing requirements of each message class. For example, messages related to warehouse stock updating maybe processed in 10 minutes intervals, while messages related to certificate applications could be processed in a daily basis. The time interval during which the IIG-PAQUED remains idle between two successive inspections of the PAQ is specified by a property in the dispatcher property file; this property is named *IIG.dispatcher.sleepTime*. If no such property is specified, the default sleep interval is used, as set in the static variable DEFAULT_SLEEP_TIME = 2520000 (42 minutes, expressed in milliseconds).
While processing the PAQ entries, certain error conditions may arise; in these cases, an exception of type *dispatcheIIGrException* is thrown.
All the events concerning the processing status are logged using the SGLogger, described in a previous part of this document. The SGLogger logs the following events:

1. the beginning of the processing
2. the errors that may arise during the execution.

Since the dispatcher uses logging facilities, it needs to have access to a property file into which the details of the communication with the SGLogListener process are specified. A special property within the dispatcher property file designates the location of the dispatcher property file. If the SGLogger property file is not provided, the dispatcher will operate normally, but events will not be logged.

5.5.5.1 The dispatcher property file

The dispatcher property file contains the following properties:

- the location of the IIG EntraPAQ property file
- the location of the IIG AdelantePAQ property file
- The configuration file of the IIG-NI.
- The property file of the SGLogger.
- The dispatcher sleep time interval in milliseconds

The property file must have the following form:

```
# Property file for the IIG dispatcher
IIG.EntraPAQ.propertyFile=<EntraPAQ property file>
IIG.AdelantePAQ.propertyFile=<AdelantePAQ property file>
IIG.NI.confFile=<configuration file name>
SGLogger.propertyFile=<property file name>
IIG.dispatcher.sleepTime=<time in milliseconds>
```

For example:

```
# Property file for the IIG dispatcher
IIG.EntraPAQ.propertyFile=c:\\smartgov\\cfg\\iig\EntraPAQConfig.txt
IIG.AdelantePAQ.propertyFile= c:\\smartgov\\cfg\\iig\adelantePAQConfig.txt
IIG.NI.confFile=c:\\smartgov\\cfg\\IIGNI\\IIGNIConf.xml
SGLogger.propertyFile=c:\\smartgov\\cfg\\SGLogging\\SGLogConfigSGA.txt
IIG.dispatcher.sleepTime=2520000
```

5.5.6 gr.uoa.di.dispatcher Package

The `gr.uoa.di.dispatcher` package provides facilities for using the *SGA Pending Actions Queue Dispatcher (SGA-PAQUED).* The SGA-PAQUED is an autonomous program that periodically scrutinises the SGA Entra and Adelante pending actions queues on the SmartGov service delivery platform, extracts actions that can be carried out, and initiates their execution.

The SGA-PAQUED is started by executing the class `gr.uoa.di.dispatcher.dispatcher`. A property file should be provided as a parameter for the execution of the SGA-PAQUED; the contents of the property file are described in the following paragraphs. The SGA-PAQUED periodically retrieves all records from the Entra (incoming) and Adelante (outgoing) PAQ and processes them, accordingly, executing the method described in each PAQ entry.

The periodicity of the scheduler scrutinising the PAQ will be determined by the IT staff supporting the actual runtime environment, taking into account any peculiarities and constraints placed by the specific working systems. The period of invocation may be variant, ranging from minutes to days, depending upon the processing requirements of each message class. For example, messages related to warehouse stock updating maybe processed in 10 minutes intervals, while messages related to certificate applications could be processed in a daily basis. The time interval during which the SGA-PAQUED remains idle between two successive inspections of the PAQ is specified by a property in the dispatcher property file; this property is named *SGA.dispatcher.sleepTime*. If no such property is specified, the default sleep interval is used, as set in the static variable DEFAULT_SLEEP_TIME = 2520000 (42 minutes, expressed in milliseconds).

While processing the PAQ entries, certain error conditions may arise; in these cases, an exception of type *dispatcherException* is thrown.
All the events concerning the processing status are logged using the SGLogger, described in a previous part of this document. The SGLogger logs the following events:

1. the beginning of the processing
2. the errors that may arise during the execution.

Since the dispatcher uses logging facilities, it needs to have access to a property file into which the details of the communication with the SGLogListener process are specified. A special property within the dispatcher property file designates the location of the dispatcher property file. If the SGLogger property file is not provided, the dispatcher will operate normally, but events will not be logged.
The dispatcher property file contains five properties, which are the following:

- the location of the SGA EntraPAQ property file
- the location of the SGA AdelantePAQ property file
- The configuration file of the SGA.
- The property file of the SGLogger.
- The dispatcher sleep time interval in milliseconds

The property file must have the following form:

```
# Property file for the SG dispatcher
SGA.EntraPAQ.propertyFile=<EntraPAQ property file>
SGA.AdelantePAQ.propertyFile=<AdelantePAQ property file>
SGA.SGAConfFile=<configuration file name>
SGLogger.propertyFile=<property file name>
SGA.dispatcher.sleepTime=<time in milliseconds>
```

For example:

```
# Property file for the SG dispatcher
SGA.EntraPAQ.propertyFile=c:\\smartgov\\cfg\\iig\EntraPAQConfig.txt
SGA.AdelantePAQ.propertyFile= c:\\smartgov\\cfg\\iig\adelantePAQConfig.txt
SGA.SGAConfFile=c:\\smartgov\\cfg\\sga\\SGAConfFile.txt
SGLogger.propertyFile=c:\\smartgov\\cfg\\SGLogging\\SGLogConfigSGA.txt
SGA.dispatcher.sleepTime=2520000
```

5.5.7 gr.uoa.di.SGA Package

The `gr.uoa.di.SGA` package provides facilities for using the *SG Agent (SGA),* a class library containing the methods that allow SmartGov applications to submit requests and retrieve results.
Applications developed within the SmartGov Framework (SGoVApps) delegate all communications with external IT systems to the SmartGov Agent (SGA). The SGA

communicates with the Information Interchange Gateway (IIG) and returns results to the calling application. A generic communication event is an event that spans the SmartGov Platform and reaches a 3rd party system. Initiation of communication may be initiated from the SGoVApp (SmartGov Application) or from the IT system and each receiving party has the responsibility of checking all necessary conditions that must hold for the event to complete.

A SGoVApp initiates communication sending requests to an SGA using the following SGA method:

```
public String SGAppToSGAgentRequest(long requestId, String
        serviceName, String XMLMessage, boolean realTime, boolean
        persistent);
```

The method parameters and their associated semantics are presented in the following table.

`requestId`	A unique request identifier that serves to characterize this request
`serviceName`	A symbolic service name that the message refers to. The receiving SGA is expected to forward the encapsulated `XMLMessage` to the named service
`XMLMessage`	A message that contains all information that the named `serviceName` requires. The SGA does not interpret this message, rather it is passed as is to the next step
`realTime`	Indicates whether the communication event is happening in real-time and consequently an immediate response is expected. When this flag is set, the SGA does not closes the communication channel with the SGoVApp but it immediately forwards the message to the appropriate IIG and returns the result to the calling SGoVApp
`persistent`	Indicates whether the message should persist in case of communication errors or other abruptions and retransmitted later. If this flag is set, message is stored in the SGA Adelante Pending Actions Queue.

In more detail, the application willing to send a request to the SGA should perform the following steps:

1. Use the *newSGAgent* method of the *SGAgentFactory* class to create a new object of type *SGAgent*. A property file should be provided as a parameter to the *newSGAgent* method; the contents of the property file are described in the following paragraphs.
2. Invoke the `SGAppToSGAgentRequest` method of the *SGAgent* object, in order to actually make the request. The *logMessage* method accepts the parameters mentioned previously, for example:

```
result = myAgent.SGAppToSGAgentRequest(15,
"getPersonalInfo", "<?xml version="1.0" encoding="utf-8"
?><name>George Georgiou</name> <name>Petros Petriou</name>
<name>Eleni Hatzimixail</name> <name>Costas Tses</name>",
SGA_NONREALTIME, SGA_PERSISTENT);
```

5.5.7.1 The SGA property file

The SG Agent property file contains six properties, which are the following:

- `SGA.SGAServicesConfFile`: The path for the XML file that binds the service names that SGA can serve, with corresponding symbolic names for IIGs and symbolic names for the communication methods to be used for the communication between SGA and IIG.

  The SGAServices configuration file is validated against a DTD document, which must be located in the same directory with the SGAServices configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD XML Schema describing the SGAServices configuration file)-->
<!ELEMENT SGAServices (service*)>
<!ELEMENT service (serviceName, IIGName, methodName+)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT IIGName (#PCDATA)>
<!ELEMENT methodName (#PCDATA)>
```

  An SGAServices configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an SGAServices XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE SGAServices SYSTEM "SGAServicesConfFile.dtd">
<SGAServices>
    <service>
        <serviceName>getPersonalInfo</serviceName>
        <IIGName>taxationIIG</IIGName>
        <methodName>SSLtcpIp1</methodName>
        <methodName>tcpIp2</methodName>
        <methodName>localFileStore2</methodName>
    </service>
    <service>
        <serviceName>getPersonalInfo</serviceName>
        <IIGName>communityIIG</IIGName>
        <methodName>tcpIp1</methodName>
        <methodName>localDataStore1</methodName>
        <methodName>SSLtcpIp1</methodName>
    </service>
    <service>
        <serviceName>getPersonalInfo2</serviceName>
        <IIGName>vatIIG</IIGName>
        <methodName>tcpIp3</methodName>
        <methodName>localDataStore3</methodName>
        <methodName>localFileStore3</methodName>
    </service>
    <service>
        <serviceName>getTaxPay</serviceName>
        <IIGName>taxationIIG</IIGName>
        <methodName>tcpIp1</methodName>
        <methodName>localDataStore1</methodName>
        <methodName>localFileStore1</methodName>
    </service>
    <service>
        <serviceName>getContact</serviceName>
        <IIGName>communityIIG</IIGName>
        <methodName>tcpIp1</methodName>
    </service>
</SGAServices>
```

- `SGA.SGACommMethConfFile`: The path for the XML file that binds the symbolic names for the communication methods with all the physical level information required for implementing the designated method.

The `SGACommunicationMethods` configuration file is validated against a DTD document, which must be located in the same directory with the SGACommunicationMethods configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD XML Schema describing the SGACommunicationMethods configuration file)-->
<!ELEMENT commMethods (method*)>
<!ELEMENT method (methodName, (tcpIpMethod | SSLtcpIpMethod | localDataStore | localFileStore))>
<!ELEMENT methodName (#PCDATA)>
<!ELEMENT tcpIpMethod (password, passwordEnc)>
<!ELEMENT SSLtcpIpMethod (keystorePath, keystorePassword, keyPassword, keystoreType, KeyManagerAlgorithm, SSLVersion, supportedSuites+)>
<!ELEMENT localDataStore (connectionStr)>
<!ELEMENT localFileStore (fileName)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT passwordEnc (#PCDATA)>
<!ELEMENT keystorePath (#PCDATA)>
<!ELEMENT keystorePassword (#PCDATA)>
<!ELEMENT keyPassword (#PCDATA)>
<!ELEMENT keystoreType (#PCDATA)>
<!ELEMENT KeyManagerAlgorithm (#PCDATA)>
<!ELEMENT SSLVersion (#PCDATA)>
<!ELEMENT supportedSuites (#PCDATA)>
<!ELEMENT connectionStr (#PCDATA)>
<!ELEMENT fileName (#PCDATA)>
```

An `SGACommunicationMethods` configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an `SGACommunicationMethods` XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE commMethods SYSTEM "SGACommMethConfFile.dtd">
<commMethods>
   <method>
      <methodName>tcpIp1</methodName>
      <tcpIpMethod>
         <password>test1</password>
         <passwordEnc>plaintext1</passwordEnc>
      </tcpIpMethod>
   </method>
   <method>
      <methodName>SSLtcpIp1</methodName>
      <SSLtcpIpMethod>
         <keystorePath>D:\j2sdk1.4.1_01\bin\trustcerts</keystorePath>
         <keystorePassword>userpass</keystorePassword>
         <keyPassword>userpass</keyPassword>
         <keystoreType>JKS</keystoreType>
         <KeyManagerAlgorithm>SunX509</KeyManagerAlgorithm>
         <SSLVersion>SSLv3</SSLVersion>
         <supportedSuites>SSL_RSA_WITH_RC4_128_MD5</supportedSuites>
         <supportedSuites>SSL_DH_anon_WITH_RC4_128_MD5</supportedSuites>
         <supportedSuites>SSL_DH_anon_WITH_3DES_EDE_CBC_SHA</supportedSuites>
         <supportedSuites>SSL_DH_anon_WITH_DES_CBC_SHA</supportedSuites>
         <supportedSuites>SSL_DH_anon_EXPORT_WITH_RC4_40_MD5</supportedSuites>
         <supportedSuites>SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA</supportedSuites>
      </SSLtcpIpMethod>
   </method>
   <method>
      <methodName>localDataStore1</methodName>
      <localDataStore>
         <connectionStr>smartgov:smartgov:sg123</connectionStr>
      </localDataStore>
   </method>
   <method>
      <methodName>localFileStore1</methodName>
      <localFileStore>
         <fileName>c:\smartgov\filestores\sga\file1.txt</fileName>
      </localFileStore>
   </method>
   <method>
      <methodName>localFileStore2</methodName>
      <localFileStore>
         <fileName> c:\smartgov\filestores\sga\file2.txt</fileName>
      </localFileStore>
   </method>
</commMethods>
```

There are 4 types of supported communication methods between SGA and IIG, which are presented below:

1. **localFileStore method**, where SGA stores the XML Message in a file. The location and the name of the file are specified in the SGACommunicationMethods XML configuration file with the following declaration:

```
<localFileStore>
   <fileName>c:\smartgov\filestores\file.txt</fileName>
</localFileStore>
```

The file path must be valid for the system that the SGA is run on; on Unix systems file specification components are separated by forward slashes, whereas on Windows based systems the separator character is the backslash, and drive letters should be included or UNC filenames can be used.

2. **localDataStore method**, where SGA stores the XML Message to a database. The information for connecting to the database (database name, username and password) are defined in SGACommunicationMethods XML configuration file with the following declaration:

```
<localDataStore>
   <connectionStr>database:username:password</connectionStr >
</localDataStore>
```

It is important to be noted that the connection string must consist of the name of the database, the username and password to be used for the connection, given in the same order as above and separated with a ":" (colon character) from each other.

3. **TCP IP method**, where the SGA communicates with the IIG through TCP/IP sockets. A password and an encrypted password are used for security reasons, allowing for authentication to be performed by the receiving IIG. These passwords are defined in SGACommunicationMethods XML configuration file with the following declaration:

```
<tcpIpMethod>
    <password>bla</password>
    <passwordEnc>bla1</passwordEnc>
</tcpIpMethod >
```

4. **TCP IP with SSL method**, where the SGA communicates with the IIG using SSL over TCP/IP. In this case several communication parameters need to be set, such as the path where keystore is located, the keystore password, the key password, the keystore type, the key manager algorithm, the SSL version and the supported cipher suites.

All these are defined in SGACommunicationMethods XML configuration file with the following declaration:

```
<SSLtcpIpMethod>
    <keystorePath>D:\j2sdk1.4.1_01\bin\trustcerts</keystorePath>
    <keystorePassword>userpass</keystorePassword>
    <keyPassword>userpass</keyPassword>
    <keystoreType>JKS</keystoreType>
    <KeyManagerAlgorithm>SunX509</KeyManagerAlgorithm>
    <SSLVersion>SSLv3</SSLVersion>
    <supportedSuites>SSL_RSA_WITH_RC4_128_MD5</supportedSuites>
</SSLtcpIpMethod >
```

- `SGA.SGAIIGConfFile`: The path for the XML file that binds the symbolic name for the IIG with all physical level information required for initiating communication with the designated IIGs.

  The SGAIIG configuration file is validated against a DTD document, which must be located in the same directory with the SGAIIG configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD XML Schema describing the SGAIIG configuration file)-->
<!ELEMENT IIGInfo (IIG*)>
<!ELEMENT IIG (IIGName, address, port, credentials)>
<!ELEMENT IIGName (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT credentials (#PCDATA)>
```

  An SGAIIG configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of a SGAIIG XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE IIGInfo SYSTEM "SGAIIGConfFile.dtd">
<IIGInfo>
   <IIG>
      <IIGName>taxationIIG</IIGName>
      <address>smartgov.mm.di.uoa.gr</address>
      <port>5656</port>
      <credentials>PASSWORD:pass1</credentials>
   </IIG>
   <IIG>
      <IIGName>communityIIG</IIGName>
      <address>smartgov2.mm.di.uoa.gr</address>
      <port>3000</port>
      <credentials>PASSWORD:pass2</credentials>
   </IIG>
   <IIG>
      <IIGName>vatIIG</IIGName>
      <address>smartgov3.mm.di.uoa.gr</address>
      <port>50000</port>
      <credentials>PUBLICKEY:pass3</credentials>
   </IIG>
</IIGInfo>
```

- `SGA.AdelantePAQConfFile`: The path for the configuration file of the SGA Adelante PAQ, which is actually the configuration file of the dispatcher.
- `SGA.DatabaseStoreConfFile`: The path for the configuration file for the databaseStore method. For more details see section "gr.uoa.di.DatabaseStore package", later in this document.
- `SGLogger.propertyFile`: The path for the property file of the SGLogger.

The SGA property file must have the following form:

```
# Property file for the SG dispatcher
SGA.SGAServicesConfFile=<configuration file specification>
SGA.SGACommMethConfFile=<configuration file specification >
SGA.SGAIIGConfFile=<configuration file specification >
SGA.AdelantePAQConfFile=<configuration file specification >
SGA.DatabaseStoreConfFile=<configuration file specification >
SGLogger.propertyFile=<property file specification >
```

For example:

```
#Property file for SGA
SGA.SGAServicesConfFile=c:\\smartgov\\conf\\sga\\SGAServicesConfFile.xml
SGA.SGACommMethConfFile=c:\\smartgov\\conf\\sga\\SGACommMethConfFile.xml
SGA.SGAIIGConfFile=c:\\smartgov\\conf\\sga\\SGAIIGConfFile.xml
SGA.AdelantePAQConfFile=c:\\smartgov\\conf\\dispatcher\\dispatchCfg.txt
SGA.DatabaseStoreConfFile=c:\\smartgov\\conf\\sga\\DatabaseStoreConfFile.txt
SGLogger.propertyFile=c:\\smartgov\\conf\\SGLogging\\SGLogCfgSGA.txt
```

5.5.7.2 The SGA Adelante PAQ property file

The Adelante PAQ (SGA-APAQ) property file is provided as a property in the SGA property file and contains the necessary information for connecting with the database where the Adelante PAQ is stored. It contains the following five properties:

- The user name for connecting with the database where the Adelante PAQ is stored.
- The name of the database where the Adelante PAQ is stored.
- The password for connecting with the database where the Adelante PAQ is stored.
- The driver for connecting with the database where the Adelante PAQ is stored.
- The connection string for connecting with the database where the Adelante PAQ is stored.

The property file must have the following form:

```
SGA.AdelantePAQ.username=<username>
SGA.AdelantePAQ.database=<database name>
SGA.AdelantePAQ.password=<password>
SGA.AdelantePAQ.driver=<driver class name>
SGA.AdelantePAQ.connectString=<connection string>
```

For example,

```
# Property file for the SGA EntraPAQ
SGA.AdelantePAQ.username=smartgov
SGA.AdelantePAQ.database=smartgov
SGA.AdelantePAQ.password=sg123!
SGA.AdelantePAQ.driver=oracle.jdbc.driver.OracleDriver
SGA.AdelantePAQ.connectString=jdbc:oracle:oci8:@
```

5.5.8 gr.uoa.di.SGAClient Package

The `gr.uoa.di.SGAClient` package provides facilities for communicating with IIG through TCP/IP sockets. This package is used by `gr.uoa.di.SGA` package, when the communication with the IIG requires the use of TCP/IP sockets.

In order to use `gr.uoa.di.SGAClient`, an `SGAClientFactory` must be created first and then through the `newSGAClient` method to create an `SGAClient`. This method takes as input parameters:

The address of the IIG Server that listens for connections.

The port number where the IIG Server listens for connections.

Successful creation of the `SGAClient` signifies that the communication between SGA and IIG has been established and both sides are ready to exchange information. Messages from SGA can be sent to IIG Server through the `sendMessage` method. This method accepts as input parameter the message that SGA sends to IIG Server and returns the answer from IIG Server, so the bidirectional communication is achieved.

A sample of the code required to use in order to communicate with IIG Server (or any server listening for TCP/IP connections in a specific address and port number) is shown below:

```
//create the factory
SGAClientFact = new SGAClientFactory();

//create the client
client = SGAClientFact.newSGAClient(address, port.intValue());

//send the xmlpacket and get the answer
answer = client.sendMessage(XMLPacket);
```

5.5.8.1 Package gr.uoa.di.SGAClient

5.5.8.1.1 public class gr.uoa.di.SGAClient.SGAClientFactory

Constructors	**public SGAClientFactory()** Creates a new instance of SGAClientFactory
Methods	**public gr.uoa.di.SGAClient.SGAClient newSGAClient(** **String address,** **int port)**

5.5.8.1.2 public class gr.uoa.di.SGAClient.SGAClientException extends java.lang.Exception

Constructors	**public SGAClientException()** Creates a new instance of SGAClientException **public SGAClientException(** **String message)** Constructs a new exception instance with a given error message. **Parameters** message - The message associated with the exception. **public SGAClientException(** **Throwable nestedException)** Constructs a new exception instance that wraps another exception instance. **Parameters** The - exception to be wrapped.

5.5.8.1.3public class gr.uoa.di.SGAClient.SGAClient

Constructors	**public SGAClient(** **String address,** **int port)**
Methods	**public java.lang.String sendMessage(** **String message)**

5.5.9 gr.uoa.di.SSLSGAClient Package

The `gr.uoa.di.SSLSGAClient` package provides facilities for communicating with IIG through TCP/IP sockets over SSL. This package is used by `gr.uoa.di.SGA` package, when the communication with the IIG requires the use of secure TCP/IP sockets, in order to guarantee both the privacy and the authenticity of the exchanged messages.

In order to use `gr.uoa.di.SSLSGAClient`, an `SSLSGAClientFactory` must be created first and then through the `newSSLSGAClient` method to create an `SSLSGAClient`. This method takes as input parameters:

1. The address of the IIG Server that listens for connections over SSL.
2. The port number where the IIG Server listens for connections over SSL.

Successful creation of the `SSLSGAClient` indicates that the communication between SGA and IIG has been established and both sides are ready to exchange information. Messages from SGA can be sent to IIG Server through the `sendMessage` method. This method accepts as input parameters the message

that SGA sends to IIG Server along with information needed for verifying the client (described in detail later in this chapter) and returns the answer from IIG Server, completing the bidirectional communication.

A sample of the code required to use in order to communicate with IIG Server (or any server listening for secure TCP/IP connections in a specific address and port number) is shown below:

```
//create the factory
SSLSGAClientFact = new SSLSGAClientFactory();

//create the client
client = SSLSGAClientFact.newSSLSGAClient(address, port.intValue());

//send the xmlpacket and get the answer
answer=client.sendMessage(XMLPacket,keystorePath,keystorePassword,
      keyPassword,keystoreType,KeyManagerAlgorithm,SSLVersion, supportedSuites);
```

5.5.9.1 Package gr.uoa.di.SSLSGAClient

5.5.9.1.1 public class gr.uoa.di.SSLSGAClient.SSLSGAClientFactory

Constructors **public SSLSGAClientFactory()**

Creates a new instance of SGAClientFactory

Methods **public gr.uoa.di.SSLSGAClient.SSLSGAClient newSSLSGAClient(**
String address,
int port)

Creates a new SGA client that communicates with the IIG server using the Secure Socket Layer (SSL). The server should be running on the machine indicated by *address* and listening on the port designated by *port*.

5.5.9.1.2 public class gr.uoa.di.SSLSGAClient.SSLSGAClientException extends java.lang.Exception

Constructors **public SSLSGAClientException()**

Creates a new instance of SGAClientException

public SSLSGAClientException(String message)

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

public SSLSGAClientException(Throwable nestedException)

Constructs a new exception instance that wraps another exception instance.

Parameters

The - exception to be wrapped.

5.5.9.1.3public class gr.uoa.di.SSLSGAClient.SSLSGAClient

Constructors	**public SSLSGAClient(String address, int port)** Creates a new SGA client that communicates with the IIG server using the Secure Socket Layer (SSL). The server should be running on the machine indicated by *address* and listening on the port designated by *port*.
Methods	**public java.lang.String sendMessage(** **String message,** **String keystorePath,** **String keystorePassword,** **String keyPassword,** **String keystoreType,** **String KeyManagerAlgorithm,** **String SSLVersion,** **String[] supportedSuites)** Sends the XML message specified in *message* to the IIG server to which the SGA client is connected. In addition to the XML message, the method accepts a number of parameters related to the Secure Socket Layer and specify the path to the keystore, the keystore password, the type of the keystore, the algorithm used by the key manager, the SSL version and the suite of cryptographic algorithms that may be used for the transfer of this message.

5.5.10 gr.uoa.di.DatabaseStore Package

The `gr.uoa.di.DatabaseStore` package provides facilities for using a database as a store in IIG for the data coming from SGoVApp. This package is used by `gr.uoa.di.SGA` package, when the communication with the IIG requires data storage to a local database (`store_to_local_data_store` method).

In order to use `gr.uoa.di.DatabaseStore`, SGA has to create first a `DatabaseStoreFactory` and then through the `newDatabaseStore` method to create a `DatabaseStore`. This method accepts the following input parameters:

1. *property file*: the file path to the property file for the `DatabaseStore` package. This file contains two properties as shown below:

   ```
   DatabaseStore.driver=oracle.jdbc.driver.OracleDriver
   DatabaseStore.connectString=jdbc:oracle:oci8:@
   ```

   The first property (`DatabaseStore.driver`) defines which driver to be used for the connection with the database. Since in this example an oracle database has been used, the corresponding driver is defined by the string `oracle.jdbc.driver.OracleDriver`. If another database is used, such as SQL Server, the administrator must set this property to the appropriate value for the communication to succeed. The second property (`DatabaseStore.connectString`) should be also adjusted accordingly to suite the specific DBMS used.

2. *database name:* the name of the database where the data will be stored. The connection is an ODBC connection, so the database name is the ODBC source name that refers to the desired database.
3. The username: the username to be used during the connection with the database.
4. The password: the password to be used during the connection with the database.

A sample of the code required to use in order to communicate with the database and insert a record is shown below:

```
//create the factory
databaseStoreFact = new databaseStoreFactory();

//create the databaseStore
databaseStoreInstance=databaseStoreFact.newDatabaseStore(DatabaseStoreConfFile,Ddatabase,DuserName,Dpassword);

//open the connection
theConn = databaseStoreInstance.openConnectionWithDatabase();

//typeCat long requestId to int requestId
reqIdLong = new Long(requestId);

//insert into database the corresponding record
databaseStoreInstance.insertIntoDatabase(theConn,reqIdLong.intValue(),serviceName,XMLMessage,realTime,persistent);

//close the connection
databaseStoreInstance.closeConnectionToDatabase(theConn);
```

The `gr.uoa.di.DatabaseStore` package provides also facilities for manipulating the specific database, such as connecting with the database, record insertion, deletion, conditional and unconditional retrieval, retrieval of specific record fields, and disconnecting from the database. The database is assumed to have a table named "databaseTable" and a table named "autokeys". The structure of these tables is presented in Appendix A.

In the following paragraphs the methods for manipulating the database are presented in detail, grouped by class. This API may be used for building custom applications that manipulate the pending actions queue, e.g. an administrative application for viewing the queue contents, deleting queue entries that are considered outdated etc.

1. `public class` **`gr.uoa.di.DatabaseStore.databaseStoreFactory`**

   which is the factory for databaseStore objects

Constructors **public databaseStoreFactory()**

Methods **public gr.uoa.di.DatabaseStore.databaseStore newDatabaseStore(String propertyFile, String database, String username, String password)**

Factory method that acts as a virtual constructor for databaseStore.

Parameters

propertyFile – The databaseStore property file

database – The name of the database

username – The username to be used during the connection with the database

password – The password to be used during the connection with the database

Returns

gr.uoa.di.DatabaseStore.databaseStore

Throws

databaseStoreException - If the databaseStore creation failed

2. public class **gr.uoa.di.DatabaseStore.databaseStore**

which is the class for databaseStore objects

Constructors **public databaseStore (String propFile,String database,String username,String password)**

Parameters

propFile – The databaseStore property file

database – The name of the database

username – The username to be used during the connection with the database

password – The password to be used during the connection with the database

Methods

public Connection openConnectionWithDatabase()

Opens a connection with the database using the property file, the database name, the username and the password defined in the constructor.

Returns

Connection – The connection with the database

Throws

databaseStoreException - If the connection with the database fails

public void closeConnectionToDatabase(Connection conn)

Closes the connection conn with the database.

Parameters

conn – The opened connection with the database that must be closed

Returns

Nothing

Throws

databaseStoreException - If the disconnection with the database fails

public void insertIntoDatabase(Connection conn, int requestId, String serviceName, String XMLPacket, boolean realTime, boolean persistent)

Inserts into the database a record with the given values.

Parameters

conn – The opened connection with the database

requestId – The id of the request that requires the store in the database

serviceName – The name of the service that requires the store in the database

XMLPacket – The XMLPacket from SGovApp

realTime – A flag indicating whether the service is synchronous or not

persistent – A flag indicating whether the service is persistent or not

Returns

Nothing

Throws

databaseStoreException - If the insertion into the database fails

public ResultSet retrieveFromDatabase(Connection conn, int databaseTableId)

Retrieve from the database a record with primary key databaseTableId.

Parameters

conn – The opened connection with the database

databaseTableId – The id of the record that must be retrieved

Returns

ResultSet – The record with primary key databaseTableId

Throws

databaseStoreException - If the retrieval from the database fails

public ResultSet retrieveAllFromDatabase(Connection conn)

3. public class **gr.uoa.di.DatabaseStore.databaseStoreException extends java.lang.Exception**

This class models the exceptions thrown by the databaseStore.

Constructors **public databaseStoreException()**

Creates a new instance of dispatcherIIGException

public databaseStoreException (String message)

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

public databaseStoreException (Throwable nestedException)

Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

5.5.11 gr.uoa.di.IIGServer Package

The `gr.uoa.di.IIGServer` package provides facilities for intercepting and handling TCP/IP communication, in the form of messages sent by the SGA. The *IIG Sever* is an autonomous program that continuously runs on the Information Interchange Gateway and listens for requests for a specific service. The *IIG Server* responds to these requests and, if necessary, forwards the results to the calling service.

The *IIG Server* performs the following actions:

1. Listens to a specified port for incoming requests.
2. Upon receiving a request, starts a new thread that forwards the handle of the request to IIGMyP through the `gr.uoa.di.IIGMyP` package, that it will be described in detail later in this chapter.
3. Gets the result of the request from IIGMyP.
4. Sends the result back to the calling service.

The *IIG Server* is started by executing the class *IIGServer*, providing as parameter the port to which the *IIG Server* will listen and the location of the *IIGMyP* property file, described in the following paragraphs.

```
java gr.uoa.di.IIGServer.IIGServer <port number> <IIGMyP property file>
```

For example,

```
java gr.uoa.di.IIGServer.IIGServer 7878 c:\smartgov\conf\iig\iigMyPconfig.txt
```

The *IIG Server* creates a server socket, and listens to the given port, for incoming requests. These requests are received in the form of text messages. Upon receiving a request, the *IIG Server* starts a new thread that will serve the request calling the `gr.uoa.di.IIGMyP` package. The *IIG Server* gets the answer from IIGMyP and sends the answer back to SGA.

The IIG Server uses the SGLogger, which is described in another part of this document, to log information on the following events:

1. receive a request from SGA
2. failure to start the thread
3. failure while executing the thread

Since *IIG Server* uses SGLogger facilities, it needs to have access to a property file with the information necessary for connecting to the SGLogListener. The location of this property file is specified as a property within the *IIGMyP* property file.

5.5.12 gr.uoa.di.SSLIIGServer Package

The `gr.uoa.di.SSLIIGServer` package provides facilities for intercepting and handling TCP/IP communication over SSL, in the form of messages sent by the SGA. The *SSLIIG Sever* is an autonomous program that continuously runs on the Information Interchange Gateway and listens for requests for a specific service. The *SSLIIG Server* responds to these requests and, if necessary, forwards the results to the calling service.

The *SSLIIG Server* performs the following actions:

1. Listens to a specified port for incoming requests.
2. Upon receiving a request, starts a new thread that forwards the handle of the request to IIGMyP through the `gr.uoa.di.IIGMyP` package, that it will be described in detail later in this chapter.
3. Gets the result of the request from IIGMyP.
4. Sends the result back to the calling service.

The *SSLIIG Server* is started by executing the class *SSLIIGServer*, providing as parameter the location of the SSL XML configuration file and the location of the *IIGMyP* property file, described in the following paragraphs.

```
java gr.uoa.di.SSLIIGServer.SSLIIGServer <SSL XML configuration file> <IIGMyP property file>
```

For example,

```
java gr.uoa.di.SSLIIGServer.SSLIIGServer c:\SG\SSLPropertyFile.xml c:\SG\IIGMyPPropertyFile.txt
```

The port number to which the *SSLIIG Server* will listen for connections is defined in the SSL XML configuration file. The SSL XML configuration file contains except

from the information about the port, all the other necessary information to obtain a secure connection over the SSL layer, i.e.:

1. *Port*: the port number to which the *SSLIIG Server* will listen for connections.
2. *KeystorePath*: the file path for the keystore file.
3. *KeystorePassword*: the password for the keystore file.
4. *KeyPassword*: the password for the keys. All keys in the keystore should have the same password, not necessarily equal to the password of the keystore.
5. *KeystoreType*: the type for the keystore.
6. *SSLVersion*: the version of the SSL protocol supported.
7. *SupportedSuites*: the supported suites for the SSL communication

The SSL XML configuration file is validated against a DTD document, which must be located in the same directory with the SSL XML configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD describing the SSL XML document for SSLIIGServer-->
<!ELEMENT serverInfo (port, keystorePath, keystorePassword, keyPassword, keystoreType,
KeyManagerAlgorithm, SSLVersion, supportedSuites+)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT keystorePath (#PCDATA)>
<!ELEMENT keystorePassword (#PCDATA)>
<!ELEMENT keyPassword (#PCDATA)>
<!ELEMENT keystoreType (#PCDATA)>
<!ELEMENT KeyManagerAlgorithm (#PCDATA)>
<!ELEMENT SSLVersion (#PCDATA)>
<!ELEMENT supportedSuites (#PCDATA)>
```

An SSL XML configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of a SSL XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE serverInfo SYSTEM "serverInfo.dtd">
<serverInfo>
    <port>5858</port>
    <keystorePath>D:\j2sdk1.4.1_01\bin\user.keystore</keystorePath>
    <keystorePassword>userpass</keystorePassword>
    <keyPassword>userpass</keyPassword>
    <keystoreType>JKS</keystoreType>
    <KeyManagerAlgorithm>SunX509</KeyManagerAlgorithm>
    <SSLVersion>SSLv3</SSLVersion>
    <supportedSuites>SSL_RSA_WITH_RC4_128_MD5</supportedSuites>
    <supportedSuites>SSL_DH_anon_WITH_RC4_128_MD5</supportedSuites>
    <supportedSuites>SSL_DH_anon_WITH_3DES_EDE_CBC_SHA</supportedSuites>
    <supportedSuites>SSL_DH_anon_WITH_DES_CBC_SHA</supportedSuites>
    <supportedSuites>SSL_DH_anon_EXPORT_WITH_RC4_40_MD5</supportedSuites>
    <supportedSuites>SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA</supportedSuites>
</serverInfo>
```

The *SSLIIG Server* creates a server socket, and listens to the given port, for incoming requests. These requests are received in the form of text messages. Upon receiving a request, the *SSLIIG Server* starts a new thread that will serve the request calling the `gr.uoa.di.IIGMyP` package. The *SSLIIG Server* gets the answer from IIGMyP and sends the answer back to SGA. It is important to note that in order for the *SSLIIG Server* to understand the end of the message received from SGA, the message must end with the string "`</XMLPacket>`" in a separate line.

The *SSLIIG Server* uses the SGLogger, which is described in another part of this document, to log information on the following events:

1. receive a request from SGA
2. failure to start the thread
3. failure while executing the thread

Since *SSLIIG Server* uses SGLogger facilities, it needs to have access to a property file with the information necessary for connecting to the SGLogListener. The location of this property file is specified as a property within the *IIGMyP* property file.

5.5.13 Prerequisites for using SSL communication

The SSL communications library can be used to protect the data secrecy, integrity and the authenticity of peer parties. More specifically, the following features are provided:

1. *data secrecy*: even if an eavesdropper captures the data, these are in an incomprehensible and thus useless form.
2. *data integrity:* if a malicious party attempts to inject data into the communication channel or alter the data exchanged, the tampering will be detected and rejected by the communication layer.
3. *peer party authenticity:* certain encryption algorithms (or *ciphers*) are able to guarantee the identity of the communicating parties. In particular, all ciphers **except** the ones whose names include the `_none_` literal are able to guarantee peer party authenticity. This is accomplished by either a prior exchange of *keys and certificates* or by relying on a trusted third party to testify for the peer party identity (a *certification authority*).

Organisations not willing to use a third party for the purpose of peer authentication may generate their own keys and certificates and install them. For the purposes of certificate creation, the command batches `mkcerts` (for Unix environments) and `mkcerts.bat` (for Windows environments) are provided. System administrators can edit these scripts to modify the appropriate parameters (in particular, the *entity distinguished name* should be modified; it is also recommended –though not mandatory- that the passwords are modified too) and then execute them. Execution of these command batches will generate two files, namely `cert-keystore` and `client-keystore`. The former file should be installed on the server running the SSL IIG Server, and the `keystore` element of the SSL XML configuration file should be edited to point to the file. The latter should be installed on the clients that should access the server, and the `keystorePath` element of the relevant `SSLtcpIpMethod` declarations in file SGA communication methods configuration file should be edited to point to this file. Finally, both in the SSL XML configuration file and in the `SSLtcpIpMethod` declarations in file SGA communication methods configuration file, the encryption algorithm designation `SSL_DHE_DSS_WITH_DES_CBC_SHA` should be included in the supported cipher suites, since the generated keys are suitable only for this cipher. The generation of suitable keys and certificates for use with other ciphers can be performed by editing the command batches and adding the appropriate options to the invocations of the `keytool` command.

5.5.14 gr.uoa.di.IIGMyP package

The `gr.uoa.di.IIGMyP` package provides facilities for processing the SGA requests received from either *IIG Server* or *SSLIIG Server*. This package is used by `gr.uoa.di.IIGServer` or `gr.uoa.di.SSLIIGServer` package The *IIGMyP*

accepts these requests from *IIG Server* or *SSLIIG Server*, processes them and forwards the results to the corresponding *IIG Server* or *SSLIIG Server*.

More specific the *IIGMyp* performs the following actions:

1. Receives a request from *IIG Server* or *SSLIIG Server*. The request is assumed to be in XML format.
2. Parses the XML request and validates it against the XML schema used for this message exchange.
3. The symbolic `service name` is checked to verify that this IIGMyP supports the service specified.
4. The security credentials of SGA are checked to verify that the SGA that sends the request for the service name is qualified to do so.
5. If the service requested is synchronous, the *IIGMyP* serves the request. For this purpose the *IIGMyP* looks up in the IIG MyP XML communication file the information concerning the method associated with the request. The options available for serving a request are:
    a. execution of a Java method.
    b. execution of an OS-level program.
    c. storage of the request to a database
    d. storage of the request to an OS file.
6. Attempts to serve the request using the designated methods.
7. Sends the result of the method back to the calling *IIG Server* or *SSLIIG Server*.
8. If the service requested is asynchronous, the *IIGMyP* stores the request in EntraPAQIIG and delegates the responsibility of request handling to the dispatcher. This approach frees the communication channel and offloads the IIG, facilitating the processing of urgent, i.e. real-time, messages. Another advantage of this approach is that further processing of messages in the PAQ can be aligned with local IT system policies.

In order to verify that the originator of the request is valid, *IIGMyP* looks up in the IIGSecurityFile the information needed for the verification. The file path for IIGSecurityFile is defined in IIGMyP property file, via property IIGMyP.IIGSecurityFile.

The IIGSecurityFile XML file is validated against a DTD document, which must be located in the same directory with the IIGSecurityFile XML file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT IIGSecurity (IIGCredentials*)>
<!ELEMENT IIGCredentials (serviceName, credentialInfo+)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT credentialInfo (IPAddress, credentials)>
<!ELEMENT IPAddress (#PCDATA)>
<!ELEMENT credentials (#PCDATA)>
```

An IIGSecurityFile XML file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of a IIGSecurityFile XML file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE IIGSecurity SYSTEM "IIGSecurity.dtd">
<IIGSecurity>
    <IIGCredentials>
        <serviceName>getTaxPay</serviceName>
        <credentialInfo>
            <IPAddress>10.10.10.2</IPAddress>
            <credentials>PUBLICKEY:key1</credentials>
        </credentialInfo>
        <credentialInfo>
            <IPAddress>10.10.10.3</IPAddress>
            <credentials>PASSWORD:pass2</credentials>
        </credentialInfo>
    </IIGCredentials>
    <IIGCredentials>
        <serviceName>getContact</serviceName>
        <credentialInfo>
            <IPAddress>10.10.10.4</IPAddress>
            <credentials>PUBLICKEY:key3</credentials>
        </credentialInfo>
        <credentialInfo>
            <IPAddress>10.10.10.5</IPAddress>
            <credentials>PASSWORD:pass4</credentials>
        </credentialInfo>
    </IIGCredentials>
    <IIGCredentials>
        <serviceName>getPersonalInfo</serviceName>
        <credentialInfo>
            <IPAddress>10.10.10.6</IPAddress>
            <credentials>PUBLICKEY:key5</credentials>
        </credentialInfo>
        <credentialInfo>
            <IPAddress>10.10.10.7</IPAddress>
            <credentials>PASSWORD:pass6</credentials>
        </credentialInfo>
        <credentialInfo>
            <IPAddress>10.10.10.8</IPAddress>
            <credentials>PASSWORD:pass7</credentials>
        </credentialInfo>
    </IIGCredentials>
</IIGSecurity>
```

For each service available through a specific IIG the corresponding IIGSecurityFile XML file contains the following information:

1. IPAddress: the IP address of the SGA that is allowed to send a request for the specific service to the specific *IIG Server* or *SSLIIG Server*.
2. Credentials: the SGA credentials that verify the authentication of the SGA. These credentials can have the form either of a password or of a public key. For the first case the credentials must be defined by the string "PASSWORD:SGApassword". It is important to be noted that the keyword "PASSWORD" and the value of the password must be given in the same order and separated with ":" from each other. For the verification of the SGA, a string comparison between the password given in IIGSecurityFile XML file and the password received from SGA is performed. If both passwords are identical the verification of SGA is successful. When authentication is to be performed by means of a public key, the credentials must be specified using a string of the format "PUBLICKEY:SGApublickey". It is important to be noted that the public key specification should follow the format designated above i.e. the keyword "PUBLICKEY" followed by a colon and the actual value of the public key, in that order. No algorithm is currently provided authenticating an SGA by means of a public key; however, the method `checkPublKeyData(String publikKey1, String publikKey2)` within the class `gr.uoa.di.IIGMyP.checkPublicKey` is provided as a placeholder. Administrators may provide a suitable implementation for comparing two public keys, achieving thus the desired authentication scheme.

For a specific service, it is possible to specify multiple credentials for authenticating a single IP address. In such a case, it the credentials presented by the SGA should match at least one of the credentials designated in the configuration file, for the authentication to be considered successful.

The *IIGMyP* uses the SGLogger, which is described in another part of this document, to log information on the following events:

1. receiving of a request, either from an *IIG Server* or an *SSLIIG Server*.
2. success or failure of the XML message validation.
3. success or failure of the verification whether the service specified in the XML message is supported by the IIGMyP.
4. success or failure of the SGA authentication process.
5. success or failure of the execution of the requested service.
6. end of the request processing.

Since *IIGMyP* uses SGLogger facilities, it needs to have access to a property file with the information necessary for connecting to the SGLogListener. The location

of this property file is specified as a property within the *IIGMyP* property file, described in the next paragraph.

5.5.14.1 The IIGMyP property file

The IIGMyP property file is used by either `gr.uoa.di.IIGServer` or `gr.uoa.di.SSLIIGServer` package when the IIG/SSLIIG Server is started and contains seven properties, which are the following:

1. IIGMyP.IIGServicesConfFile: The path for the XML file that binds the service names that IIG can serve, with corresponding a SEP name and a list of methods to be used for the communication between IIG and SEP.
   The IIGServices configuration file is validated against a DTD document, which must be located in the same directory with the IIGServices configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD describing the IIG Services Configuration XML document-->
<!ELEMENT IIGServices (service*)>
<!ELEMENT service (serviceName, SEPName, methodName+)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT SEPName (#PCDATA)>
<!ELEMENT methodName (#PCDATA)>
```

   An IIGServices configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an IIGServices XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE IIGServices SYSTEM "IIGServicesConfFile.dtd">
<IIGServices>
    <service>
        <serviceName>getPersonalInfo</serviceName>
        <SEPName>taxationIIG</SEPName>
        <methodName>localDataStore1</methodName>
    </service>
    <service>
        <serviceName>getPersonalInfo1</serviceName>
        <SEPName>communityIIG</SEPName>
        <methodName>executeJavaProcedure2</methodName>
        <methodName>localFileStore2</methodName>
        <methodName>executeCommand1</methodName>
        <methodName>localFileStore1</methodName>
        <methodName>localDataStore1</methodName>
        <methodName>executeJavaProcedure1</methodName>
    </service>
    <service>
        <serviceName>getPersonalInfo1</serviceName>
        <SEPName>vatIIG</SEPName>
        <methodName>executeCommand3</methodName>
        <methodName>localDataStore3</methodName>
        <methodName>localFileStore3</methodName>
    </service>
    <service>
        <serviceName>getContact</serviceName>
        <SEPName>communityIIG</SEPName>
        <methodName>executeCommand4</methodName>
    </service>
</IIGServices>
```

2. `IIGMyP.IIGCommMethConfFile`: The path for the XML file that binds the symbolic name for each method used to serve requests with all the physical level information required for implementing the designated method.

   The `IIGCommMethConfFile` configuration file is validated against a DTD document, which must be located in the same directory with the `IIGCommMethConfFile` configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD describing the IIGCommunicationMethods XML document-->
<!ELEMENT commMethods (method*)>
<!ELEMENT method (methodName, (javaProcedure | execCommand | localDataStore | localFileStore))>
<!ELEMENT methodName (#PCDATA)>
<!ELEMENT javaProcedure (objectPath, className, procedureName)>
<!ELEMENT execCommand (info)>
<!ELEMENT localDataStore (connectionStr)>
<!ELEMENT localFileStore (fileName)>
<!ELEMENT objectPath (#PCDATA)>
<!ELEMENT className (#PCDATA)>
<!ELEMENT procedureName (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT connectionStr (#PCDATA)>
<!ELEMENT fileName (#PCDATA)>
```

An `IIGCommunicationMethods` configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an `IIGCommunicationMethods` XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE commMethods SYSTEM "IIGCommMethConfFile.dtd">
<commMethods>
    <method>
        <methodName>executeJavaProcedure1</methodName>
        <javaProcedure>
            <objectPath>c:\\smartgov\SEPs\tests.jar</objectPath>
            <className>gr.uoa.di.testJavaProcedure</className>
            <procedureName>testingMethod</procedureName>
        </javaProcedure>
    </method>
    <method>
        <methodName>executeCommand1</methodName>
        <execCommand>
            <info>test3</info>
        </execCommand>
    </method>
    <method>
        <methodName>localDataStore1</methodName>
        <localDataStore>
            <connectionStr>smartgov:smartgov:sg123</connectionStr>
        </localDataStore>
    </method>
    <method>
        <methodName>localFileStore1</methodName>
        <localFileStore>
            <fileName>c:\smartgov\filestores\file1.txt</fileName>
        </localFileStore>
    </method>
</commMethods>
```

There are 4 types of supported methods that IIG may use to serve a request, which are presented below:

- **localFileStore method**, where the IIG stores the XML Message in a file, whose location and filename is specified in the IIGCommunicationMethods XML configuration file by means of a *localFileStore* element, e.g.:

  ```
  <localFileStore>
      <fileName>c:\smartgov\filestores\file.txt</fileName>
  </localFileStore>
  ```

- **localDataStore method**, where the IIG stores the XML Message in a database. The information for connecting to the database, i.e. the database name, username and password are defined in the

IIGCommunicationMethods XML configuration file by means of a *localDataStore* element e.g.:

```
<localDataStore>
   <connectionStr>database:username:password</connectionStr >
</localDataStore>
```

It is important to be noted that the connection string must follow the format designated above, i.e. list the database name, user name and password, in that order, separating successive elements using the colon (:) character.

- **execCommand method**, where the IIG executes an operating system-level command. The actual command to be executed is specified in the *info* element within the IIGCommunicationMethods XML configuration file, e.g.:

```
<execCommand>
      <info>command2</info>
</execCommand>
```

Service programmers should note that when an external command is executed, the SGA receives a message indicating SUCCESS, but it has also the contents of the output and error file created during the execution. So, if the execution of the command has failed, this will be reported in the error file. Therefore, the SGA has to check what the error file contains, in order to check the success of the execution.

- **javaProcedure method**, where the IIG executes a Java procedure. The actual java procedure to be executed is specified by means of a *procedureName* element within the IIGCommunicationMethods XML configuration file, e.g.:

```
<javaProcedure>
     <objectPath>gr.uoa.di.testJavaProcedure</objectPath>
     <className>gr.uoa.di.testJavaProcedure</className>
     <procedureName>testingMethod</procedureName>
</javaProcedure>
```

3. IIGMyP.IIGSEPConfFile: The path for the XML file that binds the symbolic name for the SEP with all physical level information required for initiating executing the specified SEP.

   The IIGSEPG configuration file is validated against a DTD document, which must be located in the same directory with the IIGSEP configuration file. The DTD contents are as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD describing the SG IIG SEP Configuration File-->
<!ELEMENT SEPInfo (SEP*)>
<!ELEMENT SEP (SEPName, commandPath, workingDirectory, parameters, inputFile, outputFile,
errorFile, envVariable*)>
<!ELEMENT SEPName (#PCDATA)>
<!ELEMENT commandPath (#PCDATA)>
<!ELEMENT workingDirectory (#PCDATA)>
<!ELEMENT parameters (#PCDATA)>
<!ELEMENT inputFile (#PCDATA)>
<!ELEMENT outputFile (#PCDATA)>
<!ELEMENT errorFile (#PCDATA)>
<!ELEMENT envVariable (envVariableName, envVariableValue)>
<!ELEMENT envVariableName (#PCDATA)>
<!ELEMENT envVariableValue (#PCDATA)>
```

An IIGSEP configuration file should contain only the name and not the full path of the DTD document. As stated above, the DTD document should reside in the same directory with the document. An example of an IIGSEP XML configuration file is illustrated bellow:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE SEPInfo SYSTEM "IIGSEPConfFile.dtd">
<SEPInfo>
    <SEP>
        <SEPName>taxationIIG</SEPName>
        <commandPath>java taxationIIG</commandPath>
        <workingDirectory>Z:\smartgov\wp\wp06\api\sga</workingDirectory>
        <parameters>-cp tax.jar</parameters>
        <inputFile>C:\taxation\input.txt</inputFile>
        <outputFile>C:\taxation\output.txt </outputFile>
        <errorFile> C:\taxation\error.txt</errorFile>
    </SEP>
    <SEP>
        <SEPName>vatIIG</SEPName>
        <commandPath>vatIIG</commandPath>
        <workingDirectory> Z:\smartgov\wp\wp06\api\sga</workingDirectory>
        <parameters> </parameters>
        <inputFile> </inputFile>
        <outputFile> C:\VAT\output.txt </outputFile>
        <errorFile> C:\VAT\error.txt </errorFile>
        <envVariable>
            <envVariableName>EnvVariable1</envVariableName>
            <envVariableValue>EnvVariableValue1</envVariableValue>
        </envVariable>
        <envVariable>
            <envVariableName>EnvVariable2</envVariableName>
            <envVariableValue>EnvVariableValue2</envVariableValue>
        </envVariable>
    </SEP>
</SEPInfo>
```

4. `IIGMyP.IIGSecurityFile`: The path for the IIGSecurity XML file that is used for the SGA verification and has been described in detail above.
5. `IIGMyP.EntraPAQConfFile`: The path for the configuration file of the IIG-EPAQ. For more details see the section "gr.uoa.di.dispatcherIIG Package".
6. `IIGMyP.SEPDatabaseStoreConfFile`, The path for the configuration file of the SEP Data Store. For more details see the section "gr.uoa.di.SEPDatabaseStore Package".
7. `IIGMyP.Logger.propertyFile`: The path for the property file of the SGLogger.
8. `IIGMyP.XMLPacketPath`: The containing folder for the file `XMLPacket.dtd`, specified using the URI notation. If the file is located on the file system, rather than a web server, the prefix `file:///` should be pre-pended to the folder path; additionally, the forward slash (`/`) should be *always* used

as the path component separator, rather than the back slash (\), even in Windows-based installations.

Thegeneric format for the IIG Server property file is as follows:

```
#Property file for IIG Server
IIGMyP.IIGServicesConfFile=<configuration file specification>
IIGMyP.IIGCommMethConfFile=<configuration file specification>
IIGMyP.IIGSEPConfFile=<configuration file specification>
IIGMyP.IIGSecurityFile=<configuration file specification>
IIGMyP.EntraPAQConfFile=<configuration file specification>
IIGMyP.SEPDatabaseStoreConfFile==<configuration file specification>
IIGMyP.Logger.propertyFile=<property file specification>
IIGMyP.XMLPacketPath=<containing folder for XMLPacket.dtd in URI notation>
```

For example:

```
#Property file for IIG Server
IIGMyP.IIGServicesConfFile=c:\\smartgov\\conf\\IIG\\IIGSrvCfg.xml
IIGMyP.IIGCommMethConfFile=c:\\smartgov\\conf\\IIG\\IIGCommMethCfg.xml
IIGMyP.IIGSEPConfFile=c:\\smartgov\\conf\\IIG\\IIGSEPCfg.xml
IIGMyP.IIGSecurityFile=c:\\smartgov\\conf\\IIG\\IIGSecurity.xml
IIGMyP.EntraPAQConfFile=c:\\smartgov\\cfg\\IIG\\EntraPAQIIGCfg.txt
IIGMyP.SEPDatabaseStoreConfFile=c:\\smartgov\\conf\\IIG\\SEPDbCfg.txt
IIGMyP.Logger.propertyFile=c:\\smartgov\\conf\\IIG\\SGLogCfgIIG.txt
IIGMyP.XMLPacketPath=file:///C:/smartgov/conf/IIG
```

During the processing of the request, *IIGMyP* tries to execute all the available methods that are defined for the specific service in the XML configuration files, as stated above. The success of only one type of the methods (localFileStore, localDataStore, execCommand, javaProcedure) is enough for the request execution to be considered successful. Errors that may occur in each of the executed methods are being captured and, in the case that all methods fail, the associated error descriptions are sent back as reply to the calling *IIG Server* or *SSLIIG Server*, which in turn forwards this reply to the requesting SGA. Thus, the SGA can examine the reply to determine the reason for the failure, and possibly take appropriate remedial activities. In case that at least one method succeeds, *IIGMyP* sends back a success message to the calling *IIG Server* or *SSLIIG Server*. Special care should be taken in the handling of replies when an operating system-level program is executed. In this case, it is not possible to determine whether the external program has succeeded or failed, thus the *IIGMyP* sends back a reply indicating successful execution, but additionally arranges so that the normal output and error messages emitted by the program to be bundled within the message. The calling SGA should examine these contents to determine whether

the external program execution was actually successful, taking into account the semantics and expected behaviour of the executed program.

5.5.14.2 The IIG Entra PAQ property file

The IIG EntraPAQ (IIG-EPAQ) property file is provided as a property in the IIG-MyP property file and contains the necessary information for connecting with the database where the Entra PAQ is stored. It contains the following four properties:

- The user name for connecting with the database where the Entra PAQ is stored.
- The name of the database where the Entra PAQ is stored.
- The password for connecting with the database where the Entra PAQ is stored.
- The driver for connecting with the database where the Entra PAQ is stored.
- The connection string for connecting with the database where the Entra PAQ is stored.

The property file must have the following form:

```
IIG.EntraPAQ.username=<username>
IIG.EntraPAQ.database=<database name>
IIG.EntraPAQ.password=<password>
IIG.EntraPAQ.driver=<driver class name>
IIG.EntraPAQ.connectString=<connection string>
```

For example

```
# Property file for the SGA EntraPAQ
IIG.EntraPAQ.username=smartgov
IIG.EntraPAQ.database=smartgov
IIG.EntraPAQ.password=sg123
IIG.EntraPAQ.driver=oracle.jdbc.driver.OracleDriver
IIG.EntraPAQ.connectString=jdbc:oracle:oci8:@
```

5.5.15 gr.uoa.di.SEPDatabaseStore Package

The `gr.uoa.di.SEPDatabaseStore` package provides facilities for servicing incoming requests by means of storing the incoming XML packet within a database. This package is used by `gr.uoa.di.IIGMyP` package, when the service method designated for a service is that of storing data to a local database (store_to_local_data_store method).

In order to use the facilities provided by the `gr.uoa.di.SEPDatabaseStore package`, the IIGMyP creates first a `SEPDatabaseStoreFactory` object and then

employs the `newSEPDatabaseStore` method to create a `SEPDatabaseStore` object. The `newSEPDatabaseStore` method accepts the following input parameters:

1. *configuration file* i.e. file path to the configuration file for the `SEPatabaseStore` package. The generic format for this file is as follows:

   ```
   SEP.DatabaseStore.driver=<driver class name>
   SEP.DatabaseStore.connectString=<connection string>
   ```

   For example,

   ```
   # Property file for the SEPDatabaseStore
   DatabaseStore.driver=oracle.jdbc.driver.OracleDriver
   DatabaseStore.connectString=jdbc:oracle:oci8
   ```

   The first property (`SEP.DatabaseStore.driver`) defines which driver to be used for the connection with the database. Since in this example an Oracle database has been used, the corresponding driver is defined by the string `oracle.jdbc.driver.OracleDriver`. If another database is used, such as SQL Server, the administrator must set this property to the appropriate value for the communication to succeed. The second property (`SEP.DatabaseStore.connectString`) should be also adjusted accordingly to suite the specific DBMS used.
2. *database name*, which is the name of the database where the data will be stored. The connection is an ODBC connection, so the database name is the ODBC source name that refers to the desired database.
3. The username: the username to be used during the connection with the database.
4. The password: the password to be used during the connection with the database.

A sample of the code required to use in order to communicate with the database and insert a record is shown below:

```
//create the factory
SEPDbStoreFact = new SEPDatabaseStoreFactory();

//create the databaseStore
SEPDatabaseStoreInstance = SEPDbStoreFact.newSEPDatabaseStore(
                  SEPDatabaseStoreConfFile, Ddatabase, DuserName, Dpassword);

//open the connection
theConn = SEPDatabaseStoreInstance.openConnectionWithDatabase();
//insert into database the corresponding record
SEPDatabaseStoreInstance.insertIntoDatabase(theConn, serviceName, decodedXMLMessage,
                  realTime);

//close the connection
SEPDatabaseStoreInstance.closeConnectionToDatabase(theConn);
```

The `gr.uoa.di.SEPDatabaseStore` package also provides facilities for manipulating the specific database, such as connecting to the database, inserting records, deleting records, retrieving all records, retrieving a specific record, retrieving specific fields from a record, and finally disconnecting from the database. The database is assumed to have a table named "`SEPDatabaseTable`" and a table named "`autokeys`". The structure of these tables is presented in Appendix A.

The facilities provided by the package may be used to write custom, specialised tools for managing the entries stored in the database, e.g. for reading the records and taking the relevant actions. To this end, the API provided by the `gr.uoa.di.SEPDatabaseStore` is documented in the following paragraphs.

5.5.15.1 Package gr.uoa.di.SEPDatabaseStore

5.5.15.1.1 public class gr.uoa.di.SEPDatabaseStore.SEPDatabaseStoreFactory

Constructors	**public SEPDatabaseStoreFactory()** Creates a new instance of DatabaseStoreFactory

Methods **public gr.uoa.di.SEPDatabaseStore.SEPDatabaseStore newSEPDatabaseStore(**
String propFile,
String database,
String username,
String password)

Factory method that acts as a virtual costructor for adelantePAQ.

Parameters

propFile - The property file containing necessary parameters for adelantePAQ

Throws

adelantePAQException - If the adelantePAQ creation failed

5.5.15.1.2 public class gr.uoa.di.SEPDatabaseStore.SEPDatabaseStoreException extends java.lang.Exception

Constructors **public SEPDatabaseStoreException()**

Creates a new instance of DatabaseStoreException

public SEPDatabaseStoreException(String message)

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

public SEPDatabaseStoreException(Throwable nestedException)

Constructs a new exception instance that wraps another exception instance.

Parameters

The - exception to be wrapped.

5.5.15.1.3 public class gr.uoa.di.SEPDatabaseStore.SEPDatabaseStore

Constructors **public SEPDatabaseStore(**
String propFile,
String Ddatabase,
String Dusername,
String Dpassword)

Creates a new instance of SEPDatabaseStore.

Methods **public java.sql.Connection openConnectionWithDatabase()**

Opens a connection to the database specified by the parameters passed to the constructor.

public void insertIntoDatabase(
Connection conn,
String serviceName,
String XMLMessage,
boolean realTime)

Inserts into the database a new record, mimicking the execution via a

store_to_local_data_store method of a request for “serviceName”, having the XMLMessage as parameter and its real time flag set to realTime.

public java.sql.ResultSet retrieveFromDatabase(
Connection conn,
int SEPdatabaseTableId)

This method retrieves from the database the record whose identifier matches the parameter SEPdatabaseTableId.

public java.sql.ResultSet retrieveAllFromDatabase(Connection conn)

This method retrieves from the database all the records.

public int getSEPDatabaseTableId(ResultSet resultRecord)

This method returns the SEPDatabaseTableId from a specific result set.

public java.lang.String getServiceName(ResultSet resultRecord)

This method returns the service name from a specific result set.

public java.lang.String getXMLMessage(ResultSet resultRecord)

This method returns the XML message from a specific result set.

public int getRealTime(ResultSet resultRecord)

This method returns the real time flag from a specific result set.

public java.lang.String getTimeStamp(ResultSet resultRecord)

This method returns the timestamp from a specific result set. This is equal to the time that the record was inserted in the database.

public void deleteFromDatabase(Connection conn, int SEPdatabaseTableId)

This method deletes from the database the record whose identifier matches the parameter SEPdatabaseTableId.

public void closeConnectionToDatabase(Connection conn)

Closes the connection to the database.

5.6 Database objects documentation - IIG and SGA Entra and Adelante PAQ Structure

The SmartGov platform requires the implementation of the IIG and SGA Pending Action Queues (PAQs) in order to function properly. These are implemented as database tables, the structure of which is presented later in this document. For each PAQ a java package was developed for its management. The class methods provided in these packages are used by the SmartGov processes, which need to insert, retrieve and delete entries from the PAQs.

5.6.1 The autokeys table

In order to provide to each database table a unique identifier, which will generate new values for the table primary key, the *autokeys* table must be created. It should contain the following fields:

Name	Time	Descriptions	Restrictions
keyName	VARCHAR2(32)	The name of the key, which must be unique.	NOT NULL, PRIMARY KEY
keyValue	NUMBER(38)	The current value of the key. It contains the latest value of the primary key of the respecting database table.	NOT NULL

It is not necessary to insert initial values for the primary keys of each table when creating the autokeys table. Key records with 0 as initial value are inserted automatically when a requested key is not found in the table. However, if the need arises to set a key to a specific initial value, it should be verified that no records exist with the designated key and value greater or equal than the desired key value. When it has been verified that no such record exists, a new tuple should be inserted in the autokeys table using an insert statement of the following form:

```
insert into autoKeys values('APAQ_IIG',0);
```

5.6.2 SGA Entra PAQ

The SGA Entra PAQ is used for the storage of incoming notifications and their associated method descriptions. When the SGA Notification Interceptor (gr.uoa.di.SGANI package) receives a notification name from the IIG Notification Initiator (gr.uoa.di.IIGNI package), it retrieves from the SGA-NI configuration file the information of the method associated with this notification. Then it inserts the method information along with the notification name and a time stamp to the SGA Entra PAQ. The SGA Entra PAQ is periodically scrutinized by the SGA dispatcher (gr. uoa.di.dispatcher package), which executes the method associated with each notification.

Name	**Time**	**Descriptions**	**Restrictions**
entraPAQId	NUMBER(38)	Unique identifier that serves to characterize the record in the SGA Entra PAQ	NOT NULL, PRIMARY KEY
XMLMethodDescription	VARCHAR2(4000)	The description in XML format of the method associated with this notification	NOT NULL
SGtimestamp	VARCHAR2(22)	The date and time when this record is inserted in the SGA Adelante PAQ.	NOT NULL
NotificationName	VARCHAR2(500)	A symbolic name for the notification event.	NOT NULL

5.6.3 SGA Adelante PAQ

The SGA Adelante PAQ is used for the storage of requests that the SGA could not send to the appropriate IIG. When the SGA receives a request in the form of a symbolic service name, it retrieves from the SGA configuration file the information associated with the service. If the communication methods for the specific service fail, the request message is inserted in the SGA Adelante PAQ, along with the service name, the real time and persistent indicators and a time stamp. The SGA Adelante PAQ is periodically scrutinized by the SGA dispatcher (gr. uoa.di.dispatcher package), which attempts to resend the requests.

Name	Time	Description	Restrictions
adelantePAQId	NUMBER(38)	Unique identifier that serves to characterize the record in the SGA Adelante PAQ	NOT NULL, PRIMARY KEY
requestId	NUMBER(38)	A unique request identifier that serves to characterize this request	NOT NULL
serviceName	VARCHAR2(500)	A symbolic service name that the message refers to. The receiving SGA is expected to forward the encapsulated XMLPacket to the named service	NOT NULL
XMLPacket	VARCHAR2(4000)	A message that contains all information that the named serviceName requires. The SGA does not interpret this message, rather it is passed as is to the next step	NOT NULL
realTime	NUMBER(2)	Indicates whether the communication event is happening in real-time and consequently an immediate response is expected. When this flag is set, the SGA does not close the communication channel with the SgovApp but it immediately forwards the message to the appropriate IIG and returns the result to the calling SgoVApp	NOT NULL
persistent	NUMBER(2)	Indicates whether the message should persist in case of communication errors or other abruptions and retransmitted later. If this flag is set, message is stored in the Pending Actions Queue.	NOT NULL

Name	Time	Description	Restrictions
SGtimestamp	VARCHAR2(22)	The date and time when this record is inserted in the SGA Adelante PAQ.	NOT NULL

5.6.4 IIG Entra PAQ

The IIG Entra PAQ is used for the storage of non real-time messages received from the SGA. When a non real time message is received, that is a message where the realTime indicator is not set, it is stored in the Entra Pending Actions Queue (PAQ). Storage takes place as soon as the message reaches the IIG and after the first stage of the IIG-MYP has been completed and therefore it has been identified that it is not a real-time message. This way, the communication channel is freed as soon as possible and the IIG is free to process urgent, i.e. real-time, messages. The IIG dispatcher (gr.uoa.di.dispatcherIIG.dispatcherIIG) scrutinizes periodically the IIG Entra PAQ and processes the messages.

Name	Time	Description	Restrictions
entraPAQId	NUMBER(38)	Unique identifier that serves to characterize the record in the IIG Entra PAQ	NOT NULL, PRIMARY KEY
serviceName	VARCHAR2(500)	A symbolic name referring to a business operation. It should be the same name used in the initial request	NOT NULL
XMLMessage	VARCHAR2(4000)	The same as the XMLPacket part of the original request.	NOT NULL
realTime	NUMBER(2)	The same meaning and value as in original request. The SGA expects to receive "immediate" answer from the IIG. The IIG should process the message as soon as it receives it and reply accordingly.	NOT NULL
SGtimestamp	VARCHAR2(22)	The date and time the record was inserted in the IIG Entra PAQ	NOT NULL

5.6.5 IIG Adelante PAQ

The IIG Adelante PAQ is used for the storage of failed outgoing notifications. When the IIG Notification Initiator (gr.uoa.di.IIGNI package) attempts to send a notification to the SGA Notification Interceptor (gr.uoa.di.SGANI package, it retrieves from the IIG-NI configuration file the communication information associated with this notification. If the communication fails, the IIG-NI inserts the notification name along with a time stamp to the IIG Adelante PAQ. The IIG Adelante PAQ is periodically scrutinized by the IIG dispatcher (gr.di.uoa.dispatcher package), which executes the method associated with each notification.

Name	Time	Description	Restrictions
adelantePAQId	NUMBER(38)	Unique identifier that serves to characterize the record in the IIG Adelante PAQ	NOT NULL PRIMARY KEY
notificationName	VARCHAR2(500)	A symbolic name for the notification event.	NOT NULL
SGtimestamp	VARCHAR2(22)	The date and time the record was inserted in the IIG Adelante PAQ	NOT NULL

5.6.6 databaseTable

This table is used as the persistent storage medium for the SGA localDataStore method.

Name	Time	Description	Restrictions
databaseTableId	NUMBER(38)	Unique identifier that serves to characterize the record in this table	NOT NULL, PRIMARY KEY
requestId	NUMBER(38)	A unique request identifier that serves to characterize this request	NOT NULL
serviceName	VARCHAR2(500)	A symbolic service name that the message refers to.	NOT NULL
XMLMessage	VARCHAR2(4000)	A message that contains all information that the named serviceName requires.	NOT NULL
realTime	NUMBER(2)	Indicates whether the communication event is happening in real-time and consequently an immediate response is expected. When this flag is set, the SGA does not close the communication channel with the SGoVApp but it immediately forwards the	NOT NULL

Name	Time	Description	Restrictions
		message to the appropriate IIG and returns the result to the calling SGoVApp	
persistent	NUMBER(2)	Indicates whether the message should persist in case of communication errors or other abruptions and retransmitted later. If this flag is set, message is stored in the Pending Actions Queue.	NOT NULL
SGtimestamp	VARCHAR2(22)	The date and time the record was inserted in the table	NOT NULL

5.6.7 SEPdatabaseTable

This table is used as the persistent storage medium for the IIG-myP localDataStoremethod.

Name	Time	Description	Restrictions
SEPdatabaseTableId	NUMBER(38)	Unique identifier that serves to characterize the record in this table	NOT NULL, PRIMARY KEY
serviceName	VARCHAR2(500)	A symbolic service name that the message refers to.	NOT NULL
XMLMessage	VARCHAR2(4000)	A message that contains all information that the named serviceName requires.	NOT NULL

Name	Time	Description	Restrictions
realTime	NUMBER(2)	Indicates whether the communication event is happening in real-time and consequently an immediate response is expected. When this flag is set, the SGA does not close the communication channel with the SGoVApp but it immediately forwards the message to the appropriate IIG and returns the result to the calling SGoVApp	NOT NULL
persistent	NUMBER(2)	Indicates whether the message should persist in case of communication errors or other abruptions and retransmitted later. If this flag is set, message is stored in the Pending Actions Queue.	NOT NULL
SGtimestamp	VARCHAR2(22)	The date and time the record was inserted in the table	NOT NULL

5.6.8 SQL Commands for creating the database tables

In this section the SQL commands for creating the PAQ database are given. The commands follow the SQL92 specification, including primary keys and referential constraints. The IT staff may need to adapt certain features to the requirements of the DBMS used in the installation, e.g. the VARCHAR2 data type should be substituted for the VARCHAR data type in an ORACLE installation. Other DBMSs pose limitations on the length of the VARCHAR columns or the overall length of a single row; these limitations should be sought after.

```
CREATE TABLE entraPAQ (
        entraPAQId              NUMBER(38) NOT NULL,
        XMLMethodDescription    VARCHAR2(4000) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        notificationName        VARCHAR2(500) NOT NULL,
        PRIMARY KEY (entraPAQId)
);
```

```
CREATE TABLE adelantePAQIIG (
        adelantePAQId           NUMBER(38) NOT NULL,
        notificationName        VARCHAR2(500) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        PRIMARY KEY (adelantePAQId)
);
```

```
CREATE TABLE adelantePAQ (
        adelantePAQId           NUMBER(38) NOT NULL,
        requestId               NUMBER(38) NOT NULL,
        serviceName             VARCHAR2(500) NOT NULL,
        XMLPacket               VARCHAR2(4000) NOT NULL,
        realTime                NUMBER(2) NOT NULL,
        persistent              NUMBER(2) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        PRIMARY KEY (adelantePAQId)
);
```

```
CREATE TABLE entraPAQIIG (
        entraPAQId              NUMBER(38) NOT NULL,
        serviceName             VARCHAR2(500) NOT NULL,
        XMLMessage              VARCHAR2(4000) NOT NULL,
        realTime                NUMBER(2) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        PRIMARY KEY (entraPAQId)
);
```

```
CREATE TABLE SEPdatabaseTable (
        SEPdatabaseTableId      NUMBER(38) NOT NULL,
        serviceName             VARCHAR2(500) NOT NULL,
        XMLMessage              VARCHAR2(4000) NOT NULL,
        realTime                NUMBER(2) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        PRIMARY KEY (SEPdatabaseTableId)
);
```

```
CREATE TABLE databaseTable (
        databaseTableId         NUMBER(38) NOT NULL,
        requestId               NUMBER(38) NOT NULL,
        serviceName             VARCHAR2(500) NOT NULL,
        XMLMessage              VARCHAR2(4000) NOT NULL,
        realTime                NUMBER(2) NOT NULL,
        persistent              NUMBER(2) NOT NULL,
        SGtimestamp             VARCHAR2(22) NOT NULL,
        PRIMARY KEY (databaseTableId)
);
```

```
CREATE TABLE autoKeys (
        KeyName                 VARCHAR2(32) NOT NULL,
        KeyValue                NUMBER(38) NOT NULL,
        PRIMARY KEY (keyName)
 );
```

5.7 SmartGov System Services

The SmartGov platform incorporates a number of back-end services that need to be available for the service delivery platform to operate successfully. These services are:

1. *Document storage and retrieval.* The document storage service allows for storing the documents submitted by users of the SmartGov services delivered through the service delivery platform. The documents are stored in an XML repository. The document retrieval service allows for retrieving documents that have been submitted by users of the SmartGov services and stored into the XML repository. Document retrieval may be used for presenting the documents to the users, forwarding them to the organisational back-end etc.
2. *Login validation.* Users of the SmartGov service delivery platform must authenticate themselves to the system, in order to provide a secure and personalised environment. Authentication is performed by means of entering a user name and a password, and the login validation service arranges for verifying that the presented credentials are valid. For authorised users, the login validation service returns to the service delivery environment the list of services that the user is registered to.

The SmartGov communication services configuration files included in the bundle include the necessary configuration entries for these services. These entries should not be removed; changes however should be applied to reflect local settings (IP addresses and ports, file system paths, passwords etc). In the following paragraphs, the SmartGov System Services are documented.

5.7.1 Document Storage and Retrieval Services

The document storage service arranges for the persistent storage of the documents submitted through the Service Delivery Environment. The persistent storage provider is the SmartGov Project XML repository. Documents that have been persistently stored may then be retrieved through the document retrieval service for presentation to the users or further processing. No provision is made at this level for document deletion or update, facilitating the maintenance of a complete track of submitted documents: in the case that document deletion is required, the documents should be *marked as deleted* (and not physically removed), while for document updating a *new version* of the document should be created superseding the existing one. Physical update or deletion of documents can still be performed by directly accessing the XML repository.

The document storage and retrieval services must be named `storeDocument` `retrieveDocument`, respectively, and must be declared in the relevant configuration files, both in the service delivery environment and the organisational back-end. Since the document storage and retrieval services use the XML repository as a persistent storage provider, they must be able to locate the configuration settings for the XML repository; these settings should be provided in a property file and the location of this file should be designated through the property `docStore.propertyFile`. Documents stored and retrieved through these services must follow the XML schema presented in the following figure:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:complexType name="TSE">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="value" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="RowType">
        <xs:sequence>
            <xs:element name="tseElement" type="TSE" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="groupElement" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                <xs:element name="row" type="RowType" maxOccurs="unbounded"/>
                      </xs:sequence>
                      <xs:attribute name="groupId" type="xs:string"/>
                    </xs:complexType>
                </xs:element>
        </xs:sequence>
    </xs:complexType>


    <xs:element name="ServiceResults">
         <xs:complexType>
            <xs:sequence>
                <xs:element name="serviceName" type="xs:string"></xs:element>
                <xs:element name="userName" type="xs:string"/>
                <xs:element name="timestamp" type="xs:string"/>
                <xs:element name="row" type="RowType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 77 - XML schema for XML documents managed through the storage and retrieval services

The document storage service accepts as a parameter the XML document that must be stored, and places the document into the persistent storage. The document retrieval service accepts as a parameter an XML document specifying which documents are requested to be retrieved. The specification is made as follows:

1. If the document contains only a `<userName>` element, all documents submitted by the user designated by the element value are retrieved and returned.
2. If the document contains a `<userName>` element *and* a `<serviceName>` element, then all documents submitted by the user and through the specific service designated by the element values are retrieved and returned.
3. If the document contains a `<userName>` element, a `<serviceName>` element *and* a `<timestamp>` element, then a single document is retrieved, the one submitted by the specific user and through the particular service at the given timestamp.
4. If the document contains a `<userName>` element, a `<serviceName>` element a `<timestamp>` element *and* a `<createIfNotFound>` element whose value is set to `true`, then the following actions are taken:
    a. The XML repository is queried to locate a document whose `<userName>`, `<serviceName>` and `<timestamp>` elements have values equal to the ones specified in the document supplied as a parameter. If such a document is found, it is retrieved and returned as in case (3) and the procedure terminates; otherwise step (b) is performed.
    b. A Java class named `IIGCreateServiceNameDocument` is searched for, where the *ServiceName* portion of the class name is set to the actual name of the service as specified in the `<serviceName>` element; for instance, if the value of the service name is TaxReturnForm, then the name of the class that will be searched for is `IIGCreateTaxReturnFormDocument`. The class must implement a constructor with no parameters and a `createDocument` method with the following prototype:

    ```
    String createDocument(String userName, String serviceName, String timestamp);
    ```

    If the class is found and it fulfils the aforementioned requirements, then the `createDocument` method is invoked to create the document with the pre-populated fields, which must be adherent to the XML schema depicted in Figure 77; the values for the pre-populated fields may be retrieved from files, from database registries or from any other appropriate source. If the `createDocument` method throws an exception or the class `IIGCreateServiceNameDocument` is not found or if the class

does not implement the required methods, then the returned document will only contain values for the `<userName>`, `<serviceName>` and `<timestamp>` elements. The implementation of the `IIGCreateServiceNameDocument` class must be provided by the organisation's IT staff. The file containing the implementation should be included in the CLASSPATH of the IIG.

In the first two cases, the results are returned in an XML document compliant to the XML schema depicted in Figure 78.

```
<?xml version="1.0" encoding="utf-8"?>
<results>
        <xmlDocument>… </xmlDocument>
        <xmlDocument>… </xmlDocument>
        <xmlDocument>… </xmlDocument>
</results>
```

Figure 78 – XML schema for calls returning multiple documents

In cases (3) and four, the XML document returned is compliant to the XML schema presented in Figure 77. For case (3) in particular, if no matching document is found then the empty string is returned.

The `storeDocument` method, besides storing the document into the XML document repository, provides facilities for storing to alternate registries, populating relational databases etc. More specifically, after storing the document into the XML document repository, the IIG searches for a class named `IIGStoreServiceNameDocument`, where the *ServiceName* portion of the class name is set to the actual name of the service as specified in the `<serviceName>` element; for instance, if the value of the service name is TaxReturnForm, then the name of the class that will be searched for is `IIGStoreTaxReturnFormDocument`. The class must implement a constructor with no parameters and a `storeDocument` method with the following prototype:

```
void storeDocument(IIGServiceResults XMLdocument);
```

If the class is found and it fulfils the aforementioned requirements, then the `storeDocument` method is invoked to perform any data storage, registry population triggering or any other activities pertinent to the submission of documents of the specific types. If the class `IIGStoreServiceNameDocument` is not found or if the class does not implement the required methods, no action is taken. If any exception occurs within the execution of the `storeDocument` method, it is possible that side effects will

occur, such as partially updated files or incomplete database population; the implementation of the `storeDocument` method must include safeguards to provide resilience against such situations. The `IIGServiceResults` class is used for storing a flattened description of the XML document with convenience methods to retrieve values of specific TSEs, along with facilities to traverse the whole TSE set. The documentation for the `IIGServiceResults` class is provided in the following sections.

The implementation of the `IIGStoreServiceNameDocument` class must be provided by the organisation's IT staff. The file containing the implementation should be included in the CLASSPATH of the IIG.

5.7.1.1 Preparing the Document Storage and Retrieval Service

Before the document storage and retrieval services can be used, the following steps must be taken:

1. The libraries `jaxp1.1.jar`, `jdbc2.0-stdext.jar`, `regexp.jar`, `xalan.jar`, `xerces.jar`, `xmlstoreapi-2.0.0.jar`, and `xmlstore-2.0.0.jar` must be included in the class path. If Java(tm) SDK 1.4.x is used, then the libraries `jaxp1.1.jar`, `jdbc2.0-stdext.jar` and `xerces.jar` are optional.
2. A database and the appropriate user must be created in the RDBMS on top of which the XML repository will operate.
3. The XML store configuration application should be run, by executing the java `com.archetypon.xml.store.impl.XmlStoreManager` class. Using the XML store configuration the following tasks should be performed:
    a. A new document type named `serviceResults` must be created.
    b. For the newly created document type, the following indexes must be defined:

Index name	XPath expression	Value type
`ServResUsername`	`/serviceResults/userName/text()`	`java.lang.String`
`ServResServiceName`	`/serviceResults/serviceName/text()`	`java.lang.String`
`ServResTimestamp`	`/serviceResults/timestamp/text()`	`java.lang.String`

4. The configuration must be saved. During this step, the details for communicating with the RDBMS should be entered in the dialog box that will appear, to correctly reflect underlying DBMS, type, the JDBC driver to be used, the server name, user name, password and database.

The document storage and retrieval services are now fully prepared to be used.

5.7.1.2 JavaDoc for the Document Storage and Retrieval Service

The JavaDoc for the storage and retrieval services is presented in Figure 79.

Constructors **public DocStore()**

Methods **public static void storeDocument(String servDesc)**

This method stores an XML document submitted through a SmartGov service in the XML repository. The XML document is contained in the String servDesc. If the IIGStoreServiceNameDocument class exists and it implements the method

void storeDocument(String XMLDocument);

then this method is invoked.

Parameters

servDesc - The String containing the XML description.

Throws

DocStoreException - In case of failure to store the XML document

public static java.lang.String retrieveDocument(String docDesc)

This method retrieves the document(s) containing the elements given in the XML description.

Parameters

docDesc - The XML document specifying the documents to be retrieved. The specification is made using one the following four methods:

1. <?xml version="1.0" encoding="utf-8"?>
   <userName>my user</userName>
2. <?xml version="1.0" encoding="utf-8"?>
   <userName>my user</userName>
   <serviceName>my user</serviceName>
3. <?xml version="1.0" encoding="utf-8"?>
   <userName>my user</userName>
   <serviceName>my service</serviceName>
   <timestamp>2002/12/31 12:35:32</timestamp>
4. <?xml version="1.0" encoding="utf-8"?>
   <userName>my user</userName>
   <serviceName>my service</serviceName>
   <timestamp>2002/12/31 12:35:32</timestamp>
   <createIfNotFound>true</createIfNotFound>

Returns

A String where the results of the search are located.

If username, serviceName and timestamp are given (case 3), the result is the

requested XML document. Case 4 is an extension of case 3, allowing for creation of initial documents with pre-populated fields, by invoking the method createDocument of the class IIGCreateServiceNameDocument (ServiceName is equal to the value of the serviceName element). The createDocument method prototype is as follows:
String createDocument(String userName, String serviceName, String timestamp);
If not all the elements are provided (cases 1 and 2), the result may contain more than one documents, which are grouped in an XML document of the following type:

```
<?xml version="1.0" encoding="utf-8"?>
<results>
        <xmlDocument>… </xmlDocument>
        <xmlDocument>… </xmlDocument>
        <xmlDocument>… </xmlDocument>
</results>
```

The value of each xmlDocument element is a URL-encoded XML document.

Throws

DocStoreException - In case of failure to retrieve the document(s).

Figure 79 - JavaDoc for document storage and retrieval services

5.7.1.3 JavaDoc for the IIGServiceResults

public class **gr.uoa.di.IIGServiceResults.IIGServiceResults**

The IIGServiceResults class represents a flattened XML service results document for interfacing with legacy systems. API is provided both for construction and querying

Constructors **public IIGServiceResults()**

Creates a new instance of IIGServiceResults

Methods **public gr.uoa.di.IIGServiceResults.IIGServiceResults setServiceName(**
String serviceName)

This method sets the serviceName element of the flattened representation of the XML document.

Parameters

serviceName - the value of the serviceName element of the XML document

Returns

the updated flattened representation of the XML document

public gr.uoa.di.IIGServiceResults.IIGServiceResults setUserName(

String userName)

This method sets the userName element of the flattened representation of the XML document.

Parameters

userName - the value of the userName element of the XML document

Returns

the updated flattened representation of the XML document

public gr.uoa.di.IIGServiceResults.IIGServiceResults setTimestamp(

String timestamp)

This method sets the timestamp element of the flattened representation of the XML document.

Parameters

timestamp - the value of the timestamp element of the XML document

Returns

the updated flattened representation of the XML document

public java.lang.String getUserName()

This method queries the userName element of the respective XML document

Returns

the value of the userName element

public java.lang.String getServiceName()

This method queries the serviceName element of the respective XML document

Returns

the value of the serviceName element

public java.lang.String getTimestamp()

This method returns the timestamp element of the respective XML document

Returns

the value of the timestamp element

public gr.uoa.di.IIGServiceResults.IIGServiceResults addElement(

Node theTSE)

This method adds a Node element that should represent a TSE not embedded into a group into the flattened document description.

Parameters

theTSE - the DOM node element to be inserted

Returns

the flattened description with the TSE appended to it

public gr.uoa.di.IIGServiceResults.IIGServiceResults addElement(

Node theTSE,

String groupName,

int groupRow)

This method adds a Node element that should represent a TSE embedded into a group into the flattened document description.

Parameters

theTSE - the DOM node element to be inserted

groupName - the group to which the TSE belongs

groupRow - the row within the group that the TSE appears

Returns

the flattened description with the TSE appended to it

public int size()

Queries the number of TSE values within the flattened description

Returns

the number of TSE values within the flattened description

public java.lang.String elementNameAt(int index)

Queries the name of the TSE whose value is stored at a specific position

Parameters

index - the position to be queried

Returns

the name of the TSE

public int elementValueIndexAt(int index)

Queries the value index of the TSE value stored at a specific position

Parameters

index - the position to be queried

Returns

the value index of the TSE value

public java.lang.String elementValueAt(int index)

Queries the value of the TSE value stored at a specific position

Parameters

index - the position to be queried

Returns

the TSE value

public java.lang.String elementGroupNameAt(int index)

Queries the group name of the TSE value stored at a specific position

Parameters

index - the position to be queried

Returns

the group to which the TSE value belongs

public int elementGroupRowAt(int index)

Queries the group row of the TSE value stored at a specific position

Parameters

index - the position to be queried

Returns

the group row of the TSE value

public int getElementIndex(String elementName)

Queries the position in which the first value of a TSE with a given name is stored

Parameters

elementName - the TSE name to search for

Returns

the index of the first TSE value; if no TSE with the designated name exists, -1 is returned

public java.lang.String getElementValue(String elementName)

Queries the first value of a TSE with a given name

Parameters

elementName - the TSE name to search for

Returns

the first TSE value; if no TSE with the designated name exists, null is returned

5.7.2 Login Validation Service

The login validation service provides the functionality for authenticating users, before they are allowed to access the services deployed through the SmartGov platform. User authentication is performed through a user name-password scheme: users enter their credentials, and the login validation service verifies that these credentials are correct. If the credentials are invalid, an appropriate failure indication is returned; if, however the presented credentials are valid, a success indication is returned, complemented with the services that the user is registered to use.

The login validation service must be named `loginValidation` and must be declared in the relevant configuration files, both in the service delivery environment and the organisational back-end. User credentials and service profiles are stored in a DBMS, thus the login validation service needs to access certain information regarding its connection with the database (software driver, server address etc). This information should be stored in a property file and the property `LoginValidation.propertyFile` must be set to point to this file. Example contents of the file are depicted in Figure 80:

```
# Property file for the login validation service
LoginValidation.DBusername=smartgov
LoginValidation.DBpassword=smartgov
LoginValidation.DBname=smartgov
LoginValidation.DBdriver=oracle.jdbc.driver.OracleDriver
LoginValidation.DBconnectString=jdbc:oracle:oci8:@
```

Figure 80 - Property file for login validation service

A request for validating the credentials supplied by the user should be made by invoking the `loginValidation` service with an XML document that must adhere to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT validationRequest (userName, password)>
<!ELEMENT userName(#PCDATA)>
<!ELEMENT password (#PCDATA)>
```

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<validationRequest>
    <userName>aUser</userName>
    <password>aPassword</password>
</validationRequest>
```

This request will check if the credentials `(aUser, aPassword)` are valid. Normally the request should be characterised as *real-time* and *non-persistent* since the authentication process must be carried out immediately. The reply returned to such a request will adhere to the following DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT validationResult (resultType, userID?, fullName?, serviceName*)>
<!ELEMENT resultType(#PCDATA)>
<!ELEMENT userID(#PCDATA)>
<!ELEMENT fullName(#PCDATA)>
<!ELEMENT serviceName(#PCDATA)>
```

In more detail, if the presented credentials are not valid, the reply will contain only the `<resultType>` element and the value of this element will be set to `failure`, as illustrated in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<validationResult>
    <resultType>failure</resultType>
</validationResult>
```

Figure 81 - Reply for a validation request presenting invalid credentials

If, however, the presented credentials are valid, the value of the `<resultType>` element will be set to `success`, and the element will be followed by the `<userId>` and `<fullName>` elements, which will in turn be followed by a list of `<serviceName>` elements, one for each service that the user is registered to use. An example of a reply to successful validation request is illustrated in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<validationResult>
    <resultType>success</resultType>
    <userID>42</userID>
    <fullName>Zaphod Beeblebrox</fullName>
    <serviceName>incomeTaxService</serviceName>
    <serviceName>VATservice</serviceName>
    <serviceName>realEstateTaxService</serviceName>
</validationResult>
```

5.7.2.1 Preparing the Login Validation Service

Before the document storage and retrieval services can be used, the following steps must be taken:

1. a database must be created in an RDBMS. A database user with privileges to create table in this database must also be created.

2. the tables SGuserData and SGuserServices must be created in the database using the following SQL commands[2]:

```
create table SGuserData (
   userID       NUMBER(12) NOT NULL,
   userName     VARCHAR(64) NOT NULL UNIQUE,
   password     VARCHAR(64) NOT NULL,
   fullName     VARCHAR(128) NOT NULL,
   PRIMARY KEY(userID)
);

create table SGuserServices (
   userID       NUMBER(12) NOT NULL REFERENCES SGuserData (userId),
   serviceName  VARCHAR(128) NOT NULL,
   PRIMARY KEY(userID, serviceName)
);
```

The population and maintenance of these tables may be performed using any appropriate tool. For MySQL installations, the phpMyAdmin tool is recommended (http://sourceforge.net/projects/phpmyadmin/). For Oracle installations, the DBA studio shipped with Oracle may be used, while in Microsoft SQL Server installations the SQL Server Enterprise Manager® can be employed to maintain table data.

[2] The commands use ANSI SQL 92 syntax. For Oracle databases it is recommended that the VARCHAR2 data type is substituted for the VARCHAR data type. For MySQL and MS SQL Server systems, the NUMBER(12) data type must be replaced by the INTEGER data type.

6 Conclusions

This deliverable presented the user manual of the front-end, result of the final iteration of the development phase of the SmartGov platform and focused on the components developed within work package 5 and 6. …

7 References

[D31] State-of-the-Art and Current Situation at Public Authorities, SmartGov Project Deliverable 31, Stelios Gorilas, George Boukis, Giorgos Lepouras, Kostas Vassilakis, Akrivi Katifori, John Fraser, Heredia Larios Segundo, Rafael Canadas Martinez, Gerald Weiss, Kirstin Karasz, Spyros Argyropoulos and Hilary Coyne (May 31, 2002) available at http://www.smartgov-project.org/index.php?category=results&langid=eng

[D41] User Requirements, Services and Platform Specifications, SmartGov Project Deliverable 41, Akrivi Katifori, Anna Charissi, George Lepouras, Stathis Rouvas, Costas Vassilakis, Nick Adams, John Fraser, Segundo Heredia Larios, George Boukis, Stelios Gorilas, Rafael Canadas Martinez and George Laskaridis, available (July 31, 2002) at http://www.smartgov-project.org/index.php?category=results&langid=eng

[D51-61] Low-level Specifications of SmartGov Services and Applications and the Knowledge-Based Core Platform, SmartGov Project Deliverable 51-61, Stelios Gorilas, Pablo Fernadez Pardo, Tomas Pariente Lobo, Costas Vassilakis, Akrivi Katifori, Anna Charissi, George Lepouras, Stathis Rouvas, Nick Adams, John Fraser, Ann Makynthos (February 28, 2003) available at http://www.smartgov-project.org/index.php?category=results&langid=eng

[D62] Implementation of SmartGov Services and Applications, SmartGov Project Deliverable 62, Stelios Gorilas, Pablo Fernadez Pardo, Tomas Pariente Lobo, Costas Vassilakis, Akrivi Katifori, Anna Charissi, George Lepouras, Nick Adams, John Fraser, Ann Makynthos, Vassilis Stoumpos, available (July 31, 2003) at http://www.smartgov-project.org/index.php?category=results&langid=eng

Appendix A. Glossary of elements

Transaction Service Elements (TSEs): Transaction Service Elements (TSEs) are considered as the basic building block of an e-service. They are used to represent basic data types used within the organization. TSEs are not to be confused with basic data types as handled by programming languages. They are not just strings, integers, floats etc: TSEs are conceptual constructs that map onto the organization's practices. A TSE represents a real-world entity and its attributes model this entity's characteristics in a self-contained manner. TSEs are defined in an XML format and could contain the following properties:

- Unique identifier
- Machine-oriented data type, e.g. integer, string, float etc.
- Data type format rules
- Presentational info, possibly according to dissemination channel, e.g. length of data, number of decimals, colour, etc.
- Interface definitions for transforming the TSE values from and to different formats and for communicating with the SmartGov agent they refer to (for exchange of data with third-party systems)
- Generic name and/or service specific aliases (or handles)
- Generic validation constraints/conditions. Service specific constraints and/or more detailed ones are considered to belong to the Knowledge Repository.

TSEs are "cloned" when the time comes to implement a new service. The properties of the cloned TSEs can be overridden with service-specific properties. These properties are expected to be specialized versions of the properties offered by the generic TSE they clone, suitable for the service they refer to.

TSE group: TSEs can be grouped arbitrarily forming TSE groups. TSE groups like TSEs are defined in an XML format and could contain the following properties:

- Unique identifier
- Presentational info, possibly according to dissemination channel
- Interface definitions for communicating with the SmartGov agent they refer to (for exchange of data with third-party systems)
- Generic name and/or service specific aliases (or handles)
- Generic validation constraints/conditions
- Active code to be executed when they are instantiated

A TSE group cannot be used as an element to form another TSE group.

TSE Repository: A repository containing all available TSEs and TSE groups. A TSE repository is needed to coordinate access and to ensure overall consistency, e.g. that referenced items are not deleted in a later stage.

Integrator: The integrator produces code to be executed in the environment of the deployment server. It gathers the respective information from the various repositories and combines them accordingly.

Transaction Service: A set of forms that implement a specific service. It contains TSEs, validation constraints, pre- and post-conditions and the layout to be used.

Transaction Service Repository: A repository containing all available Transaction Services.

Layout: Contains static attributes of presentation of a Transaction Service. These static attributes will include colour schemes, font families and sizes, border designations etc. A layout may also contain guidelines to the designer about style of placement of elements on the form, pieces of advice regarding form size, or even automated checks that scan for guideline violations.

Layout Repository: A repository containing all available Layouts.

Interaction Templates: A template that includes the basic interaction between the delivered service and the end user. The interaction template may specify:

Style of navigation between forms in multi-form services (back-next-finish, index form with references to individual forms etc.)

Use of confirmation screens (every form, once before final submittal, never)

Style of error presentations (separate window, within the form etc)

Interaction Templates Repository: A repository containing all available Interaction Templates.

Process Management Information: Information about flow control, data control/integrity constraints, user profiles/roles and preconditions/postconditions.

Knowledge Repository:

The knowledge repository will contain all the knowledge items (examples, best practices, legislation, guidelines) along with structures facilitating access to it, such as knowledge maps. Items within the knowledge repository may reference or be referenced from items in other repositories; e.g. an example on filling in a TSE may reference the specific TSE and/or be referenced from it.

Appendix B. Validation Rules User's Guide and Reference

There are two kinds of rules: *compact rules* and *full rules*. Although compact rules cannot describe any possible full rule, they serve as shortcuts for the most common rules. All rules are attached to the element they refer to. For example a rule that checks the validity of a "total income" element is attached to that element. Rules are also allowed to reference objects "contained" in the element they are attached to; thus

1. rules attached to TSE groups may reference the TSEs within the group
2. rules attached to forms may reference TSEs and TSE groups contained within the form
3. rules attached to the service may reference any TSE or TSE group within the service.

B.1 Attaching validation rules to SmartGov entities

For SmartGov platform entities that validation rules may be associated with (transaction services, forms, TSE groups and TSEs), the user should navigate to the relevant editing page and open the *Validation Rules* section. An example for validation rules attached to forms is illustrated in Figure 82:

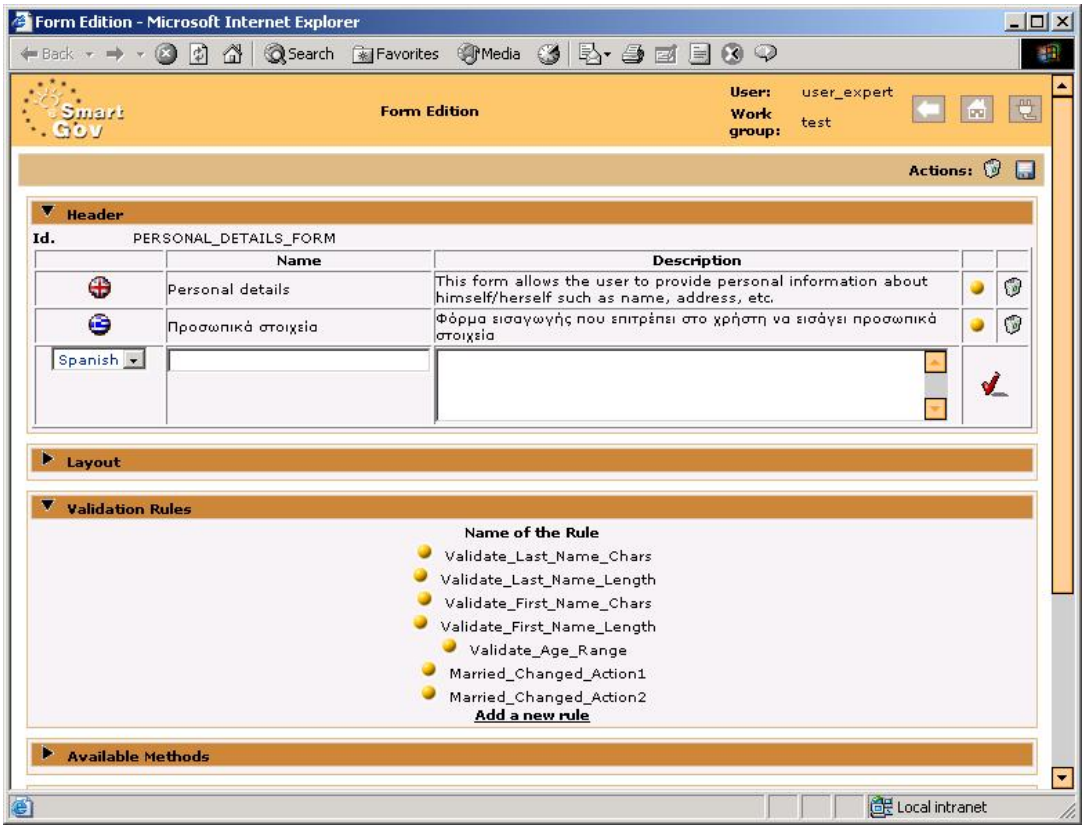


Figure 82 – The Validation Rules section in a form

The validation rules section enables the user to add new validation rules or modify existing validation rules. In order to add a validation rule, the user should click on the *Add a new rule* hyperlink, whereas for modifying an existing validation rule the user should click on the bullet appearing at the left of the rule name in the *"name of the rule"* list.

B.2 Working with validation rules

Once the "add a new rule" has been selected or a rule name has been clicked on, the rule-editing page appears. If a new rule is being added, the page fields will be blank, otherwise they will be pre-populated with the current rule details. In the rule-editing page (shown in Figure 83), the following information should be filled in:

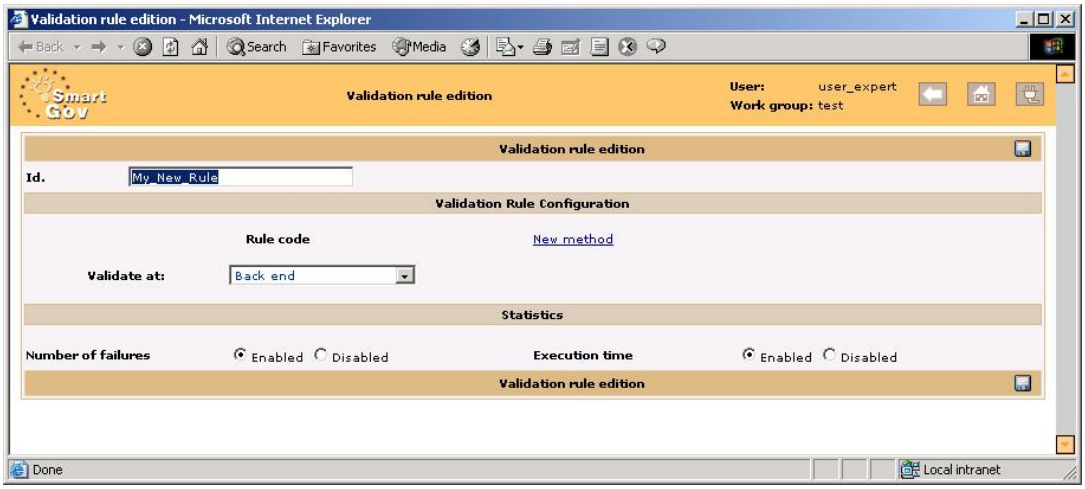


Figure 83 - Rule editing page

1. the identity of the new rule (id field)
2. the specification whether the rule should be validated at the back-end only or at the front-end *and* the back-end (there is no option to validate the rule only at the front-end)
3. the statistics to be collected for rule execution. Two pre-defined statistics are supported, namely the total number of failures for the rule and the rule execution time. For either statistic the user can enable or disable statistics collections.
4. Finally, the actual content of the validation rule can be specified. For new rules, the "*new method"* hyperlink should be clicked on, whereas for existing rules the description in the "validation rule configuration" section should be clicked on. Validation rule method configuration is described in the following section.

B.3 Validation rule method configuration

The main editing page for validation rule methods is depicted in Figure 84. The user first enters the description of the rule; this is a multilingual resource and may be provided in multiple languages. After entering the rule description, the user should select the type of the code implementing the validation check. This may be code in some native language (Java or Javascript), or code in the SmartGovLang validation language; the latter is subdivided in two categories, namely SmartGov full rules and SmartGov compact rules. Details for each method category are provided in the following paragraphs.

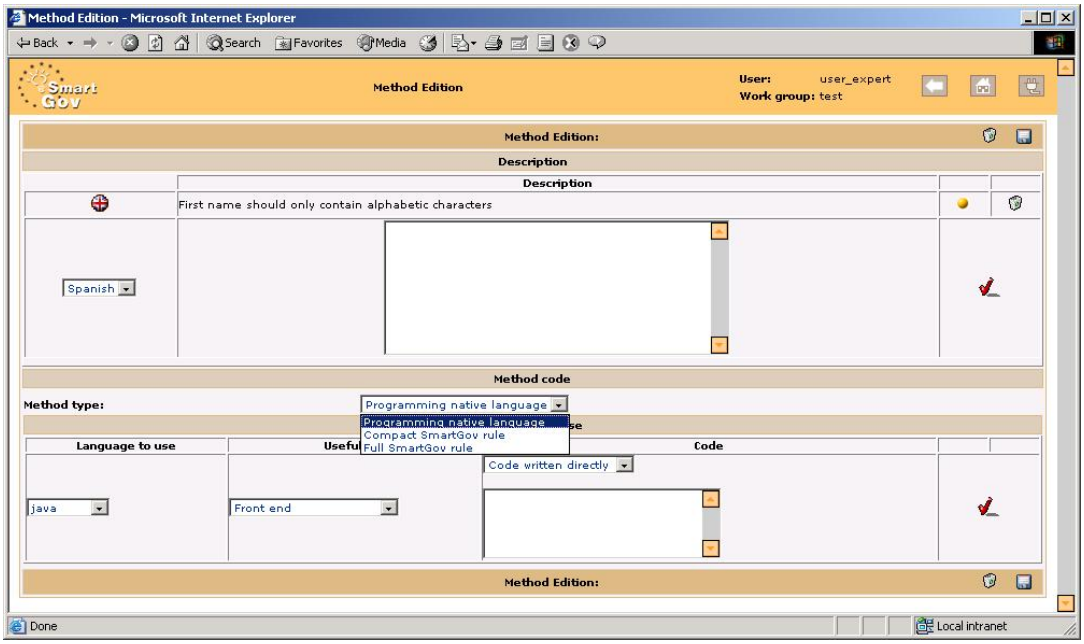


Figure 84 – Validation rule main editing page

B.3.1 Native language validation checks

For a validation check implemented in some native language, the coding language can be entered (Java and Javascript are the available options), an indication on whether the code can implement checks for the front-end, back-end or both and the actual code fragment implementing the validation, either by directly typing it in a relevant area or by uploading a file with the code. Note that the Integrator does not take into account validation checks coded in any native language; the IT staff (which will write the code) should cater for the integration of code into the produced services by uncompressing the produced *war* file and modifying the files therein.

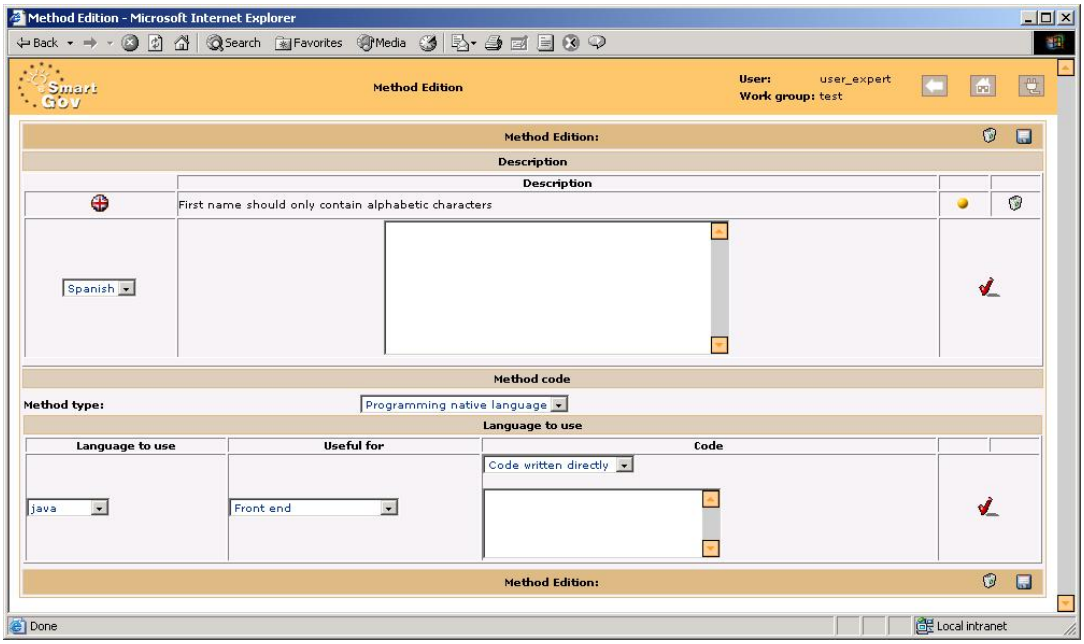


Figure 85 - Entering validation methods in native language

B.3.2 SmartGov language compact rules

Compact rule methods can be used to express simple validation checks in a user-friendly and intuitive manner. Besides the "description" part, which is common for all rule method types, each compact rule has the following elements:

- a *severity designation*, which classifies the validation check either as an *error* or as a *warning*. Errors must be corrected before the user continues to the next stage of the e-service, while a warning is an advisory message to the user. Experience however has shown that users tend to ignore warning messages so it strongly recommended that only validation checks with *error* severity designations are used.
- a *validation message* which is the message that will be displayed to the user of the electronic service, in the case that the validation check fails. This is a multilingual resource and should be provided in the languages that the e-service target group uses.
- the *type* of the compact rule. Six pre-defined compact rule types are provided:
  - *field between values of two fields,* illustrated in Figure 86. In this case the developer selects the TSE to be checked from a drop down list, and enters the lower and upper bound in the two adjacent input areas. The e-service user should provide for the selected TSE a value between the designated limits.

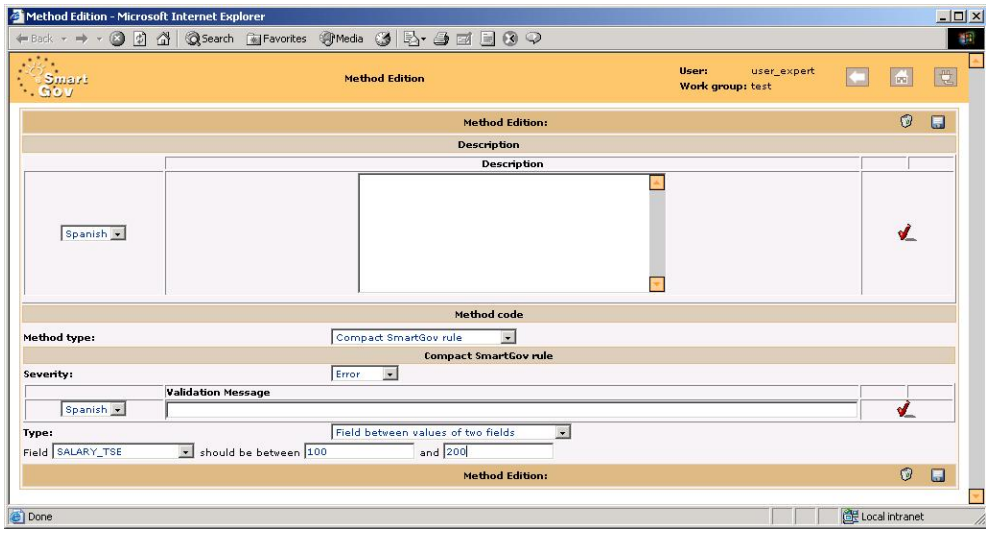


Figure 86 – Compact SmartGovLang validation method editing

- *field requires other field,* illustrated in Figure 87. In this case the developer selects two TSEs from respective drop-down lists. If the e-service user provides a value for the TSE selected in the first drop-down list, then she must provide a value for the TSE selected in the second drop-down as well.

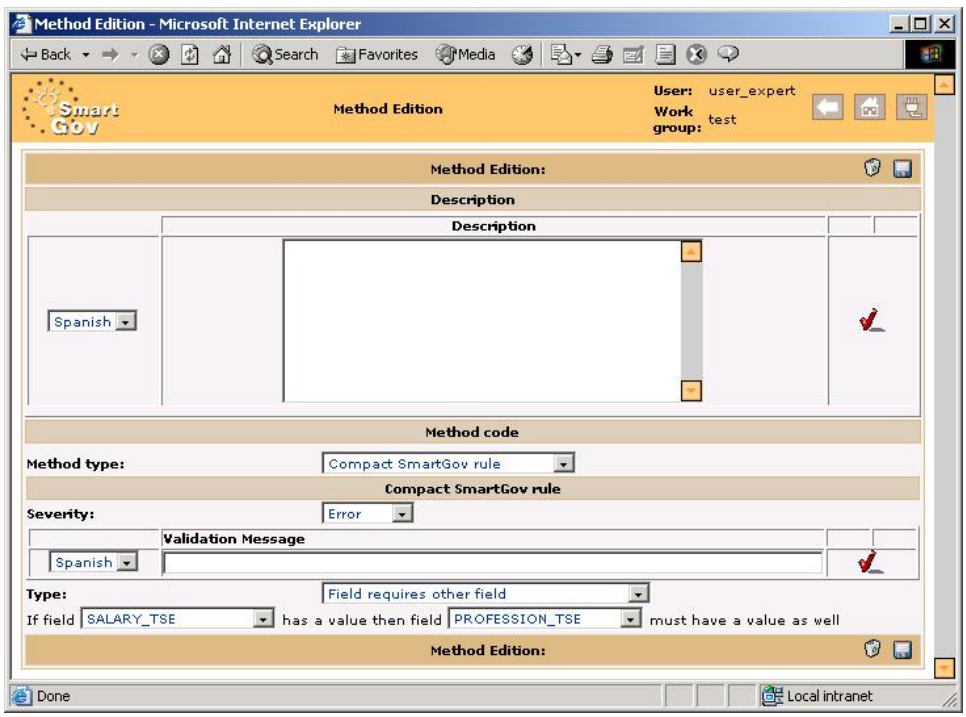


Figure 87 – Compact SmartGovLang validation method editing

- *field precludes other field,* illustrated in Figure 88. In this case the developer selects two TSEs from respective drop-down lists. If the e-service user provides a value for the TSE selected in the first drop-down list, then she *must not* provide a value for the TSE selected in the second drop-down.

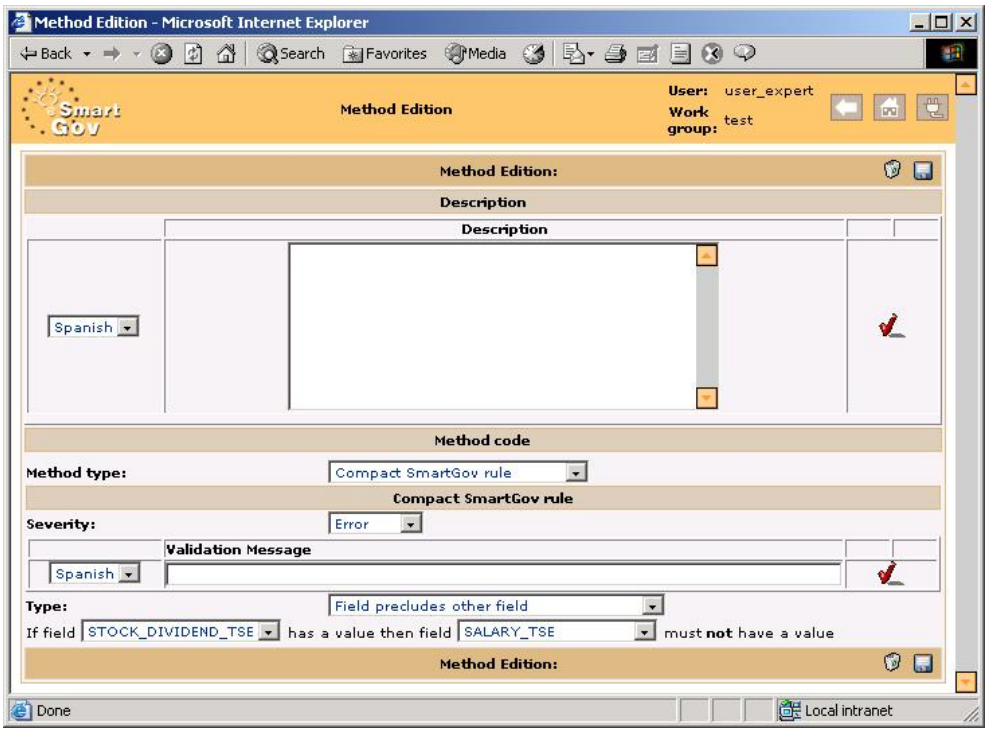


Figure 88 – Compact SmartGovLang validation method editing

- *field requires other fields,* illustrated in Figure 89. In this case the developer selects one TSEs from a drop-down lists and one or more TSEs from a second list. If the e-service user provides a value for the TSE selected in the first drop-down list, then she must provide a value for *at least one* of the TSEs chosen in the second list.

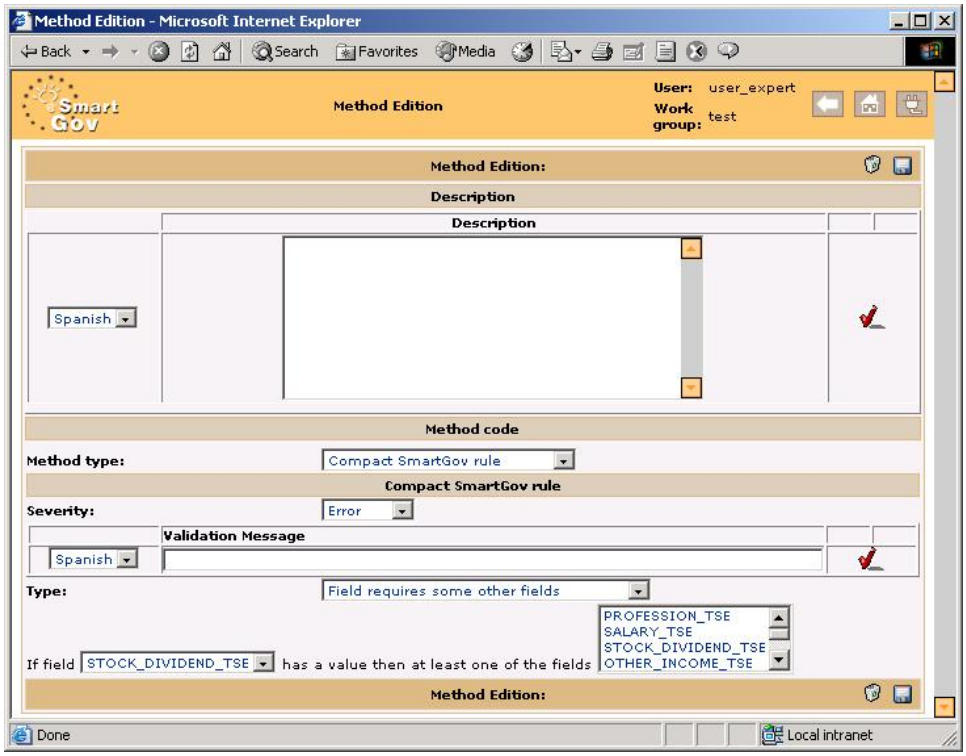


Figure 89 – Compact SmartGovLang validation method editing

- *several fields required at the same time,* illustrated in Figure 90. In this case the developer selects a list of TSEs. The e-service user should either provide a value for *all* of the selected TSEs or for none of them. In other words, it is not allowed for the e-service user to provide values for *some of the TSEs* and *not* provide values for the remaining ones.

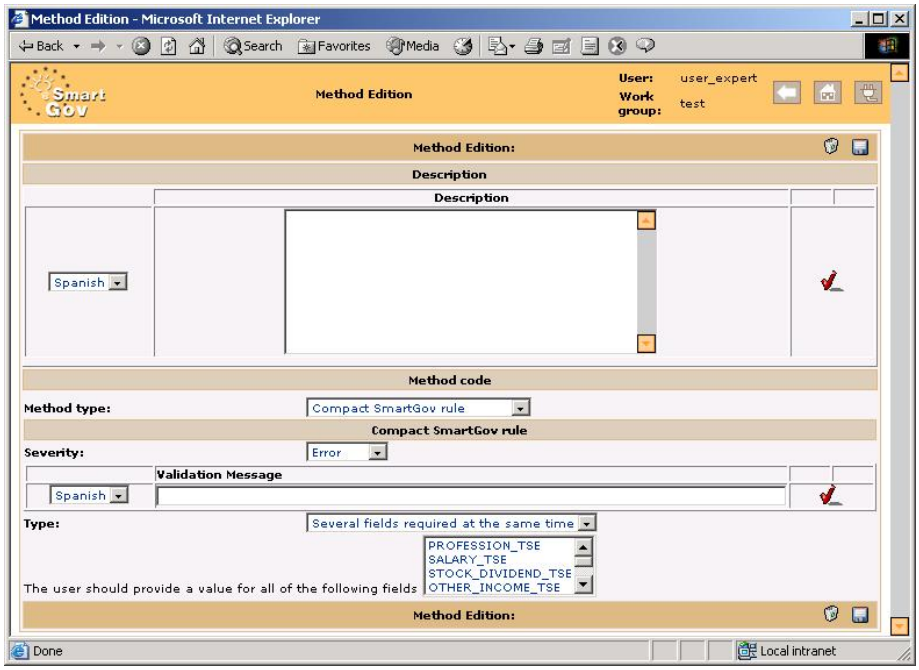


Figure 90 – Compact SmartGovLang validation method editing

- *a condition holds for a field,* illustrated in Figure 91. In this case the developer selects a TSE *tse1* from a drop-down list, a relational operator *rop* (=, ≠, >, >=, <, <=) then a second field *tse2* and finally enters a factor *fact*. The values entered by the e-service user for the referenced TSEs must satisfy the condition *tse1 rop tse2 * fact*; for instance a tax payer's net income may not be less than the 80% of her salaries, which can be expressed by *NET_INCOME_TSE >= SALARY_TSE * 0.8*.

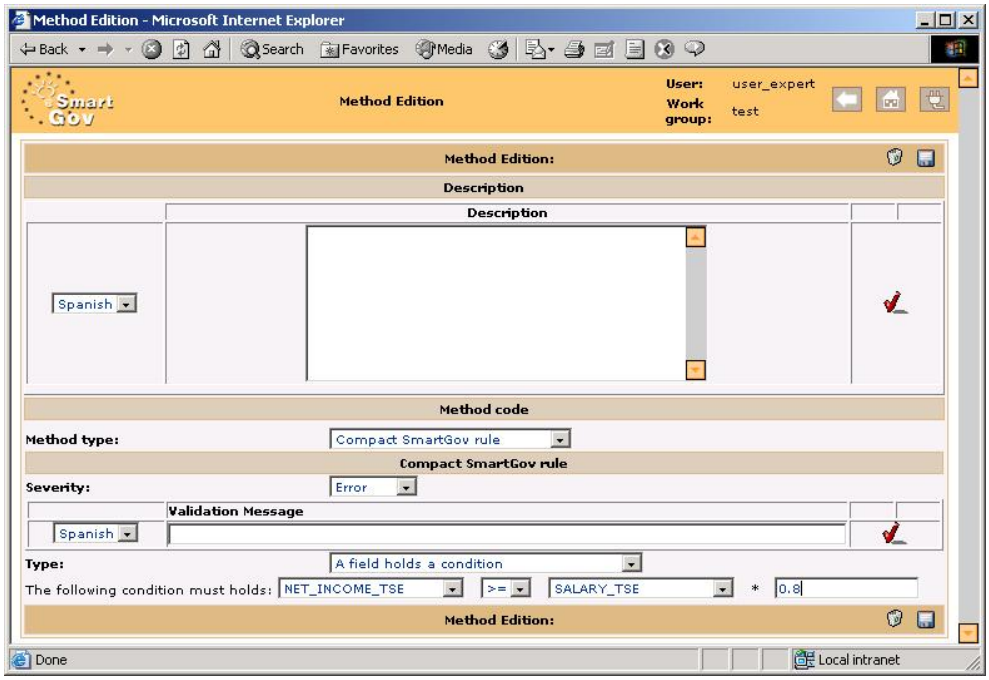


Figure 91 – Compact SmartGovLang validation method editing

B.3.3 Full Rules

Full rules are a more powerful means of coding validation checks, allowing for a plethora of tests to be made over the data values and providing facilities for active behaviour such as automatic value calculation. Full rules have a *condition part* and an *action part*, which may be entered in the respective areas of the validation rule editing form, when the method type has been set to *Full SmartGov rule* (see Figure 92). For a full compact rule, the SmartGov platform evaluates the condition part and then inspects the result: if this is *true* then the actions specified in the *action* part are taken; if the condition evaluates to *false*, then no action is taken. In the following, a rule having a condition part *condition* and an action part *action* will be denoted as

condition → action

The condition and action parts of a SmartGovlang full rule are discussed in the following paragraphs, and examples are given.

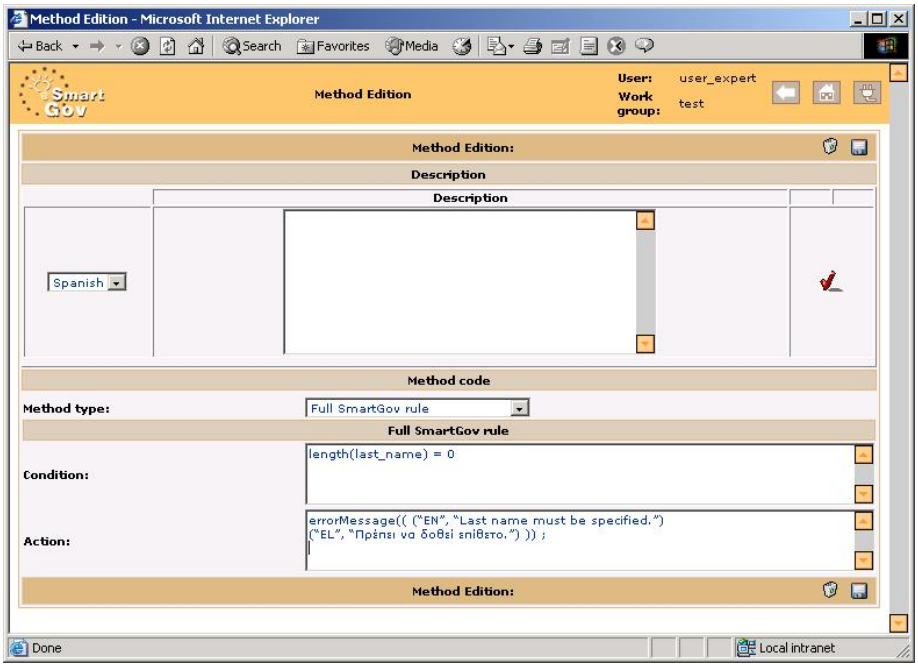


Figure 92 – Full SmartGovLang rule editing

B.3.3.1 Condition Part – Data types

Any expression that can evaluate to *true* or *false* can be used as a condition. So the literal:

true

Could be used as a condition that would always lead to the execution of the associated actions. For the majority of cases though we need to access user input on order to determine if action needs to be taken. To access the user input, we merely use the corresponding element name. For example, assuming an element "married" the expression:

married

evaluates to true or false, depending on user input. All user input elements are of type *number*, *text* or b*oolean,* depending on how each user input element is designed. The married element in the last example must be of type boolean for the condition to be valid.

Assuming non-boolean elements, valid conditions are relational expressions among them. Operators can be the ones listed in Table 2 and an example is:

total_income < total_expenses

which leads to the execution of the action part if the specified total income is less than the total expenses, for the appropriate "total_income" and "total_expenses" numeric elements. Another example is:

employment_status = "unemployed"

where the actions are executed if the "employment_status" element (could be a combo box of 4 values) has the value "unemployed". Finally, boolean expressions can be combined to form more complex expressions, as the following one:

married AND (total_income < total_expenses)

which uses the previous expressions to form a new one that evaluates to true only if both sub-expressions evaluate to true. The full list of boolean operators is presented in Table 3.

Note the use of parenthesis that specifies that the numeric comparison is the left-hand-side expression. Omitting the parenthesis would lead to "total_income" being mistakenly used as a boolean element and form the left-hand-side expression on its own.

B.3.3.2 Condition Part – Functions

To form even more complex conditions and process user input functions are provided in SmartgovLang. We can logically divide functions in four types: *arithmetic*, *string*, *date* and *aggregate*. Arithmetic functions are listed in

Table 4. An example condition that checks for integer user input is:

fractional(numof_children) != 0.0

where the action part is triggered if the user specified a non-integer number of children.

One of the most common functions used are string functions that operate on text elements.

Table 5 shows a list of al available string functions. A typical example checks for no-user input in a text field:

length(last_name) = 0

which if true means the user entered no last name.

Almost all functions can be written by means of the *matches* functions. This function uses a regular expression to test for certain string format. Describing regular expressions is beyond the scope of this document. Regular expressions are described

in [Sun][3]. Instead of directly using regular expressions, naive users should resort to using the rest of the functions:

isAlpha(last_name) AND

(startsWith(title, "Dr.") OR startsWith(title, "Doctor"))

This condition will evaluate to true if the user specified last name consists of only letters and the user title either starts with "Dr." or "Doctor", capturing the case of the user being a doctor.

Another common input check has to do with date manipulation, especially validation. The condition

NOT isValidDate(birth_date)

will trigger the execution of rules if the specified birth date is illegal. The complete list for date and time functions is given in

Table 6. Note that date and time are always expected to be of the form "yyyy/mm/dd" and "hh:mm:ss" respectively. A more complex scheme can be employed if isValidDate(yyyy, dd, ss) and isValidTime(hh, mm, ss) are used instead of the single argument ones.

Finally, aggregate functions (

Table 7) like min, max, count and sum can be used to compute the aggregate over all elements in a given form.

B.3.3.3 Action List

An action list is a sequence of actions, with each action being terminated by a semi-colon (;). An action can either be a *message action* or a *field action.* We will visit available actions them in this order.

The most common action to be taken is to present an error message to the user upon invalid input. The message can be an *error message*, a *warning message* or an *information message*, so actions in

Table 8 are used to distinguish between the cases. Different types of messages lead to different system behavior. For example, an error message will not allow the form to be submitted, while an information message will.

[3] Note that the interpretation of regular expression depends on the execution engine, which is Java for the back-end and Javascript for the front-end, thus some incompatibilities in execution may arise. Consult [Sun] and [Javascript] for a thorough coverage of the interpretations.

An important aspect of the message actions is the use of a multilingual message. A multilingual message contains the same message in different languages. For example, suppose message "sample message" is rewritten in languages "lang1", "lang2" and "lang4" as "sample message 1", "sample message 2" and "sample message 4". Producing the corresponding warning message is done by:

warningMessage((("lang1", "sample message 1")
("lang2", "sample message 2") ("lang3", "sample message 3")))

Notice that all messages are enclosed in parenthesis and every language-dependent entry is also enclosed in parenthesis. Every entry contains the language identifier and the translated message separated by commas. Thus, an error message that could be triggered by the "length(last_name) = 0" condition seen before is:

errorMessage((("EN", "Last name must be specified.")
("EL", "Πρέπει να δοθεί επίθετο."))) ;

The second type of actions is field actions. A field can be enabled or disabled (typically due to some value in another field), take a new value or take the user input focus. All these actions are presented in Table 9. Recalling the simple example condition "married", it could be necessary that field "spouse_last_name" is enabled or disabled. Thus, both rules should be added to the enclosing form:

married → enableField(spouse_last_name)
setField(spouse_last_name, "") setFocus(spouse_last_name) ;

and

NOT married → disableField(spouse_last_name) ;

In this manner we enable or disable the "spouse_last_name" dependent on the user specifying him as single or married. Notice that when the user is married, the "spouse_last_name" field is enabled, cleared with the empty string and has the user input focus.

B.4 Reference Tables

Table 2 - Operators and their meaning for operands of type *number* and *text*.

Operator	Example	Number Operands	Text Operands
=	E1 = E2	True for E1 equal to E2, otherwise false.	True for E1 same as E2, otherwise false.
!=	E1 != E2	False for E1 equal to E2, otherwise true.	False for E1 same as E2, otherwise true.

Operator	Example	Number Operands	Text Operands
>	E1 > E2	True for E1 greater than E2, otherwise false.	True for E1 lexicographically after E2, otherwise false.
>=	E1 >= E2	True for E1 greater or equal to E2, otherwise false.	True for E1 lexicographically after or same as E2, otherwise false.
<	E1 < E2	True for E1 less than E2, otherwise false.	True for E1 lexicographically before E2, otherwise false.
<=	E1 <= E2	True for E1 less or equal to E2, otherwise false.	True for E1 lexicographically before or same as E2, otherwise false.

Table 3 Boolean operators.

Operator	Example	Evaluation
AND	E1 AND E2	True for both E1 and E2 are true, otherwise false.
OR	E1 OR E2	True if at least on of E1 and E2 is true, otherwise false.
NOT	E1 NOT E2	True only if E1 is false, otherwise false.

Table 4 Arithmetic functions.

Function	Evaluation
int(x)	The integral part of value *x*. The expected data type is numeric.
fractional(x)	The fractional part of value *x*. The expected data type is numeric.

Table 5 String functions.

Function	Evaluation
length(string)	The number of characters in *string*.
index(string, token)	Returns the position, in characters, numbering from 1, in *string* where *token* first occurs, or zero if it does not occur at all.
substr(string, offset, n)	Returns the at most n-character substring of *string* that begins at position *offset*, numbering from 1.
concatenate(string1, string2)	Returns a string whose value is *string1*, followed by *string2*.
hasAlphabet(string, alphabet)	Returns true if the *string* contains only characters listed in *alphabet*, or false, otherwise
matches(string, *expression*)	Returns true if *string* matches the regular expression pattern specified in *pattern* or false otherwise. [reg exp specs]
isAlphaNumeric(string)	Returns true if *string* consists of only letters, spaces, numbers and underscores ("_"), otherwise false.
IsAlpha(string)	Returns true if *string* consists of only letters, spaces and underscores ("_"), otherwise false.
isNumeric(string)	Returns true if *string* consists of only numbers, otherwise false.
startsWith(string, prefix)	Equivalent to the following expression: (substr(string, 1, length(prefix)) = prefix)
endsWith(string, postfix)	Equivalent to the following expression: (substr(string, length(s)-length(postfix), length(prefix)) = postfix)

Table 6 Date functions.

Function	Evaluation
date()	Returns the current system date in a string of the form "yyyy/mm/dd".

Function	Evaluation
isValidDate(yyyy, mm, dd)	Returns true if the given date is valid, otherwise false.
isValidDate(string)	Returns true if the given date is of the form "yyyy/mm/dd" and valid, otherwise false.
year()	Returns the current system year.
month()	Returns the current system month.
day()	Returns the current system day in the month.
weekDay()	Returns the current system week day (1 for Sunday, 2 for Monday, …, 7 for Saturday).
time()	Returns the current system time in a string of the form "hh:mm:ss".
isValidTime(hh, mm, ss)	Returns true if the given time is valid, otherwise false.
isValidTime(string)	Returns true if the given time is of the form "hh:mm:ss" and valid, otherwise false.
hour()	Returns the current system hour.
minute()	Returns the current system minute.
second()	Returns the current system second.

Table 7 Aggregate functions.

Function	Comments
count(group, element)	Returns the number of rows in a repeating group (table) html table? Server side checking?
sum(group, element)	Returns the sum of the values in the designated column.
max(group, element)	Returns the maximum of the values in the designated column.
min(group, element)	Returns the minimum of the values in the designated column

Table 8 Message actions.

Action	Behaviour
errorMessage(multilingual messsage)	Uses the current locale to produce the appropriate error message. The operation halts.
warningMessage(multilingual messsage)	Uses the current locale to produce the appropriate warning message. The user is asked whether the operation should continue.
informationMessage(multilingual messsage)	Uses the current locale to produce the appropriate error message. The operation continues normally.

Table 9 Field actions.

Action	Behavior
DisableField(field)	Disallows the user to enter values to the designated form field.
EnableField(field)	Allows the user to enter values to the designated field.
setField(field, value)	Sets the designated field to the specified value.
SetFocus(field, value)	Moves the input focus to the designated field.

B.5 References

[Sun] Sun Microsystems, JavaDoc for the Pattern class, http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html

[Javascript] Netscape Corporation, Javascript Reference (versions 1.3, 1.4 and 1.5), http://devedge.netscape.com/central/javascript/

Appendix C. Front-end databases scripts

User-roles database

MS SQL Server

```
IF EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = N'sgUsers')
	DROP DATABASE [sgUsers]
GO

CREATE DATABASE [sgUsers]
GO

use [sgUsers]
GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[group_users]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[group_users]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[groups]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[groups]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[roles]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[roles]
GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[rolescopes]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[rolescopes]
GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[user_roles]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[user_roles]
GO

if exists (select * from dbo.sysobjects where id = object_id(N'[dbo].[users]')
and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[users]
GO

if not exists (select * from master.dbo.syslogins where loginname = N'db_user')
BEGIN
	declare @logindb nvarchar(132), @loginlang nvarchar(132) select @logindb =
N'outUsers', @loginlang = N'us_english'
	if @logindb is null or not exists (select * from master.dbo.sysdatabases
where name = @logindb)
		select @logindb = N'master'
	if @loginlang is null or (not exists (select * from master.dbo.syslanguages
where name = @loginlang) and @loginlang <> N'us_english')
		select @loginlang = @@language
	exec sp_addlogin N'db_user', null, @logindb, @loginlang
END
```

```
GO

if not exists (select * from dbo.sysusers where name = N'db_user' and uid <
16382)
	EXEC sp_grantdbaccess N'db_user', N'db_user'
GO

exec sp_addrolemember N'db_datareader', N'db_user'
GO

exec sp_addrolemember N'db_datawriter', N'db_user'
GO

CREATE TABLE [dbo].[group_users] (
	[group_id] [varchar] (15)  NOT NULL ,
	[smartgov_user_id] [varchar] (15)  NOT NULL
)
GO

CREATE TABLE [dbo].[groups] (
	[group_id] [varchar] (15)  NOT NULL ,
	[type] [varchar] (8)  NOT NULL ,
	[outer_user_system] [varchar] (15)  NULL ,
	[group_name] [varchar] (25)  NOT NULL
)
GO

CREATE TABLE [dbo].[roles] (
	[rolescope_id] [varchar] (15)  NOT NULL ,
	[role_id] [varchar] (15)  NOT NULL ,
	[role_name] [varchar] (50)  NOT NULL
)
GO

CREATE TABLE [dbo].[rolescopes] (
	[rolescope_id] [varchar] (15)  NOT NULL ,
	[rolescope_name] [varchar] (50)  NOT NULL
)
GO

CREATE TABLE [dbo].[user_roles] (
	[group_id] [varchar] (15)  NOT NULL ,
	[smartgov_user_id] [varchar] (15)  NOT NULL ,
	[role_scope_id] [varchar] (15)  NOT NULL ,
	[role_id] [varchar] (15)  NOT NULL
)
GO

CREATE TABLE [dbo].[users] (
	[smartgov_user_id] [varchar] (15)  NOT NULL ,
	[smartgov_user_password] [varchar] (15)  NOT NULL ,
	[outer_user_id] [varchar] (15)  NOT NULL
)
GO

ALTER TABLE [dbo].[group_users] WITH NOCHECK ADD
	CONSTRAINT [PK_group_users] PRIMARY KEY  CLUSTERED
	(
		[group_id],
		[smartgov_user_id]
	)
GO
```

```
ALTER TABLE [dbo].[groups] WITH NOCHECK ADD
	CONSTRAINT [PK_groups] PRIMARY KEY  CLUSTERED
	(
		[group_id]
	)
GO

ALTER TABLE [dbo].[roles] WITH NOCHECK ADD
	CONSTRAINT [PK_roles] PRIMARY KEY  CLUSTERED
	(
		[rolescope_id],
		[role_id]
	)
GO

ALTER TABLE [dbo].[rolescopes] WITH NOCHECK ADD
	CONSTRAINT [PK_rolescopes] PRIMARY KEY  CLUSTERED
	(
		[rolescope_id]
	)
GO

ALTER TABLE [dbo].[user_roles] WITH NOCHECK ADD
	CONSTRAINT [PK_user_roles] PRIMARY KEY  CLUSTERED
	(
		[group_id],
		[smartgov_user_id],
		[role_scope_id],
		[role_id]
	)
GO

ALTER TABLE [dbo].[users] WITH NOCHECK ADD
	CONSTRAINT [PK_users] PRIMARY KEY  CLUSTERED
	(
		[smartgov_user_id]
	)
GO


/*
 * Dumping data for table 'roles'
 */
INSERT INTO roles VALUES('smartgov','admin','Administrator');
INSERT INTO roles VALUES('smartgov','manager','Manager');
INSERT INTO roles VALUES('smartgov','expert','Domain Expert');
INSERT INTO roles VALUES('smartgov','staff','IT Staff');
INSERT INTO roles VALUES('smartgov','worker','Service Worker');
INSERT INTO roles VALUES('wf_ku','editor','Editor');
INSERT INTO roles VALUES('wf_ku','reviewer','Reviewer');
INSERT INTO roles VALUES('wf_ku','approver','Approver');
INSERT INTO roles VALUES('wf_ts','editor','Editor');
INSERT INTO roles VALUES('wf_ts','approver','Approver');


/*
* Dumping data for table 'rolescopes'
*/
INSERT INTO rolescopes VALUES('smartgov','SmartGov System');
INSERT INTO rolescopes VALUES('wf_ts','Workflow of TS');
INSERT INTO rolescopes VALUES('wf_ku','Workflow of KU');
```

```
/*
* Dumping data for table 'users'
*/
INSERT INTO users VALUES('administrator','administrator','SmartGov');
INSERT INTO users VALUES('user_expert','user_expert','SmartGov');
INSERT INTO users VALUES('user_manager','user_manager','SmartGov');
INSERT INTO users VALUES('user_staff','user_staff','SmartGov');

/*
* Dumping data for table 'user_roles'
*/
INSERT INTO user_roles VALUES('test','administrator','wf_ts','editor');
INSERT INTO user_roles VALUES('test','user_expert','wf_ku','editor');
INSERT INTO user_roles VALUES('test','user_expert','wf_ts','editor');
INSERT INTO user_roles VALUES('test','user_manager','wf_ku','editor');
INSERT INTO user_roles VALUES('test','user_manager','wf_ts','editor');
INSERT INTO user_roles VALUES('test','user_staff','wf_ku','editor');
INSERT INTO user_roles VALUES('test','user_staff','wf_ts','editor');
INSERT INTO user_roles VALUES('user_system','administrator','smartgov','admin');
INSERT INTO user_roles VALUES('user_system','user_expert','smartgov','expert');
INSERT INTO user_roles VALUES('user_system','user_manager','smartgov','manager');
INSERT INTO user_roles VALUES('user_system','user_staff','smartgov','staff');

/*
* Dumping data for table 'groups'
*/
INSERT INTO groups VALUES('user_system','us','smartgov_outer','User System');
INSERT INTO groups VALUES('test','wg','','Testing Group');

/*
* Dumping data for table 'group_users'
*/
INSERT INTO group_users VALUES('test','administrator');
INSERT INTO group_users VALUES('test','user_expert');
INSERT INTO group_users VALUES('test','user_manager');
INSERT INTO group_users VALUES('test','user_staff');
INSERT INTO group_users VALUES('user_system','administrator');
INSERT INTO group_users VALUES('user_system','user_expert');
INSERT INTO group_users VALUES('user_system','user_manager');
INSERT INTO group_users VALUES('user_system','user_staff');
```

MySQL

```
CREATE DATABASE smartgov_user_system;

USE smartgov_user_system;

GRANT ALL PRIVILEGES ON *.* TO db_user@"%" IDENTIFIED BY 'egov' WITH GRANT
OPTION;


#
# Table structure for table 'group_users'
#

DROP TABLE IF EXISTS group_users;
CREATE TABLE `group_users` (
  `group_id` varchar(15) NOT NULL default '',
  `smartgov_user_id` varchar(15) NOT NULL default '',
```

```
  PRIMARY KEY  (`group_id`,`smartgov_user_id`)
) TYPE=MyISAM;


#
# Table structure for table 'groups'
#

DROP TABLE IF EXISTS groups;
CREATE TABLE `groups` (
  `group_id` varchar(15) NOT NULL default '',
  `type` varchar(8) NOT NULL default '',
  `outer_user_system` varchar(15) default '',
  `group_name` varchar(25) NOT NULL default '',
  PRIMARY KEY  (`group_id`)
) TYPE=MyISAM;

#
# Table structure for table 'roles'
#

DROP TABLE IF EXISTS roles;
CREATE TABLE `roles` (
  `rolescope_id` varchar(15) NOT NULL default '',
  `role_id` varchar(15) NOT NULL default '',
  `role_name` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`rolescope_id`,`role_id`)
) TYPE=MyISAM;


#
# Table structure for table 'rolescopes'
#

DROP TABLE IF EXISTS rolescopes;
CREATE TABLE `rolescopes` (
  `rolescope_id` varchar(15) NOT NULL default '',
  `rolescope_name` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`rolescope_id`)
) TYPE=MyISAM;

#
# Table structure for table 'user_roles'
#

DROP TABLE IF EXISTS user_roles;
CREATE TABLE `user_roles` (
  `group_id` varchar(15) NOT NULL default '',
  `smartgov_user_id` varchar(15) NOT NULL default '',
  `role_scope_id` varchar(15) NOT NULL default '',
  `role_id` varchar(15) NOT NULL default '',
  PRIMARY KEY  (`group_id`,`smartgov_user_id`,`role_scope_id`,`role_id`)
) TYPE=MyISAM;


#
# Table structure for table 'users'
#

DROP TABLE IF EXISTS users;
CREATE TABLE `users` (
  `smartgov_user_id` varchar(15) NOT NULL default '',
```

```
  `smartgov_user_password` varchar(15) NOT NULL default '',
  `outer_user_id` varchar(15) NOT NULL default '',
  PRIMARY KEY  (`smartgov_user_id`)
) TYPE=MyISAM;



#
# Dumping data for table 'roles'
#
INSERT INTO roles VALUES("smartgov","admin","Administrator");
INSERT INTO roles VALUES("smartgov","manager","Manager");
INSERT INTO roles VALUES("smartgov","expert","Domain Expert");
INSERT INTO roles VALUES("smartgov","staff","IT Staff");
INSERT INTO roles VALUES("smartgov","worker","Service Worker");
INSERT INTO roles VALUES("wf_ku","editor","Editor");
INSERT INTO roles VALUES("wf_ku","reviewer","Reviewer");
INSERT INTO roles VALUES("wf_ku","approver","Approver");
INSERT INTO roles VALUES("wf_ts","editor","Editor");
INSERT INTO roles VALUES("wf_ts","approver","Approver");


#
# Dumping data for table 'rolescopes'
#
INSERT INTO rolescopes VALUES("smartgov","SmartGov System");
INSERT INTO rolescopes VALUES("wf_ts","Workflow of TS");
INSERT INTO rolescopes VALUES("wf_ku","Workflow of KU");


#
# Dumping data for table 'users'
#
INSERT INTO users VALUES("administrator","administrator","SmartGov");
INSERT INTO users VALUES("user_expert","user_expert","SmartGov");
INSERT INTO users VALUES("user_manager","user_manager","SmartGov");
INSERT INTO users VALUES("user_staff","user_staff","SmartGov");

#
# Dumping data for table 'user_roles'
#
INSERT INTO user_roles VALUES("test","administrator","wf_ts","editor");
INSERT INTO user_roles VALUES("test","user_expert","wf_ku","editor");
INSERT INTO user_roles VALUES("test","user_expert","wf_ts","editor");
INSERT INTO user_roles VALUES("test","user_manager","wf_ku","editor");
INSERT INTO user_roles VALUES("test","user_manager","wf_ts","editor");
INSERT INTO user_roles VALUES("test","user_staff","wf_ku","editor");
INSERT INTO user_roles VALUES("test","user_staff","wf_ts","editor");
INSERT INTO user_roles VALUES("user_system","administrator","smartgov","admin");
INSERT INTO user_roles VALUES("user_system","user_expert","smartgov","expert");
INSERT INTO user_roles VALUES("user_system","user_manager","smartgov","manager");
INSERT INTO user_roles VALUES("user_system","user_staff","smartgov","staff");

#
# Dumping data for table 'groups'
#
INSERT INTO groups VALUES("user_system","us","smartgov_outer","User System");
INSERT INTO groups VALUES("test","wg","","Testing Group");

#
# Dumping data for table 'group_users'
#
```

```
INSERT INTO group_users VALUES("test","administrator");
INSERT INTO group_users VALUES("test","user_expert");
INSERT INTO group_users VALUES("test","user_manager");
INSERT INTO group_users VALUES("test","user_staff");
INSERT INTO group_users VALUES("user_system","administrator");
INSERT INTO group_users VALUES("user_system","user_expert");
INSERT INTO group_users VALUES("user_system","user_manager");
INSERT INTO group_users VALUES("user_system","user_staff");
```

Outer users database

MS SQL Server

```
IF EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = N'outUsers')
       DROP DATABASE [outUsers]
GO

CREATE DATABASE [outUsers]
GO

use [outUsers]
GO

if exists (select * from dbo.sysobjects where id =
object_id(N'[dbo].[outer_users]') and OBJECTPROPERTY(id, N'IsUserTable') = 1)
drop table [dbo].[outer_users]
GO

if not exists (select * from master.dbo.syslogins where loginname = N'db_user')
BEGIN
       declare @logindb nvarchar(132), @loginlang nvarchar(132) select @logindb =
N'outUsers', @loginlang = N'us_english'
       if @logindb is null or not exists (select * from master.dbo.sysdatabases
where name = @logindb)
              select @logindb = N'master'
       if @loginlang is null or (not exists (select * from master.dbo.syslanguages
where name = @loginlang) and @loginlang <> N'us_english')
              select @loginlang = @@language
       exec sp_addlogin N'db_user', null, @logindb, @loginlang
END
GO

if not exists (select * from dbo.sysusers where name = N'db_user' and uid <
16382)
       EXEC sp_grantdbaccess N'db_user', N'db_user'
GO

exec sp_addrolemember N'db_datareader', N'db_user'
GO

exec sp_addrolemember N'db_datawriter', N'db_user'
GO

CREATE TABLE [dbo].[outer_users] (
       [outer_user_id] [varchar] (15)  NOT NULL ,
       [outer_user_first_name] [varchar] (50) NOT NULL ,
       [outer_user_last_name] [varchar] (50) NOT NULL ,
       [outer_user_e_mail] [varchar] (80) NOT NULL
```

```
)
GO

ALTER TABLE [dbo].[outer_users] WITH NOCHECK ADD
       CONSTRAINT [PK_outer_users] PRIMARY KEY  CLUSTERED
       (
              [outer_user_id]
       )
GO


INSERT INTO outer_users
VALUES('administrator','SmartGov','Administrator','admin@smartgov.com')
INSERT INTO outer_users
VALUES('user_staff','test','staff','user_staff@smartgov.com')
INSERT INTO outer_users
VALUES('user_manager','test','manager','user_manager@smartgov.com')
INSERT INTO outer_users
VALUES('user_expert','expert','expert','expert@smartgov.com')
```

MySQL

```
CREATE DATABASE smartgov_outer_users;

USE smartgov_outer_users;


#
# Table structure for table 'outer_users'
#

DROP TABLE IF EXISTS outer_users;
CREATE TABLE `outer_users` (
  `outer_user_id` varchar(15) NOT NULL default '',
  `outer_user_first_name` varchar(50) NOT NULL default '',
  `outer_user_last_name` varchar(50) NOT NULL default '',
  `outer_user_e_mail` varchar(80) NOT NULL default '',
  PRIMARY KEY  (`outer_user_id`)
) TYPE=MyISAM;

#
# Dumping data for table 'outer_users'
#
INSERT INTO outer_users
VALUES("administrator","SmartGov","Administrator","admin@smartgov.com");
INSERT INTO outer_users
VALUES("user_staff","test","staff","user_staff@smartgov.com");
INSERT INTO outer_users
VALUES("user_manager","test","manager","user_manager@smartgov.com");
INSERT INTO outer_users
VALUES("user_expert","expert","expert","expert@smartgov.com");
```

XML Repository database

MS SQL Server

```
IF EXISTS (SELECT name FROM master.dbo.sysdatabases WHERE name = N'xmlStore')
       DROP DATABASE [xmlStore]
```

```
GO

CREATE DATABASE [xmlStore]
GO

use [xmlStore]
GO

if not exists (select * from master.dbo.syslogins where loginname =
N'xmlstore_user')
BEGIN
	declare @logindb nvarchar(132), @loginlang nvarchar(132) select @logindb =
N'xmlStore', @loginlang = N'us_english'
	if @logindb is null or not exists (select * from master.dbo.sysdatabases
where name = @logindb)
		select @logindb = N'master'
	if @loginlang is null or (not exists (select * from master.dbo.syslanguages
where name = @loginlang) and @loginlang <> N'us_english')
		select @loginlang = @@language
	exec sp_addlogin N'xmlstore_user', null, @logindb, @loginlang
END
GO

if not exists (select * from dbo.sysusers where name = N'xmlstore_user' and uid <
16382)
	EXEC sp_grantdbaccess N'xmlstore_user', N'xmlstore_user'
GO

exec sp_addrolemember N'db_datareader', N'xmlstore_user'
GO

exec sp_addrolemember N'db_datawriter', N'xmlstore_user'
GO

exec sp_addrolemember N'db_owner', N'xmlstore_user'
GO
```

MySQL

```
CREATE DATABASE xmlstore;

GRANT ALL PRIVILEGES ON *.* TO xmlstore_user@"%" IDENTIFIED BY 'egov' WITH GRANT
OPTION;
GRANT ALL PRIVILEGES ON *.* TO xmlstore_user@locahost IDENTIFIED BY 'egov' WITH
GRANT OPTION;
```

Appendix D. IIG / SGA DBs creation script[4]

MS SQL Server

```
drop table entraPAQ;
drop table adelantePAQIIG;
drop table adelantePAQ;
drop table entraPAQIIG;
drop table SEPdatabaseTable;
drop table databaseTable;
drop table autoKeys;

CREATE TABLE entraPAQ (
      entraPAQId INTEGER NOT NULL,
      XMLMethodDescription VARCHAR(4000) NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      notificationName VARCHAR(500) NOT NULL,
      PRIMARY KEY (entraPAQId)
);

CREATE TABLE adelantePAQIIG (
      adelantePAQId INTEGER NOT NULL,
      notificationName VARCHAR(500) NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));

CREATE TABLE adelantePAQ (
      adelantePAQId INTEGER NOT NULL,
      requestId INTEGER NOT NULL,
      serviceName VARCHAR(500) NOT NULL,
      XMLPacket VARCHAR(4000) NOT NULL,
      realTime INTEGER NOT NULL,
      persistent INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));
```

[4] Oracle script has not been tested

```
CREATE TABLE entraPAQIIG (
      entraPAQId INTEGER NOT NULL,
      serviceName VARCHAR(500) NOT NULL,
      XMLMessage VARCHAR(4000) NOT NULL,
      realTime INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (entraPAQId) );

CREATE TABLE SEPdatabaseTable (
      SEPdatabaseTableId INTEGER NOT NULL,
      serviceName VARCHAR(500) NOT NULL,
      XMLMessage VARCHAR(4000) NOT NULL,
      realTime INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (SEPdatabaseTableId));

CREATE TABLE databaseTable (
      databaseTableId INTEGER NOT NULL,
      requestId INTEGER NOT NULL,
      serviceName VARCHAR(500) NOT NULL,
      XMLMessage VARCHAR(4000) NOT NULL,
      realTime INTEGER NOT NULL,
      persistent INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (databaseTableId) );

CREATE TABLE autoKeys (
      KeyName VARCHAR(32) NOT NULL,
      KeyValue INTEGER NOT NULL,
      PRIMARY KEY (keyName) );
```

MySQL

```
drop table entraPAQ;
drop table adelantePAQIIG;
drop table adelantePAQ;
drop table entraPAQIIG;
```

```
drop table SEPdatabaseTable;
drop table databaseTable;
drop table autoKeys;

CREATE TABLE entraPAQ (
      entraPAQId INTEGER NOT NULL,
      XMLMethodDescription BLOB NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      notificationName BLOB NOT NULL,
      PRIMARY KEY (entraPAQId));

CREATE TABLE adelantePAQIIG (
      adelantePAQId INTEGER NOT NULL,
      notificationName BLOB NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));

CREATE TABLE adelantePAQ (
      adelantePAQId INTEGER NOT NULL,
      serviceName BLOB NOT NULL,
      XMLPacket BLOB NOT NULL,
      realTime INTEGER NOT NULL,
      persistent INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));

CREATE TABLE entraPAQIIG (
      entraPAQId INTEGER NOT NULL,
      serviceName BLOB NOT NULL,
      XMLMessage BLOB NOT NULL,
      realTime INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (entraPAQId) );

CREATE TABLE SEPdatabaseTable (
      SEPdatabaseTableId INTEGER NOT NULL,
      serviceName BLOB NOT NULL,
```

```
      XMLMessage BLOB NOT NULL,
      realTime INTEGER NOT NULL,
      SGtimeStamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (SEPdatabaseTableId));

CREATE TABLE databaseTable (
      databaseTableId INTEGER NOT NULL,
      requestId INTEGER NOT NULL,
      serviceName BLOB NOT NULL,
      XMLMessage BLOB NOT NULL,
      realTime INTEGER NOT NULL,
      persistent INTEGER NOT NULL,
      SGtimestamp VARCHAR(22) NOT NULL,
      PRIMARY KEY (databaseTableId) );

CREATE TABLE autoKeys (
      KeyName VARCHAR(32) NOT NULL,
      KeyValue INTEGER NOT NULL,
      PRIMARY KEY (keyName) );
```

Oracle

```
drop table entraPAQ;
drop table adelantePAQIIG;
drop table adelantePAQ;
drop table entraPAQIIG;
drop table SEPdatabaseTable;
drop table databaseTable;
drop table autoKeys;

CREATE TABLE entraPAQ (
      entraPAQId NUMBER(38) NOT NULL,
      XMLMethodDescription VARCHAR2(4000) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      notificationName VARCHAR2(500) NOT NULL,
      PRIMARY KEY (entraPAQId)
);
```

```
CREATE TABLE adelantePAQIIG (
      adelantePAQId NUMBER(38) NOT NULL,
      notificationName VARCHAR2(500) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));

CREATE TABLE adelantePAQ (
      adelantePAQId NUMBER(38) NOT NULL,
      requestId NUMBER(38) NOT NULL,
      serviceName VARCHAR2(500) NOT NULL,
      XMLPacket VARCHAR2(4000) NOT NULL,
      realTime NUMBER(2) NOT NULL,
      persistent NUMBER(2) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      PRIMARY KEY (adelantePAQId));

CREATE TABLE entraPAQIIG (
      entraPAQId NUMBER(38) NOT NULL,
      serviceName VARCHAR2(500) NOT NULL,
      XMLMessage VARCHAR2(4000) NOT NULL,
      realTime NUMBER(2) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      PRIMARY KEY (entraPAQId) );

CREATE TABLE SEPdatabaseTable (
      SEPdatabaseTableId NUMBER(38) NOT NULL,
      serviceName VARCHAR2(500) NOT NULL,
      XMLMessage VARCHAR2(4000) NOT NULL,
      realTime NUMBER(2) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      PRIMARY KEY (SEPdatabaseTableId));

CREATE TABLE databaseTable (
      databaseTableId NUMBER(38) NOT NULL,
      requestId NUMBER(38) NOT NULL,
      serviceName VARCHAR2(500) NOT NULL,
      XMLMessage VARCHAR2(4000) NOT NULL,
```

```
      realTime NUMBER(2) NOT NULL,
      persistent NUMBER(2) NOT NULL,
      SGtimestamp VARCHAR2(22) NOT NULL,
      PRIMARY KEY (databaseTableId) );

CREATE TABLE autoKeys (
      KeyName VARCHAR2(32) NOT NULL,
      KeyValue NUMBER(38) NOT NULL,
      PRIMARY KEY (keyName) );
```

Appendix E. Login DB creation script[5]

MS SQL Server

```
CREATE TABLE SGuserData (
      userID      INTEGER NOT NULL,
      userName    NVARCHAR(64) NOT NULL UNIQUE,
      password    NVARCHAR(64) NOT NULL,
      fullName    NVARCHAR(128) NOT NULL,
      PRIMARY KEY(userID)
);

CREATE TABLE SGuserServices (
      userID      INTEGER NOT NULL REFERENCES SGuserData (userId),
      serviceName VARCHAR(128) NOT NULL,
      PRIMARY KEY(userID, serviceName) )
```

MySQL

```
CREATE TABLE SGuserData (
      userID      INTEGER NOT NULL,
      userName    VARCHAR(64) NOT NULL UNIQUE,
      password    VARCHAR(64) NOT NULL,
      fullName    VARCHAR(128) NOT NULL,
      PRIMARY KEY(userID)
);

CREATE TABLE SGuserServices (
      userID      INTEGER NOT NULL REFERENCES SGuserData (userId),
      serviceName VARCHAR(128) NOT NULL,
      PRIMARY KEY(userID, serviceName) )
```

[5] Oracle script has not been tested

Oracle

```
CREATE TABLE SGuserData (
      userID      NUMBER(12) NOT NULL,
      userName    VARCHAR(64) NOT NULL UNIQUE,
      password    VARCHAR(64) NOT NULL,
      fullName    VARCHAR(128) NOT NULL,
      PRIMARY KEY(userID)
);

CREATE TABLE SGuserServices (
      userID      NUMBER(12) NOT NULL REFERENCES SGuserData (userId),
      serviceName VARCHAR(128) NOT NULL,
      PRIMARY KEY(userID, serviceName)
```

Appendix F. Sample XML document for eVies personal details

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceResults>
    <serviceName>EVAT_AQ</serviceName>
    <userName>XXX</userName>
    <timestamp>XXX</timestamp>
    <row>
        <tseElement>
            <name>TSE_EVAT_IS_CORRECTIVE</name>
            <value>true</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_CURRENCY</name>
            <value>Euro</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_DCL_NO</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_YEAR</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_TAX_OFFICE</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_SUBM_DATE</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_TRIMESTER</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
            <name>TSE_EVAT_RECEIVING_TAX_OFFICE</name>
            <value>XXX</value>
        </tseElement>
        <tseElement>
```

```
            <name>TSE_EVAT_RECEPTION_DATE</name>
            <value>XXX</value>
        </tseElement>
        <groupElement groupId="TSEG_EVAT_PERIOD">
            <row>
                <tseElement>
                    <name>TSE_EVAT_PERIOD_END</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_PERIOD_BEGIN</name>
                    <value>XXX</value>
                </tseElement>
            </row>
        </groupElement>
        <groupElement groupId="TSEG_EVAT_CONTACT">
            <row>
                <tseElement>
                    <name>TSE_EVAT_FILE_NO</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_AFM</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_ADDRESS</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_COMPANY_TITLE</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_FAX</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_PHONE</name>
                    <value>XXX</value>
                </tseElement>
                <tseElement>
                    <name>TSE_EVAT_REG_AREA</name>
```

```
                <value>XXX</value>
            </tseElement>
            <tseElement>
                <name>TSE_EVAT_REG_TK</name>
                <value>XXX</value>
            </tseElement>
        </row>
    </groupElement>
</row>
</ServiceResults>
```