eAUDIT: Designing a generic tool to review entitlements

GIAC (GCCC) Gold Certification

Author: François Bégin, francois.begin@telus.com
Advisor: Dr. Ray Davidson
Accepted: June 15th 2015

Abstract

In a perfect world, identity and access management would be handled in a fully automated way. On their first day of work, new employees would receive all the required access to the systems they need in order to perform their job function. Over time, as their roles within the company evolved, these entitlements would be automatically adjusted. Unfortunately, we do not live in a perfect world. Access to systems is often cumulative, with employees keeping access they no longer require. This in turn poses a risk to the enterprise: unneeded access can lead to abuses and increases the possibility of data leakage if an employee is social engineered. This paper proposes a system to help address this problem: eAUDIT is a custom-built, generic entitlement review system that can simplify the task of reviewing user entitlements. eAUDIT is well suited to cases where no such tool exists in an enterprise, but can also complement an identity management system that does not fully cover all systems and applications. This paper covers the design of eAUDIT as well as an overview of its implementation, including sample code.

Доверяй, но проверяй

Trust, but verify

Russian proverb

Special thanks to my A-Team of coders, Kobe Lin and Jaya Balasubramaniam, who made eAUDIT happen.

François Bégin, francois.begin@telus.com

1. Introduction

When a new employee joins a company, he should be automatically provided an identity within this company as well as various methods to authenticate himself. These authentication methods can take various forms: a physical access badge, a centrally-managed username and password, a virtual smart card carrying a cryptographically signed certificate, etc. Once an identity has been established for the new employee, this identity should be automatically associated with all the access entitlements he requires to perform his job functions. From that point on, as the employee progresses in his career, it is fairly likely that he will change teams, roles and positions. At every such milestone, his entitlements should be automatically re-adjusted, ensuring that he always has the access he requires. This will ensure that the employee's access is limited to the resources he requires, a key element of a good authorization policy (Ballad, Ballad, and Banks, 2010).

This, of course, is an idealized description of Identity and Access Management (IAM), which is of great value to any enterprise. But IAM can be challenging. As Mather, Kumaraswamy and Latif (2009) aptly pointed out, "Many of these [IAM] initiatives are entered into with high expectations, which is not surprising given that the problem is often large and complex."

This is particularly true for organizations that have grown through mergers and acquisition, as well as for organizations that have existed for decades and rely on legacy systems. Building IAM hooks (if that is even an option) into these systems can be costly or complicated. Sometimes, the best approach is to retain existing entitlements and address any access management gaps by conducting regular reviews of these entitlements to prune the ones that are no longer required.

Building a system that can conduct regular entitlement reviews is in itself a significant project, best handled by following a software development life cycle model. As part of the pre-development activities, one element of the discussion will invariably touch upon whether to develop the software functions in-house, outsource them, get an off-the-shelf package, or adapt and reuse exiting software (Saleh, 2009). While there are commercial products that can help conduct reviews of entitlements, these tools are often tied to large, costly products related to both IAM and Governance, Risk and Compliance (GRC) management. This paper chooses the first path: a custom-built solution. The main argument in favor of this choice is that, with a suitably clear scope, such a tool can be built and deployed at limited cost and provide a quick and significant benefit to an enterprise that does not have this capability. The tool presented here is called eAUDIT and can achieve a simple goal of providing a custom-made solution for entitlement review. The scope, design and high-level implementation of this tool are covered in this paper.

2. eAUDIT

2.1. Defining the elements of an entitlement audit

Prior to diving into the main topic, some of the key words that will be used throughout this paper need to be clearly defined.

In this paper, an *entitlement* refers to a privilege that has been granted to a user. Typical examples of entitlements include authorization to access a particular software application/data source, privileged access to a system or application, etc. Note that entitlements are granular: if a specific system has different levels of access, e.g. read access vs. read/write access, each of these count as a separate entitlement. The goal of an audit is to determine whether or not the entitlement is still valid.

An *authorizer* is defined as the person (or persons) who can grant someone a specific entitlement. In many organizations, managers often take on that role for their direct reports. This is supported by the fact that managers are well positioned to assess the business needs related to this type of access. Security professionals who may wish for more access control enforcements should be reminded that "Business will always trump security […]" (Kadrich, 2007). With that said, a manager is not always the main authorizer, as some systems have a specific business owner who plays that role.

Entitlements are granted to *entities*. In most cases, an entity will simply be an employee of the company, but since the goal of eAUDIT it to create a system that is purely generic, the word 'entity' is used instead. For instance, an audit could be conducted against physical access cards that are not associated directly to a user e.g.

generic access cards for escorted access that are left in the care of the security guard desk. These cards are a good example of a business need (convenience of being able to provide quick access to vendors on support calls) that outweighs security concerns (difficulty in associating a specific user to these cards).

One of the key elements of eAUDIT is its generic nature. Entities can be anything and the various characteristics that these entities possess can also be anything. In the previous example, badge entities can have attributes such as: Badge ID, Badge Label, Manager (if applicable), Badge Type, Expiry Date, etc., as shown in Table 1.

Primary attributes			***Secondary attributes***	
Badge ID	**Badge Label**	**Manager (if applicable)**	**Badge Type**	**Expiry date**
1000000	Penny Robinson	John Robinson [200000]	Employee badge	2020-01-01
1000001	Guard Desk	n/a	Generic cards	2015-12-31

Table 1. Mockup data for an audit of physical access badges.

In another example such as Table 2, where entities are defined as employees, the attributes of these employees would be different: Employee ID, First Name, Last Name, Title, Computer ID, City, etc.

Primary attributes				***Secondary attributes***	
Employee ID	**First Name**	**Last Name**	**Title**	**Computer ID**	**City**
200002	Don	West	Pilot	LX73-26	Toronto
200003	Judy	Robinson	Zoologist	LD04-34	Edmonton

Table 2. Mockup data for an audit of employee.

eAUDIT users are those individuals responsible to conduct audits based on any type of entities and/or attributes. Giving eAUDIT users the ability to define their own attributes to meet their needs is crucial. In eAUDIT, entity attributes are therefore referred to as *user-defined entity attributes* (UDEA). Furthermore, a distinction is made between *primary attributes* and *secondary attributes.* Primary attributes are those attributes that normally suffice to an authorizer for an entitlement review. Secondary attributes contain extra information that an authorizer may need to consult to make a final determination. How these attributes are presented to authorizers will prove important when discussing the creation of the web interface for the audit engine.

2.2. Scope

Since eAUDIT is a custom-built design and implementation project, one of the most important aspects of this project is to provide a clear scope. This will avoid *scope creep*, which is one of the top five reasons why a project can fail (Doraiswamy and Shiv, 2012).

The main elements of eAUDIT's scope are:

- Import generic data by relying on user-defined entity attributes.
- Present a simple landing page to authorizers, showing them active audits that need their attention.
- Present a simple entitlement review page where all entities are listed in a sortable/filterable data table. The data table will show the primary entity attributes (always) and secondary attributes (on demand) to the authorizer.
- Conduct entitlement review based on binary responses (confirm | revoke) through a single click for each entity.
- Support for multiple authorizers for any given entitlement.
- Ability to group similar entitlements together.

The last point is important to clarify: in eAUDIT, a given audit can cover more than one entitlement, provided that all entities have the same attributes. For example, consider Table 3, a mockup of the entitlement review page presented to an authorizer:

François Bégin, francois.begin@telus.com

Authorizer: John Robinson

Badge ID	Badge Label	Entitlement	Verify
1000000	Penny Robinson	*R* Main Data Center	**Review | Revoke**
1000001	Guard Desk	*R* Main Data Center	**Review | Revoke**
1000002	Maj. Don West	*R* DR Site	**Review | Revoke**
1000003	Robot	*R* Wire Center	**Review | Revoke**

Table 3. Mockup of an entitlement review page as presented to an authorizer.

In this particular audit, John Robinson is responsible for three different entitlements: ***R* Main Data Center**, ***R* DR Site** and ***R* Wire Center**. Since all these entitlements are similar in nature – and since they all relate to the same type of entities with the same attributes (badges granting physical access to buildings) – they are all part of the same meta-audit. Section 2.5 will discuss how the web interface will support the authorizers who are faced with multiple entitlements to review.

When entities and their attributes are loaded at the start of an audit, they are locked-in for the duration of the audit. This is an important design decision that warrants some additional explanation: after all, if we are reviewing administrative access to company-issued laptops, should users be added and removed from the audit if they are granted those rights while the audit is taking place?

Although this may appear to provide a more accurate representation of the data, the complications associated with auto-adjusting the dataset greatly outweigh the benefits of a locked-in audit. In some cases, data will be loaded in eAUDIT through a spreadsheet. Re-creating the spreadsheet throughout the audit would be time-consuming, and additional code would be required to adjust the existing data to handle deltas. Furthermore, audits conducted by eAUDIT typically last 3-4 weeks. These short audit windows are another mitigating factor. Rather than capturing in-flight deltas, eAUDIT will strive for a high success rate and repeated audits throughout the year. With all of this said, if data is loaded dynamically inside eAUDIT through an adapter, then that adapter can be written to handle in-flight changes.

2.3. Design

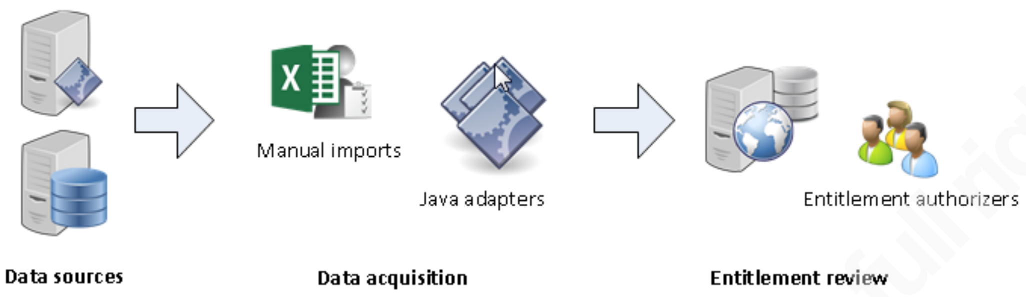
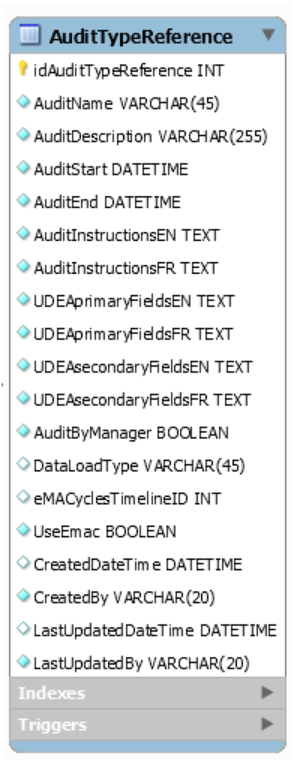


Figure 1. High-level overview of eAUDIT's design.

As previously alluded to, the design of eAUDIT (Figure 1) is fairly simple. At a high level, entitlement data needs to be acquired from various data sources, and then presented to authorizers for review. Section 2.4 covers data acquisition in more detail. How the data ends up in the database is critical. eAUDIT needs to handle generic data efficiently. Taking a page from Jurney (2013): "In choosing our tools, we seek linear scalability, but above all, we seek simplicity", the data model for eAUDIT seeks simplicity. In Jurney's case, he was referring to NoSQL big data tools, but his comment is just as applicable to a traditional SQL database. eAUDIT's data is held in a SQL database, with a simple data model (see Appendix A). Let us consider the data model and go over some of the key characteristics.

The **AuditTypeReference** table (figure 2) holds the base audit type information, such as the name and description of the audit, a start and end date and some instructions. This is where the overall type of audit is defined: reviewing active contractors at TELUS, reviewing badges with *R* profiles attached, etc. Since TELUS is a bilingual company, various key fields are held in two separate fields (EN | FR), which will be required by the Web interface.



AuditTypeReference

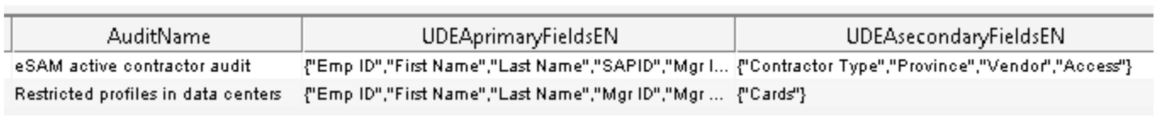
- idAuditTypeReference INT
- AuditName VARCHAR(45)
- AuditDescription VARCHAR(255)
- AuditStart DATETIME
- AuditEnd DATETIME
- AuditInstructionsEN TEXT
- AuditInstructionsFR TEXT
- UDEAprimaryFieldsEN TEXT
- UDEAprimaryFieldsFR TEXT
- UDEAsecondaryFieldsEN TEXT
- UDEAsecondaryFieldsFR TEXT
- AuditByManager BOOLEAN
- DataLoadType VARCHAR(45)
- eMACyclesTimelineID INT
- UseEmac BOOLEAN
- CreatedDateTime DATETIME
- CreatedBy VARCHAR(20)
- LastUpdatedDateTime DATETIME
- LastUpdatedBy VARCHAR(20)

Indexes

Triggers

Figure 2. AuditTypeReference table in the eAUDIT database.

The AuditTypeReference table also holds the labels for the primary and secondary User Defined Entity Attributes (UDEA) fields. Initially, a pre-defined set of fields was considered e.g. *UDEAPrimaryField1*, *UDEAPrimaryField2*, etc. but this set a hard limit as well as being less than esthetically pleasing from a data model perspective. Another approach considered was a mapping table that would hold as many of these user-defined fields as required, with a many-to-one relationship with the AuditTypeReference table. In the end though, a simple and elegant solution was chosen: the UDEA fields hold the labels in JSON format. Figure 3 shows examples of primary and secondary fields (labels) for two different types of audits:



AuditName	UDEAprimaryFieldsEN	UDEAsecondaryFieldsEN
eSAM active contractor audit	{"Emp ID","First Name","Last Name","SAPID","Mgr I...	{"Contractor Type","Province","Vendor","Access"}
Restricted profiles in data centers	{"Emp ID","First Name","Last Name","Mgr ID","Mgr ...	{"Cards"}

Figure 3. User-defined entity attributes fields for two different audits.

Table **Audit** (Figure 4) defines the actual entitlements being reviewed within an audit type. It also has a count of the total number of entities that bear each entitlement.

François Bégin, francois.begin@telus.com

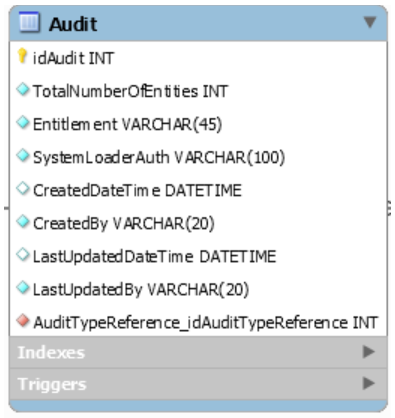
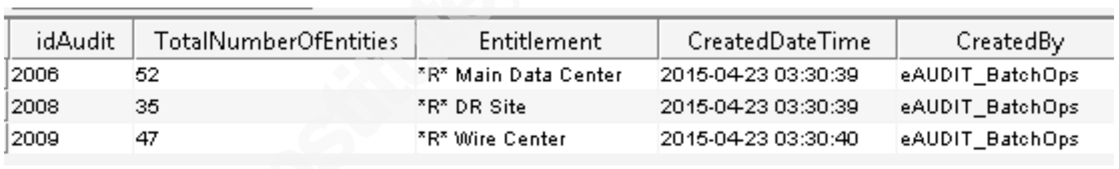

Audit
idAudit INT
TotalNumberOfEntities INT
Entitlement VARCHAR(45)
SystemLoaderAuth VARCHAR(100)
CreatedDateTime DATETIME
CreatedBy VARCHAR(20)
LastUpdatedDateTime DATETIME
LastUpdatedBy VARCHAR(20)
AuditTypeReference_idAuditTypeReference INT
Indexes
Triggers

Figure 4. Audit table in the eAUDIT database.

Using our previous example of restricted profiles on access badges, this table would hold all the different types of restricted profiles that are to be reviewed (Figure 5). The table uses a foreign key to link itself to the AuditTypeReference table. It is the relationship between Audit and AuditTypeReference that allows eAUDIT to group similar entitlement reviews together.



idAudit	TotalNumberOfEntities	Entitlement	CreatedDateTime	CreatedBy
2006	52	*R* Main Data Center	2015-04-23 03:30:39	eAUDIT_BatchOps
2008	35	*R* DR Site	2015-04-23 03:30:39	eAUDIT_BatchOps
2009	47	*R* Wire Center	2015-04-23 03:30:40	eAUDIT_BatchOps

Figure 5. Different entitlements reviewed within the same audit.

The **Entities** table (Figure 6) is the one that holds a list of who (or what) was granted entitlements. This is where the *UDEAprimaryFields* and *UDEAsecondaryFields* from the AuditTypeReference table will find their counterparts, called *UDEAprimaryFieldsValues* and *UDEAsecondaryFieldsValues* respectively.

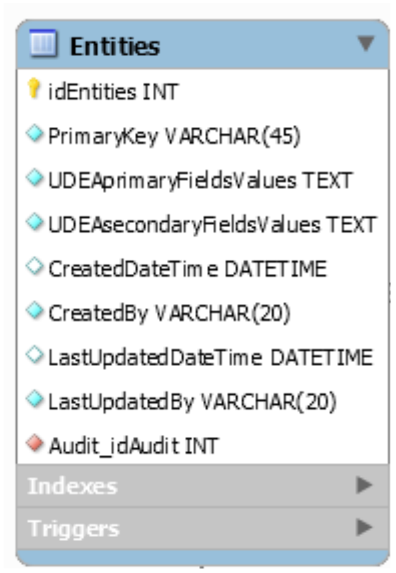
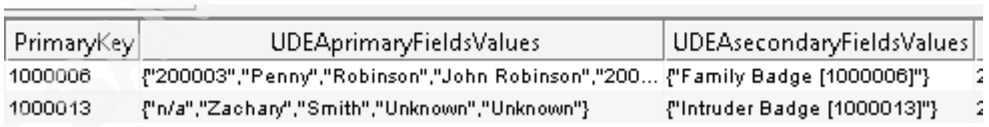


Figure 6. Entities table in the eAUDIT database.

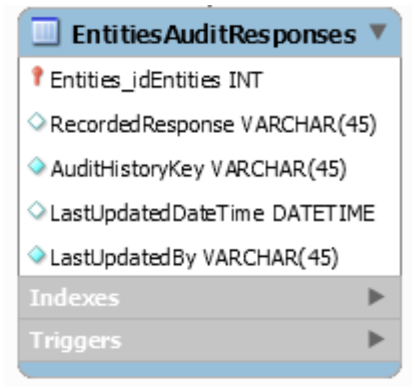
The values are saved in JSON format (see Figure 7), just like the labels. The choice of JSON as a format to hold field values may appear a little strange at first. After all, flattening all entities' attributes in a single field will make SQL searches more expensive. But the main goal of eAUDIT is not efficient searches. The goal is to efficiently present datasets for entitlement review. The choice of JSON to keep labels and values is directly related to the web interface that will allow authorizers to conduct their review – and having field values in this format will prove useful once we start discussing eAUDIT's web implementation in section 2.5.



PrimaryKey	UDEAprimaryFieldsValues	UDEAsecondaryFieldsValues	
1000006	{"200003","Penny","Robinson","John Robinson","200...	{"Family Badge [1000006]"}	2
1000013	{"n/a","Zachary","Smith","Unknown","Unknown"}	{"Intruder Badge [1000013]"}	2

Figure 7. User-defined field values as kept in the Entities table.

The **EntitiesAuditReponses** table (Figure 8) holds the responses i.e. whether or not the authorizer confirmed or revoked the entitlement. Keeping this data in a separate table allows for fields such as *LastUpdatedDateTime* and *LastUpdatedBy* to capture who reviewed the entitlement. Although mentioned here, note that the full implementation of this table's functionality is not covered in this paper.



EntitiesAuditResponses

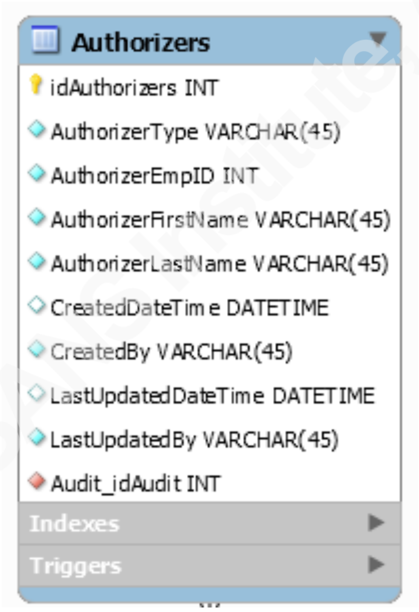
- Entities_idEntities INT
- RecordedResponse VARCHAR(45)
- AuditHistoryKey VARCHAR(45)
- LastUpdatedDateTime DATETIME
- LastUpdatedBy VARCHAR(45)

Indexes

Triggers

Figure 8. EntitiesAuditReponses table in the eAUDIT database.

Finally, the **Authorizers** table (Figure 9) is where the authorizers of a particular Audit record are kept. This table has a many-to-one relationship to the Audit table, allowing multiple individuals to be named authorizers of a particular audit. TELUS uses a unique numerical value for its employee IDs, and this is reflected in field *AuthorizerEmpID*, which is defined as an integer value.



Authorizers

- idAuthorizers INT
- AuthorizerType VARCHAR(45)
- AuthorizerEmpID INT
- AuthorizerFirstName VARCHAR(45)
- AuthorizerLastName VARCHAR(45)
- CreatedDateTime DATETIME
- CreatedBy VARCHAR(45)
- LastUpdatedDateTime DATETIME
- LastUpdatedBy VARCHAR(45)
- Audit_idAudit INT

Indexes

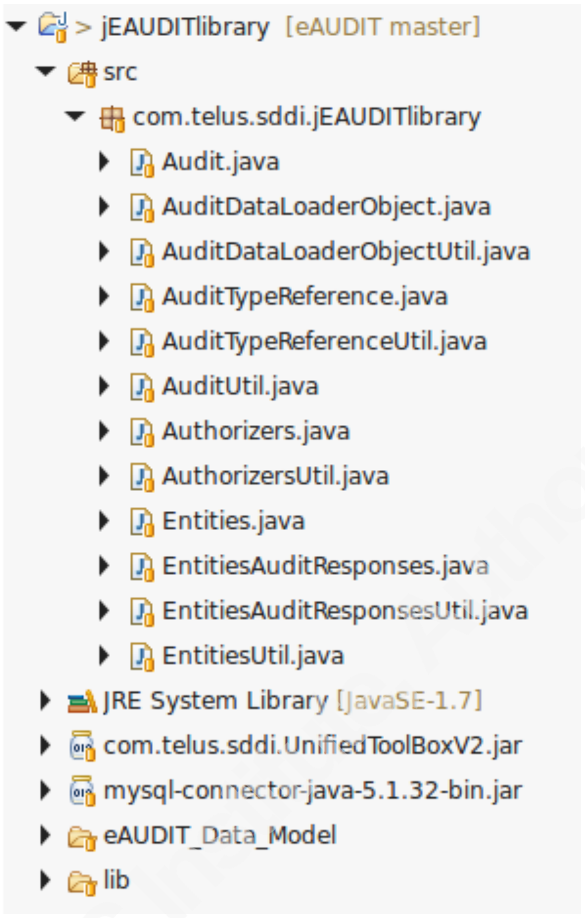
Triggers

Figure 9. Authorizers table in the eAUDIT database.

.

2.4. Data acquisition

Since eAUDIT is all about validating data, acquiring data is key to the tool. Not only that, but from the scope defined in section 2.2, data need to be acquired as easily as possible. After all, if you cannot load authorizers, entitlements and entities into the eAUDIT database, you cannot review these records. To help with the acquisition of data, a java library called **jEAUDITlibrary** was created (Figure 10).

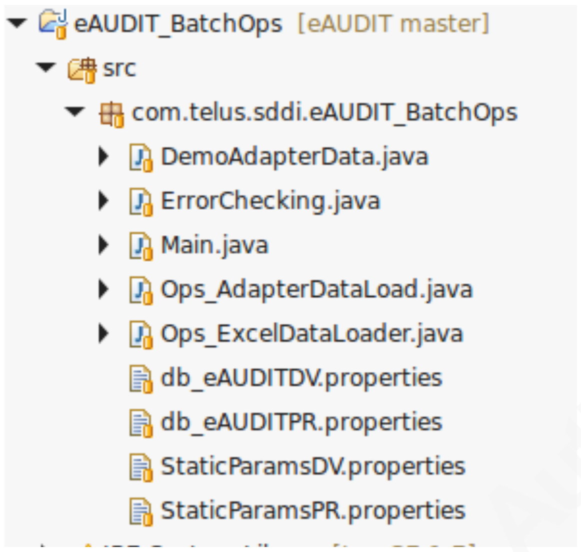


```
▾ > jEAUDITlibrary [eAUDIT master]
  ▾ src
    ▾ com.telus.sddi.jEAUDITlibrary
      ▸ Audit.java
      ▸ AuditDataLoaderObject.java
      ▸ AuditDataLoaderObjectUtil.java
      ▸ AuditTypeReference.java
      ▸ AuditTypeReferenceUtil.java
      ▸ AuditUtil.java
      ▸ Authorizers.java
      ▸ AuthorizersUtil.java
      ▸ Entities.java
      ▸ EntitiesAuditResponses.java
      ▸ EntitiesAuditResponsesUtil.java
      ▸ EntitiesUtil.java
  ▸ JRE System Library [JavaSE-1.7]
  ▸ com.telus.sddi.UnifiedToolBoxV2.jar
  ▸ mysql-connector-java-5.1.32-bin.jar
  ▸ eAUDIT_Data_Model
  ▸ lib
```

Figure 10. Main classes and supporting libraries of jEAUDITlibrary.

As would be expected from any other java library, jEAUDITlibrary contains classes for the main objects that are at the core of eAUDIT: *Audit*, *AuditTypeReference*, *Authorizers*, *Entities*, and *EntitiesAuditReponses*. These classes map to the data model (Appendix A) that was discussed in the previous section. There are also *Util classes that contains static methods related to these objects: *AuditUtil*, *AuditTypeReferenceUtil*, *AuthorizersUtil*, *EntitiesUtil*, and *EntitiesAuditReponsesUtil.* The eAUDIT library relies on two external libraries: the standard MySQL java connector and a toolbox library called UnifiedToolBoxV2, which is mostly used to help manage SQL connections as well as provide logging capabilities.

The source code for jEAUDITlibrary is available from the eAUDIT project page on GitHub (https://github.com/francoisbegin/eAUDIT). The project page also includes the source code for eAUDIT_BatchOps (Figure 11), a program that contains sample code to demonstrate data acquisition using the eAUDIT library. Appendix B gives a quick overview of how to set up an environment to use the library and eAUDIT_BatchOps.



```
eAUDIT_BatchOps [eAUDIT master]
  src
    com.telus.sddi.eAUDIT_BatchOps
      DemoAdapterData.java
      ErrorChecking.java
      Main.java
      Ops_AdapterDataLoad.java
      Ops_ExcelDataLoader.java
      db_eAUDITDV.properties
      db_eAUDITPR.properties
      StaticParamsDV.properties
      StaticParamsPR.properties
```

Figure 11. Main classes of eAUDIT_BatchOps (supporting libraries not shown)

2.4.1. Excel-driven data acquisition

The first demo in eAUDIT_BatchOps covers data import from an Excel spreadsheet. This particular section of the code relies heavily on Apache POI, an open-source Java API for Microsoft Documents (Apache Foundation, 2015). Note that all libraries required by eAUDIT_BatchOps are included with the project.

A demo spreadsheet, **dataLoadExample.xlsx**, is included with the eAUDIT_BatchOps project (Figure 12)

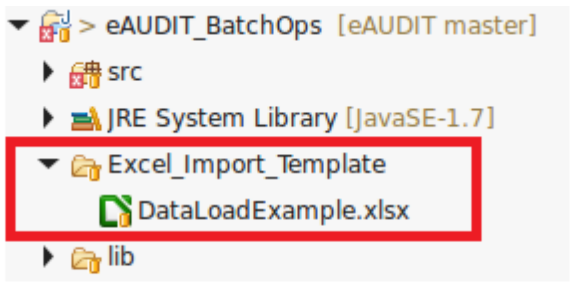

> eAUDIT_BatchOps [eAUDIT master]
src
JRE System Library [JavaSE-1.7]
Excel_Import_Template
DataLoadExample.xlsx
lib

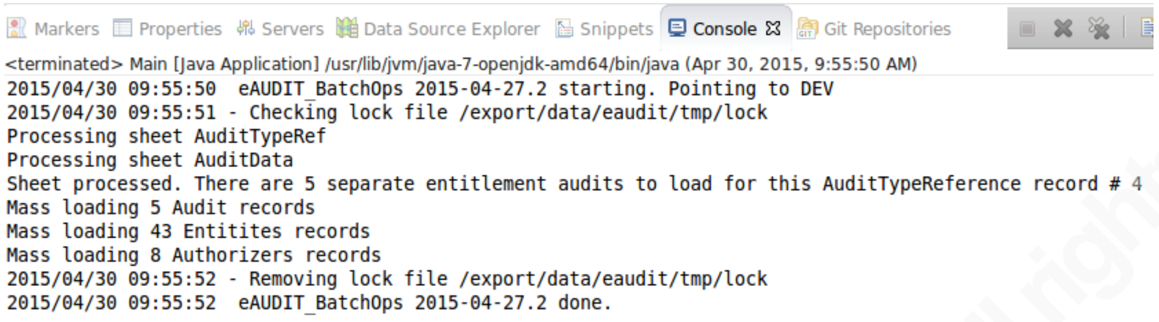
Figure 12. DataLoadExample.xlsx spreadsheet included with eAUDIT_BatchOps.

This spreadsheet holds three separate sheets: **AuditTypeRef** (Appendix C) contains all the data required to create the AuditTypeReference record in the eAUDIT database. **AuditData** (Appendix D) contains all the data required to create the Authorizers, Entities and Audit records. Finally, **Notes** contains notes to help users fill in the template correctly.

This spreadsheet can be used as a template for Excel-driven data loads. It is built to support generic data. To add/remove primary and secondary user-defined entity attribute fields, one only has to create/delete columns in the AuditData sheet and set the top row label to either *UDEAprimary* or *UDEAsecondary*.

Multiple Authorizers per entitlements is supported by listing them under the Authorizers column and separating each authorizer by a semi-colon. As mentioned in section 2.3, authorizer's IDs are expected to be integer values that map to the TELUS employee ID of the authorizer. Using eAUDIT at a company that relies on non-numerical employee IDs would require making modifications to the Authorizers class of jEAUDITlibrary.

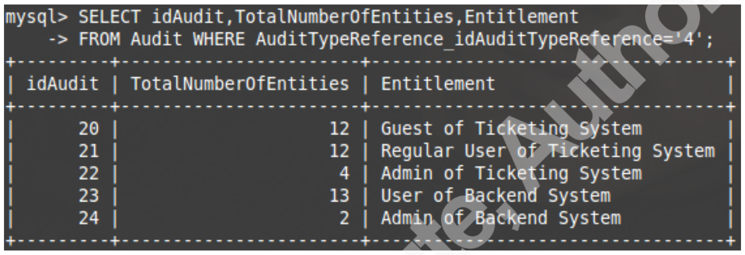
Method *Ops_ExcelDataLoader.loadFromExcelTemplate* handles the spreadsheet data load, provided it matches the expected template. To demonstrate this, one can create a new directory called *dataLoad* under the base path of eAUDIT and copy the demo spreadsheet to that location. One then compiles and runs eAUDIT_BatchOps with the *loadFromExcelTemplate* switch (or runs it with that switch inside a Java IDE):



```
<terminated> Main [Java Application] /usr/lib/jvm/java-7-openjdk-amd64/bin/java (Apr 30, 2015, 9:55:50 AM)
2015/04/30 09:55:50  eAUDIT_BatchOps 2015-04-27.2 starting. Pointing to DEV
2015/04/30 09:55:51 - Checking lock file /export/data/eaudit/tmp/lock
Processing sheet AuditTypeRef
Processing sheet AuditData
Sheet processed. There are 5 separate entitlement audits to load for this AuditTypeReference record # 4
Mass loading 5 Audit records
Mass loading 43 Entitites records
Mass loading 8 Authorizers records
2015/04/30 09:55:52 - Removing lock file /export/data/eaudit/tmp/lock
2015/04/30 09:55:52  eAUDIT_BatchOps 2015-04-27.2 done.
```

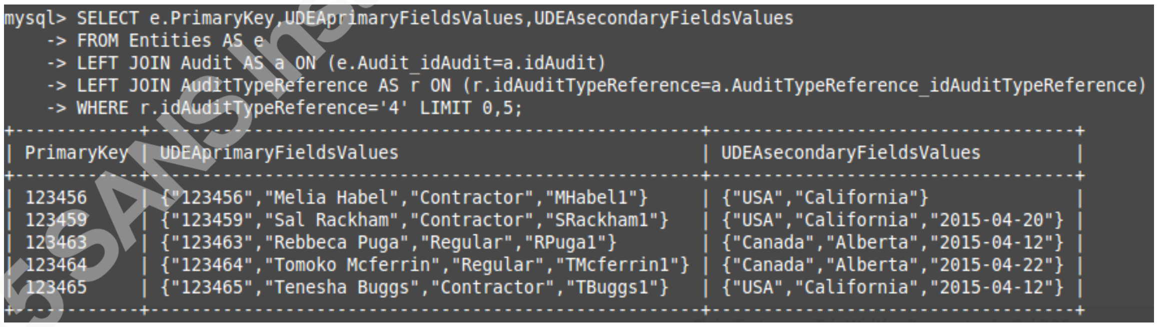
Figure 13. Loading data into eAUDIT using a spreadsheet.

In the example shown in Figure 13, a new AuditTypeReference record with id = 4 was created. Under that audit type reference, 5 sub-audits were created to match the 5 different entitlements that were defined in the spreadsheet (Figure 14), and 43 entities will need to be reviewed (Figure 15).



```
mysql> SELECT idAudit,TotalNumberOfEntities,Entitlement
    -> FROM Audit WHERE AuditTypeReference_idAuditTypeReference='4';
+---------+-----------------------+------------------------------+
| idAudit | TotalNumberOfEntities | Entitlement                  |
+---------+-----------------------+------------------------------+
|      20 |                    12 | Guest of Ticketing System    |
|      21 |                    12 | Regular User of Ticketing System |
|      22 |                     4 | Admin of Ticketing System    |
|      23 |                    13 | User of Backend System       |
|      24 |                     2 | Admin of Backend System      |
+---------+-----------------------+------------------------------+
```

Figure 14. Entitlements to be reviewed following the data load from a spreadsheet.



```
mysql> SELECT e.PrimaryKey,UDEAprimaryFieldsValues,UDEAsecondaryFieldsValues
    -> FROM Entities AS e
    -> LEFT JOIN Audit AS a ON (e.Audit_idAudit=a.idAudit)
    -> LEFT JOIN AuditTypeReference AS r ON (r.idAuditTypeReference=a.AuditTypeReference_idAuditTypeReference)
    -> WHERE r.idAuditTypeReference='4' LIMIT 0,5;
+------------+---------------------------------------------------+-----------------------------------+
| PrimaryKey | UDEAprimaryFieldsValues                           | UDEAsecondaryFieldsValues         |
+------------+---------------------------------------------------+-----------------------------------+
| 123456     | {"123456","Melia Habel","Contractor","MHabel1"}   | {"USA","California"}              |
| 123459     | {"123459","Sal Rackham","Contractor","SRackham1"} | {"USA","California","2015-04-20"} |
| 123463     | {"123463","Rebbeca Puga","Regular","RPuga1"}      | {"Canada","Alberta","2015-04-12"} |
| 123464     | {"123464","Tomoko Mcferrin","Regular","TMcferrin1"} | {"Canada","Alberta","2015-04-22"} |
| 123465     | {"123465","Tenesha Buggs","Contractor","TBuggs1"} | {"USA","California","2015-04-12"} |
+------------+---------------------------------------------------+-----------------------------------+
```

Figure 15. Entities to be reviewed following the data load from a spreadsheet.

Obviously, loading data from a spreadsheet is not ideal. Dealing with spreadsheets comes with pitfalls such as sensitive data exposure (Walsh, 2014). Still, spreadsheet imports may prove useful in cases where an application/system does not have a simple API to extract the data, or when a support team is reluctant to provide direct access to its data.

Another well-known reality is that, in many cases, spreadsheets are all that people know (Allen, 2015). Asking a subject-matter expert to do an export of the key data, and massaging these data to fit the eAUDIT Excel template might be the path of least resistance to acquire audit data.

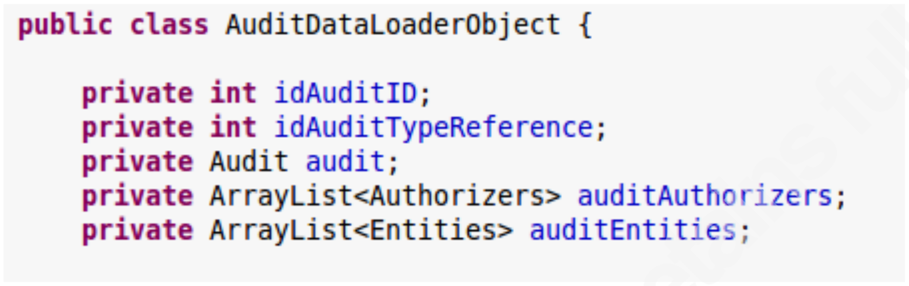
2.4.2. Adapter-driven data acquisition

A better solution to an Excel data-load is to write an adapter to retrieve the audit data from a system. In its current incarnation, eAUDIT does not provide a generic interface for such adapters, but the jEAUDITlibrary provides methods and classes to support their creation. Additionally, method *Ops_AdapterDataLoad.loadFromAdapter()* of eAUDIT_BathOps demonstrates the implementation of a simple adapter.

The first step in building a custom adapter is to determine how the data can be acquired programmatically. In this case, each solution will likely be custom-built. For instance, TELUS has developed a custom adapter to interact with the Lenel Onguard physical access system. Data acquisition was achieved by the use of two database service accounts: one account to interact with Lenel Onguard to obtain a list of badges and access profiles, and another account to interact with the database where authorizers of these access profiles are maintained. The adapter simply correlates the two data sources and feeds the resulting data to the eAUDIT database.

For an adapter data load, one must refer back to the data model and note that the Audit table is at the heart of the model. Contrary to all other tables in the model, its primary key, *idAudit*, is not an auto-increment field. This was done on purpose as it allows for the controlled creation of new records. When a custom-adapter needs to send data to the eAUDIT database, it needs to determine the next available idAudit value. Then, each new entitlement is assigned the next idAudit value. As this list is built up, the list of corresponding Authorizers and Entities records can also be built.

If all of these records were written one at a time, data loads would be woefully slow. A custom-built adapter should therefore take a different approach. In the demo code proposed in method *loadFromAdapter()*, a HashMap is built as the code iterates through all entities. If it finds a new entitlement, it creates a new Audit record and attaches to it the authorizers for that entitlement and the entity record. If it finds an entitlement it already knows about, it retrieves it from the HashMap, increments field *Audit.TotalNumberOfEntities*, and adds the new entity to previously correlated entities for that particular entitlement. All of this is done in memory, inside the HashMap. A special object called *jEAUDITlibrary.AuditDataLoaderObject* (Figure 16) is used to organize all the data:

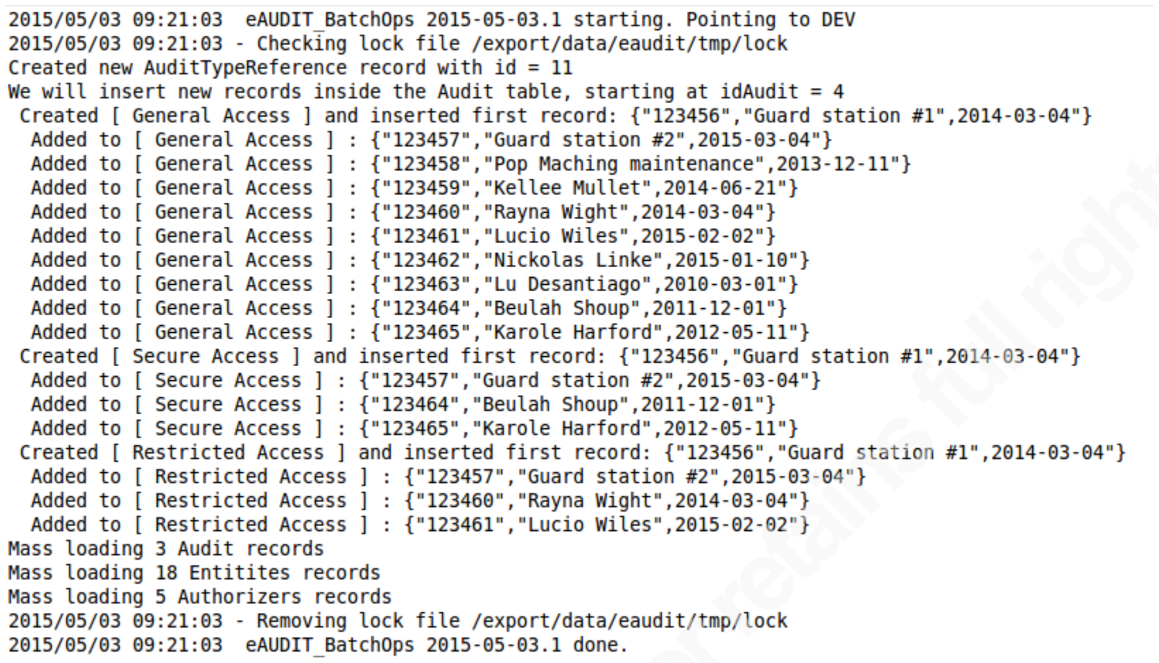


```
public class AuditDataLoaderObject {

    private int idAuditID;
    private int idAuditTypeReference;
    private Audit audit;
    private ArrayList<Authorizers> auditAuthorizers;
    private ArrayList<Entities> auditEntities;
```

Figure 16. Fields for java class AuditDataLoaderObject.

This object is a meta-object, which can hold all the key data: idAudit, the corresponding Audit record, a list of Authorizers associated with the Audit record, and all entities for that particular audit. Once the HashMap has been fully populated, a simple call to method *AuditDataLoaderObjectUtil.massLoad()* handles batched writes to the database. Compiling eAUDIT_BatchOps and running it with the **-loadFromAdapter** switch demonstrates a simple adapter (Figure 17):



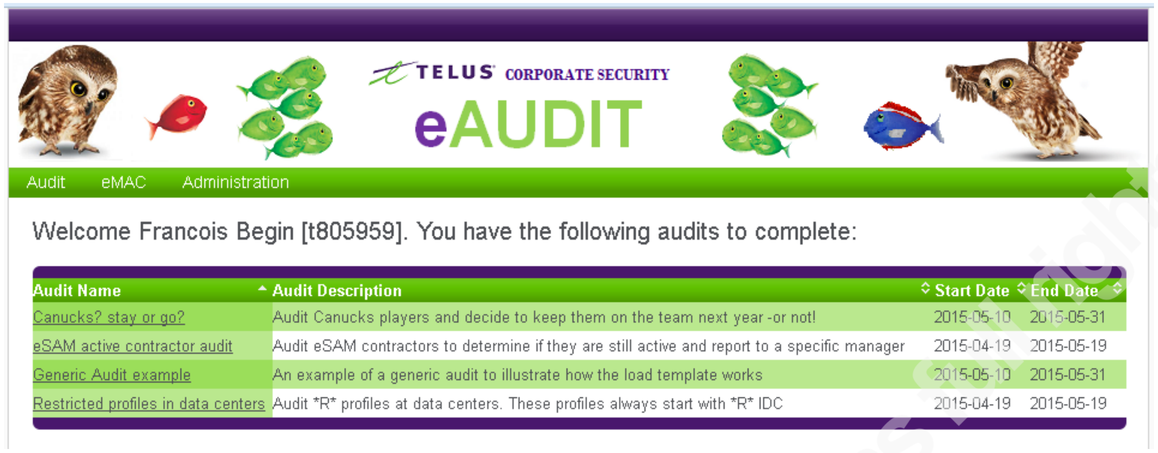
```
2015/05/03 09:21:03  eAUDIT_BatchOps 2015-05-03.1 starting. Pointing to DEV
2015/05/03 09:21:03 - Checking lock file /export/data/eaudit/tmp/lock
Created new AuditTypeReference record with id = 11
We will insert new records inside the Audit table, starting at idAudit = 4
 Created [ General Access ] and inserted first record: {"123456","Guard station #1",2014-03-04"}
  Added to [ General Access ] : {"123457","Guard station #2",2015-03-04"}
  Added to [ General Access ] : {"123458","Pop Maching maintenance",2013-12-11"}
  Added to [ General Access ] : {"123459","Kellee Mullet",2014-06-21"}
  Added to [ General Access ] : {"123460","Rayna Wight",2014-03-04"}
  Added to [ General Access ] : {"123461","Lucio Wiles",2015-02-02"}
  Added to [ General Access ] : {"123462","Nickolas Linke",2015-01-10"}
  Added to [ General Access ] : {"123463","Lu Desantiago",2010-03-01"}
  Added to [ General Access ] : {"123464","Beulah Shoup",2011-12-01"}
  Added to [ General Access ] : {"123465","Karole Harford",2012-05-11"}
 Created [ Secure Access ] and inserted first record: {"123456","Guard station #1",2014-03-04"}
  Added to [ Secure Access ] : {"123457","Guard station #2",2015-03-04"}
  Added to [ Secure Access ] : {"123464","Beulah Shoup",2011-12-01"}
  Added to [ Secure Access ] : {"123465","Karole Harford",2012-05-11"}
 Created [ Restricted Access ] and inserted first record: {"123456","Guard station #1",2014-03-04"}
  Added to [ Restricted Access ] : {"123457","Guard station #2",2015-03-04"}
  Added to [ Restricted Access ] : {"123460","Rayna Wight",2014-03-04"}
  Added to [ Restricted Access ] : {"123461","Lucio Wiles",2015-02-02"}
Mass loading 3 Audit records
Mass loading 18 Entitites records
Mass loading 5 Authorizers records
2015/05/03 09:21:03 - Removing lock file /export/data/eaudit/tmp/lock
2015/05/03 09:21:03  eAUDIT_BatchOps 2015-05-03.1 done.
```

Figure 17. Loading data into eAUDIT using an adapter.

2.5. Web interface implementation

Gathering data and normalizing it into a database is fine, but eAUDIT needs to present this data to authorizers to allow for an efficient review. Currently, the web interface implementation of eAUDIT exists as a proof-of-concept written in Java and deployed in a customized Spring framework commonly used by TELUS Chief Security Office (CSO). Obviously, this customized framework is infused with TELUS enterprise standards: it relies on TELUS's single sign-on (SSO) infrastructure, it has built-in security groups based on TELUS employee IDs, and it makes heavy use of TELUS-branded CSS. Many of these elements are either irrelevant to another organization or considered proprietary to TELUS and therefore will not be discussed here.

That being said, the core of the eAUDIT web implementation can be covered. This core is the audit interface, and to a lesser extent, the landing page for authorizers. Figure 18 shows the landing page for eAUDIT.

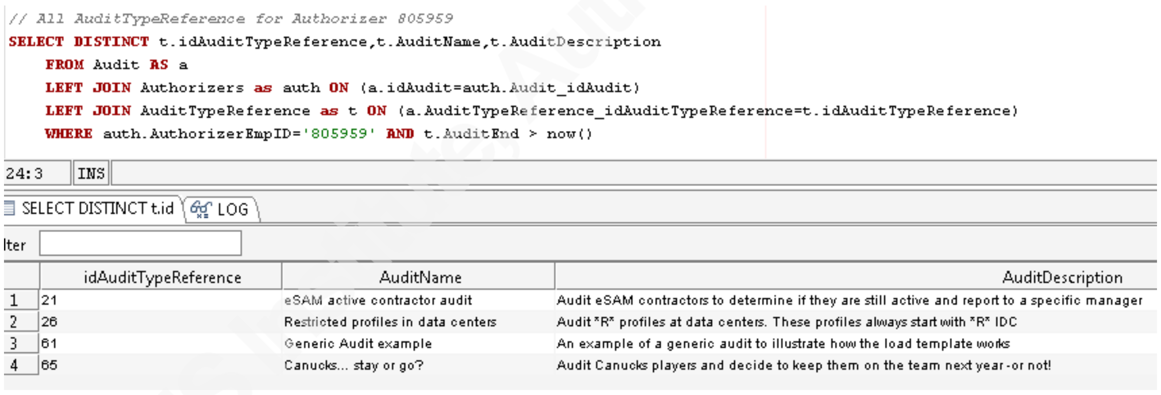


Welcome Francois Begin [t805959]. You have the following audits to complete:

Audit Name	Audit Description	Start Date	End Date
Canucks? stay or go?	Audit Canucks players and decide to keep them on the team next year -or not!	2015-05-10	2015-05-31
eSAM active contractor audit	Audit eSAM contractors to determine if they are still active and report to a specific manager	2015-04-19	2015-05-19
Generic Audit example	An example of a generic audit to illustrate how the load template works	2015-05-10	2015-05-31
Restricted profiles in data centers	Audit *R* profiles at data centers. These profiles always start with *R* IDC	2015-04-19	2015-05-19

Figure 18. eAUDIT web interface main landing page for an authorizer.

The landing page is a simple data table triggered when an authorizer logs in to the tool. A query is run using the employee ID of the authorizer as authenticated by SSO and returns audits that are currently active for that person (see Figure 19):



```
// All AuditTypeReference for Authorizer 805959
SELECT DISTINCT t.idAuditTypeReference,t.AuditName,t.AuditDescription
    FROM Audit AS a
    LEFT JOIN Authorizers as auth ON (a.idAudit=auth.Audit_idAudit)
    LEFT JOIN AuditTypeReference as t ON (a.AuditTypeReference_idAuditTypeReference=t.idAuditTypeReference)
    WHERE auth.AuthorizerEmpID='805959' AND t.AuditEnd > now()
```

	idAuditTypeReference	AuditName	AuditDescription
1	21	eSAM active contractor audit	Audit eSAM contractors to determine if they are still active and report to a specific manager
2	26	Restricted profiles in data centers	Audit *R* profiles at data centers. These profiles always start with *R* IDC
3	61	Generic Audit example	An example of a generic audit to illustrate how the load template works
4	65	Canucks... stay or go?	Audit Canucks players and decide to keep them on the team next year -or not!

Figure 19. SQL query to build the authorizer main landing page.

The names of the audits are links to the review engine. Note that the audits here are AuditTypeReference records, with a single such record potentially representing multiple entitlements of similar entities.

If a user selects a specific audit, this request is intercepted by the controller. Appendix E and F show the simple code for the controller, as well as the SQL query that retrieves Entities records for a particular AuditTypeReference record – and a particular authorizer. Entities data is attached as an attribute called *EntityList* to a ModelAndView object (See Code sample 1).

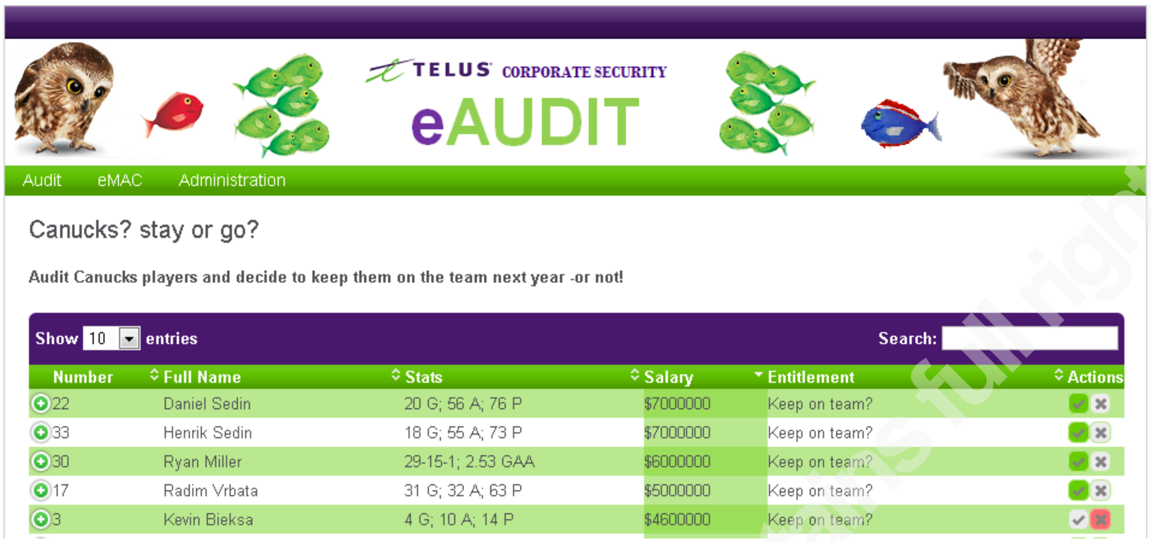
```
ModelAndView mv = new ModelAndView("audit");
[…]
mv.addObject("EntityList", auditDao.getEntitiesByAuditAndAuthorizer(auditID,
        SDDIHelper.getCurrentTeamMemberNumericalPIDFromRequest(request)));
```

Code sample 1. Attaching entity data to a ModelView.

Audit.jsp (Appendix G) is where this data gets presented to the authorizer. Fundamentally, this is a simple page that presents the audit name, description and a data table with the entities (Code sample 2 and Figure 20)

```
    <div><h3>${AuditDetail.auditName}</h3></div>
    <div><h5>${AuditDetail.auditDescription}</h5></div>
    <div id="tableContainer"
            style="margin-left:auto; margin-right:auto;
            width: auto;clear:both;">
      <table id="dataTable" style="width:100%;"></table>
    </div>
```

Code sample 2. Main section of audit.jsp.



Canucks? stay or go?

Audit Canucks players and decide to keep them on the team next year -or not!

Show 10 entries Search:

Number	Full Name	Stats	Salary	Entitlement	Actions
22	Daniel Sedin	20 G; 56 A; 76 P	$7000000	Keep on team?	✓ ✗
33	Henrik Sedin	18 G; 55 A; 73 P	$7000000	Keep on team?	✓ ✗
30	Ryan Miller	29-15-1; 2.53 GAA	$6000000	Keep on team?	✓ ✗
17	Radim Vrbata	31 G; 32 A; 63 P	$5000000	Keep on team?	✓ ✗
3	Kevin Bieksa	4 G; 10 A; 14 P	$4600000	Keep on team?	✓ ✗

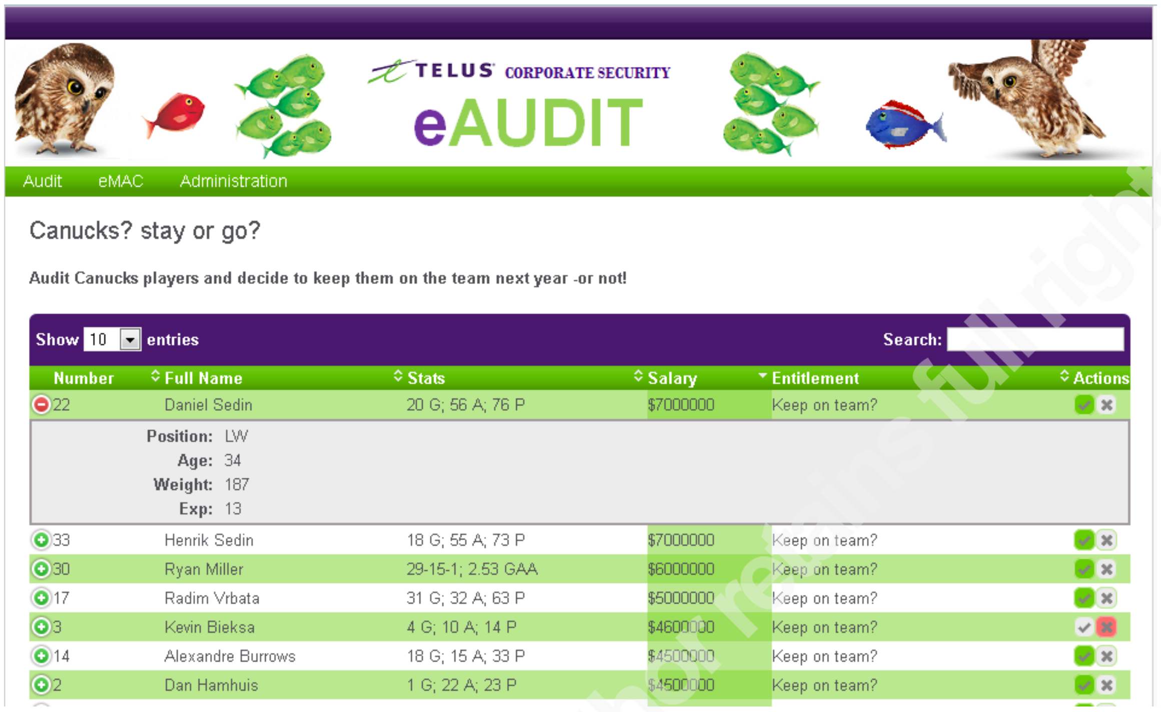
Figure 20. eAUDIT web interface audit review page.

The design decision to keep user-defined fields in JSON format was motivated by the fact that this data format can easily be manipulated. Javascript is used to generate the main audit data table on the fly. Although these fields are not easily or efficiently searchable by the system from the perspective of the whole dataset, once re-formatted in a data table, that data becomes easily sortable and filterable.

User-defined primary fields form visible rows while hidden rows hold the secondary attributes. Clicking on the details icon will trigger the hidden rows and present them to the authorizer (Code sample 3 and Figure 21):

```
    $('#dataTable tbody td img').click(function() {
        var nTr = this.parentNode.parentNode;
                if ( this.src.match('details_close') ){ /* row is already open - close it */
                        this.src =
                        "${pageContext.request.contextPath}/img/details_open.png";
                        oTable.fnClose(nTr);
                } else { /* Open this row */
                        this.src =
                        "${pageContext.request.contextPath}/img/details_close.png";
                        oTable.fnOpen(nTr,fnFormatDetails(oTable,nTr),'details');
                }
        });
```

Code sample 3. Trigger to display/hide secondary entity attributes.



Canucks? stay or go?

Audit Canucks players and decide to keep them on the team next year -or not!

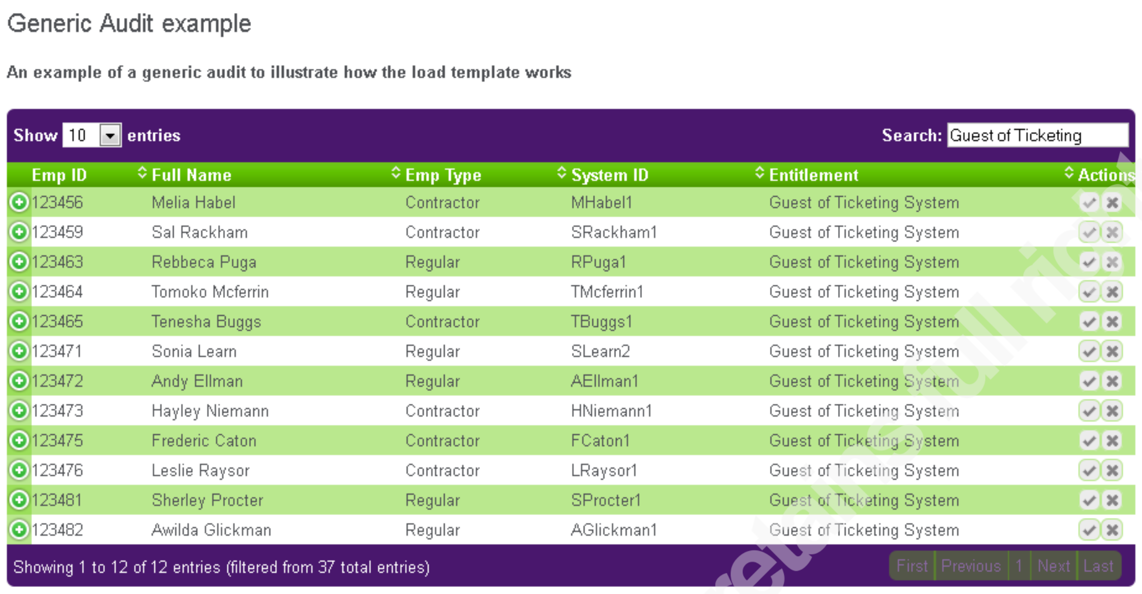
Show 10 entries | Search:

Number	Full Name	Stats	Salary	Entitlement	Actions
22	Daniel Sedin	20 G; 56 A; 76 P	$7000000	Keep on team?	✓ ✗
Position: LW Age: 34 Weight: 187 Exp: 13					
33	Henrik Sedin	18 G; 55 A; 73 P	$7000000	Keep on team?	✓ ✗
30	Ryan Miller	29-15-1; 2.53 GAA	$6000000	Keep on team?	✓ ✗
17	Radim Vrbata	31 G; 32 A; 63 P	$5000000	Keep on team?	✓ ✗
3	Kevin Bieksa	4 G; 10 A; 14 P	$4600000	Keep on team?	✓ ✗
14	Alexandre Burrows	18 G; 15 A; 33 P	$4500000	Keep on team?	✓ ✗
2	Dan Hamhuis	1 G; 22 A; 23 P	$4500000	Keep on team?	✓ ✗

Figure 21. Displaying secondary user-defined entity attributes in eAUDIT web.

The table auto-adjusts to the number of user-defined primary fields and to the number of user-defined secondary fields. There is obviously a physical limitation on the amount of screen real estate that one has access to, so the number of primary fields must be kept reasonable. Primary fields should be chosen amongst those fields that are most likely to clearly identify the entity to an authorizer, with secondary fields used occasionally by the authorizer to ascertain whether or not to confirm the entitlement. This confirmation is handled by action buttons at the end of each row.

As previously mentioned, using a data table has numerous advantages, including the ability to sort and filter the data. The filtering in particular can be used to focus on specific entitlements if multiple entitlements are being reviewed by the same authorizer within the same audit (see figure 22). Moreover, all of the work is offloaded to the client, leaving the server to simply process user requests.


Generic Audit example

An example of a generic audit to illustrate how the load template works

Show 10 entries

Search: Guest of Ticketing

Emp ID	Full Name	Emp Type	System ID	Entitlement	Actions
123456	Melia Habel	Contractor	MHabel1	Guest of Ticketing System	✓ ✗
123459	Sal Rackham	Contractor	SRackham1	Guest of Ticketing System	✓ ✗
123463	Rebbeca Puga	Regular	RPuga1	Guest of Ticketing System	✓ ✗
123464	Tomoko Mcferrin	Regular	TMcferrin1	Guest of Ticketing System	✓ ✗
123465	Tenesha Buggs	Contractor	TBuggs1	Guest of Ticketing System	✓ ✗
123471	Sonia Learn	Regular	SLearn2	Guest of Ticketing System	✓ ✗
123472	Andy Ellman	Regular	AEllman1	Guest of Ticketing System	✓ ✗
123473	Hayley Niemann	Contractor	HNiemann1	Guest of Ticketing System	✓ ✗
123475	Frederic Caton	Contractor	FCaton1	Guest of Ticketing System	✓ ✗
123476	Leslie Raysor	Contractor	LRaysor1	Guest of Ticketing System	✓ ✗
123481	Sherley Procter	Regular	SProcter1	Guest of Ticketing System	✓ ✗
123482	Awilda Glickman	Regular	AGlickman1	Guest of Ticketing System	✓ ✗

Showing 1 to 12 of 12 entries (filtered from 37 total entries)

First | Previous | 1 | Next | Last

Figure 22. eAUDIT search filter capability used to group entitlements.

Since most of the heavy lifting is handled client-side, the tool must ensure that authorizers can only review entities and entitlements for which they are authorized, and this verification is made server-side, prior to updating table EntitiesAuditResponses.

2.6. Future enhancement

eAUDIT is still a proof-of-concept at this stage and requires polishing. The data model supports the ability for authorizers to delegate an audit to someone else. Delegation will give a busy authorizer the ability to offload some of his audit work to a trusted direct report. Additionally, the TELUS implementation will implicitly provide access to the supervisors of authorizers. This means that anyone above a base audit authorizer will be a super-authorizer, able to view and delegate any of these audits – or let their subalterns know they need to pick up the pace! Both the delegation feature and the implied super-authorizers feature should help promote high audit completion rates.

Another reality of audits is that the data provided may be stale or inadequate at load time. For instance, an employee loaded as an entity may not have a current manager. If the manager is the authorizer of a particular entitlement, this employee will not be reviewed. eAUDIT will address this challenge by taking the company's organizational hierarchy into account when allowing entitlement review. It will provide reports to the audit administrator of these stale/incomplete records, allowing the administrator to review them himself, or re-assign them to someone who can.

Yet another feature of TELUS's planned eAUDIT implementation is the use of a *PrimaryKey* in the Entities table to provide historical correlation between audits. This will allow an authorizer to determine whether or not this entity's entitlement was reviewed or revoked the last time the audit ran.

Another important aspect of auditing that this paper did not cover is communication. A successful audit is one where authorizers are clearly informed of deadlines and reminded to meet them. The TELUS CSO has a communication tool called eMAC that is capable of handling audit communications (kickoff email, reminders, escalations) and eAUDIT will be tightly integrated with this tool to offer an end-to-end auditing solution.

3. Conclusion

This paper presented eAUDIT, a generic audit review tool to conduct entitlement reviews. This tool is currently being implemented at TELUS to mitigate some shortcomings of our identity and access management infrastructure. It should be noted that IAM is often seen through the lenses of compliance, and it is true that TELUS's audit tools predecessors have been compliance-driven. On the other hand, there are other solid business cases to be made with regard to auditing and managing access properly: operational effectiveness (or cost savings) as well as business enablement are other considerations (Osmanoglu, 2013). There is already evidence to support the proposition that eAUDIT will benefit TELUS beyond compliance.

François Bégin, francois.begin@telus.com

References

Allen, J. (2015). *Excel Is NOT a Database!* Retrieved from *http://adminsecret.monster.com/benefits/articles/2599-excel-is-not-a-database*.

Apache Foundation (2015). *Apache POI – The Java API for Microsoft Documents.* Retrieved from https://poi.apache.org/.

Ballad, B., Ballad, T. and Banks, E. (2010). *Access Control, Authentication, and Public Key Infrastructure*. Jones & Bartlett Learning.

Doraiswamy, P., Shiv, P. (2012). *50 Top IT Project Management Challenges.* IT Governance Ltd.

Jurney, R. (2013). *Agile Data Science.* O'Reilly Media Inc.

Kadrich, M. (2007). *Endpoint Security.* Addison-Wesley Professional.

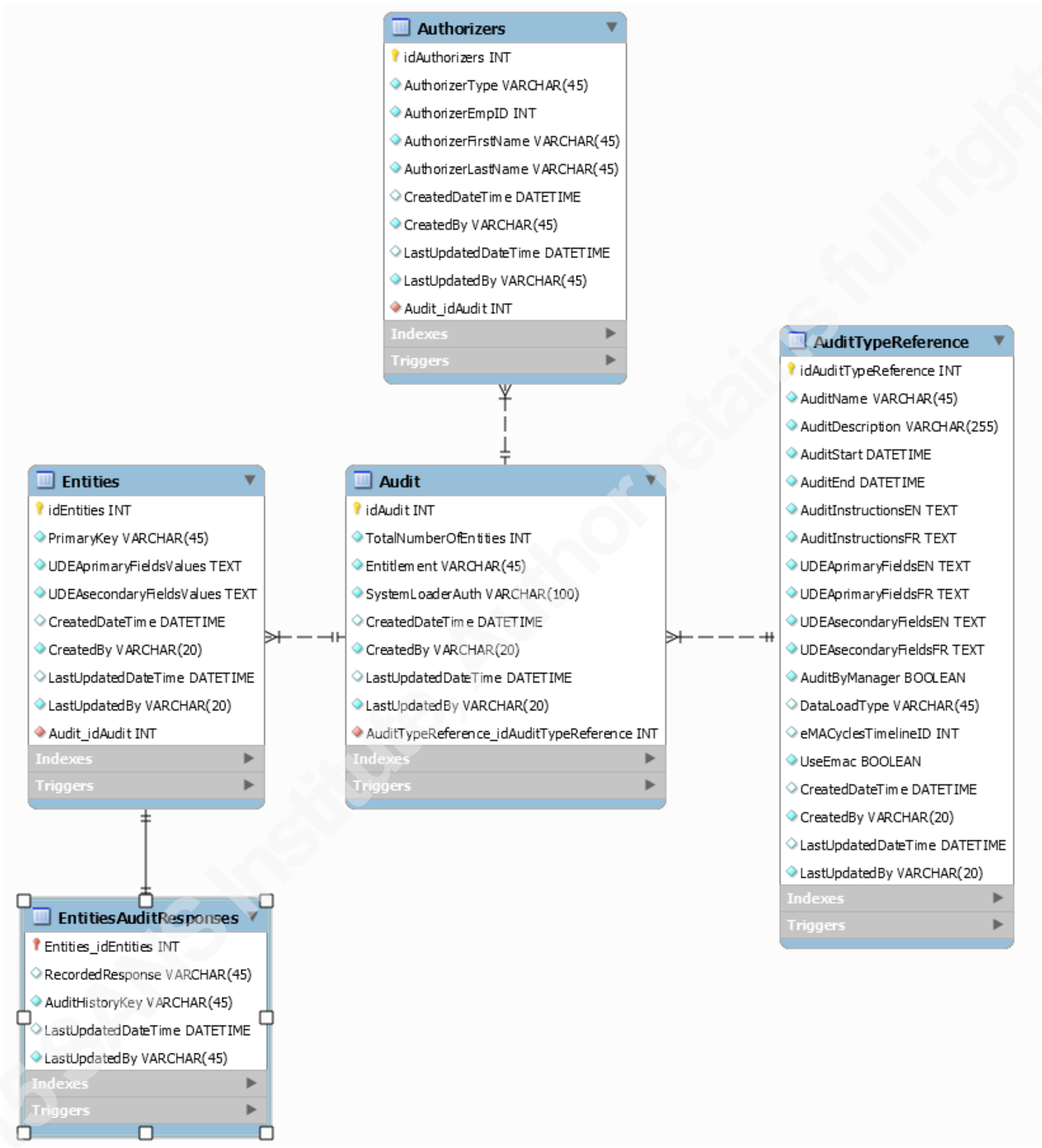
Mathers, T. Kumaraswamy, S. and Latif, S. (2009). *Cloud Security and Privacy.* O'Reilly Media Inc.

Osmanoglu, E. (2013). *Identity and Access Management.* Syngress Press.

Saleh, K. A. (2009). *Software Engineering.* J. Ross Publishing.

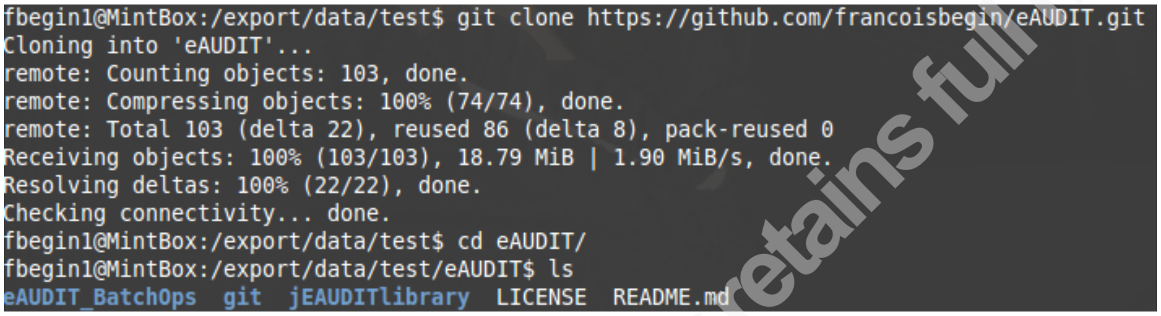
Walsh, K. (2014). *5 Pitfalls of Using Spreadsheets for Business Analytics.* Retrieved from http://www.bisoftwareinsight.com/5-pitfalls-spreadsheets-for-business-analytics/.

Appendix A – eAUDIT data model



Appendix B – Setting up eAUDIT for a demo

Here is a quick overview of how to set up an environment to run/demo eAUDIT's data acquisition capability. First, in a directory of your choice, clone the eAUDIT project code (Figure B-1).

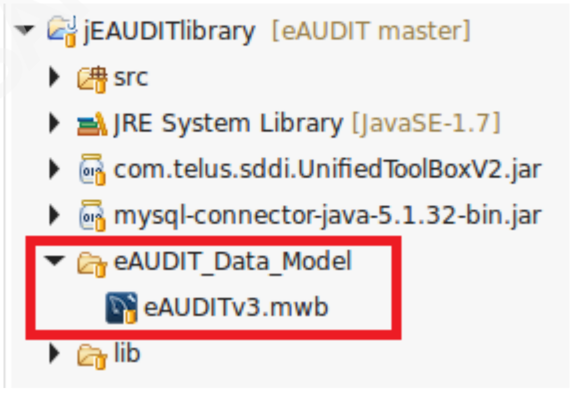


```
fbegin1@MintBox:/export/data/test$ git clone https://github.com/francoisbegin/eAUDIT.git
Cloning into 'eAUDIT'...
remote: Counting objects: 103, done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 103 (delta 22), reused 86 (delta 8), pack-reused 0
Receiving objects: 100% (103/103), 18.79 MiB | 1.90 MiB/s, done.
Resolving deltas: 100% (22/22), done.
Checking connectivity... done.
fbegin1@MintBox:/export/data/test$ cd eAUDIT/
fbegin1@MintBox:/export/data/test/eAUDIT$ ls
eAUDIT_BatchOps  git  jEAUDITlibrary  LICENSE  README.md
```

Figure B-1. Cloning eAUDIT code from GitHub.

There are two separate projects in the code pulled from GitHub: the *jEAUDITlibrary* itself, as well as *eAUDIT_BatchOps*, a project to demonstrate data acquisition. You can import these projects into your preferred Java IDE to examine the code.

Next, you need to set up a server to run the eAUDIT database. eAUDIT was built on MySQL and the data model has been included as a MySQL WorkBench file inside jEAUDITlibrary (Figure B-2). A service account with read-write access to the database also needs to be created



```
jEAUDITlibrary [eAUDIT master]
  src
  JRE System Library [JavaSE-1.7]
  com.telus.sddi.UnifiedToolBoxV2.jar
  mysql-connector-java-5.1.32-bin.jar
  eAUDIT_Data_Model
    eAUDITv3.mwb
  lib
```

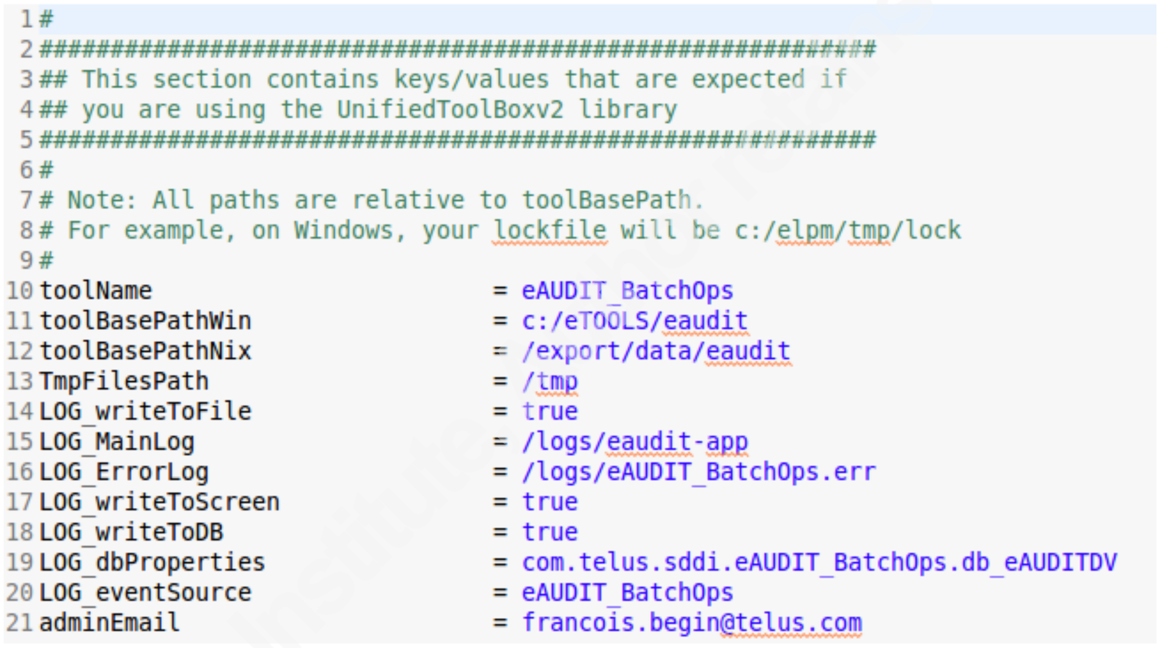
Figure B-2. MySQL WorkBench data model file for eAUDIT.

François Bégin, francois.begin@telus.com

Prior to running eAUDIT_BatchOps, a few configuration changes must be made:

1. Edit files *db_eAUDITDV* and *db_eAUDITPR*. Set the database name, user, password and URL to point to the database that was created to hold eAUDIT data.

2. Edit files *StaticParamsDV* and *StaticParamsPR* (Figure B-3). Set the path for the system where you will be trying out eAUDIT_BatchOps. Logging can also be adjusted through the various LOG_* parameters.



```
#
###########################################################
## This section contains keys/values that are expected if
## you are using the UnifiedToolBoxv2 library
###########################################################
#
# Note: All paths are relative to toolBasePath.
# For example, on Windows, your lockfile will be c:/elpm/tmp/lock
#
toolName                          = eAUDIT_BatchOps
toolBasePathWin                   = c:/eTOOLS/eaudit
toolBasePathNix                   = /export/data/eaudit
TmpFilesPath                      = /tmp
LOG_writeToFile                   = true
LOG_MainLog                       = /logs/eaudit-app
LOG_ErrorLog                      = /logs/eAUDIT_BatchOps.err
LOG_writeToScreen                 = true
LOG_writeToDB                     = true
LOG_dbProperties                  = com.telus.sddi.eAUDIT_BatchOps.db_eAUDITDV
LOG_eventSource                   = eAUDIT_BatchOps
adminEmail                        = francois.begin@telus.com
```

Figure B-3. Static parameters as defined in class StaticParamsDV/PR.

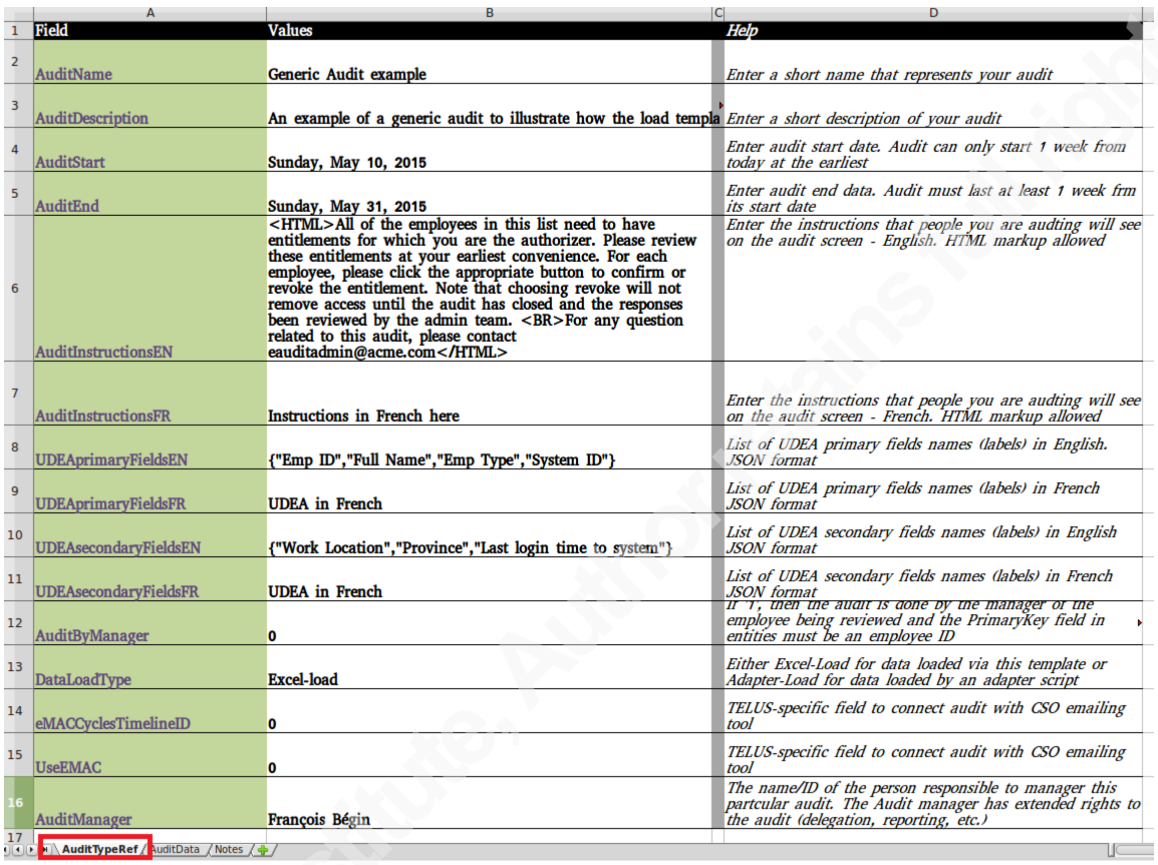
3. Create the *toolBasePath* directories. Assuming the demo is running on a Linux computer and the configuration file shown in Figure B-3, the following directories would need to be created:

    a. */export/data/eaudit*

    b. */export/data/eaudit/tmp*

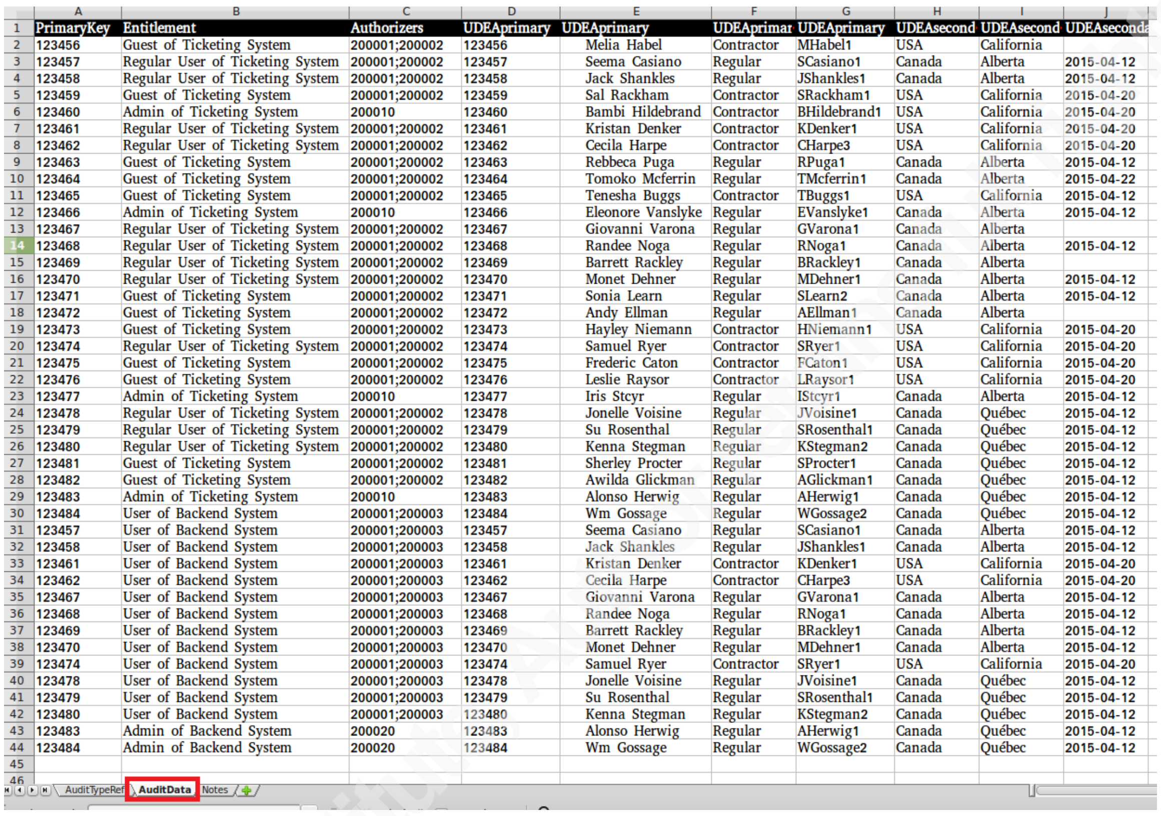
    c. */export/data/eaudit/logs*

Appendix C – AuditTypeRef sheet of the eAUDIT Excel template



	A	B	C	D
1	**Field**	**Values**		***Help***
2	AuditName	Generic Audit example		*Enter a short name that represents your audit*
3	AuditDescription	An example of a generic audit to illustrate how the load templa		*Enter a short description of your audit*
4	AuditStart	Sunday, May 10, 2015		*Enter audit start date. Audit can only start 1 week from today at the earliest*
5	AuditEnd	Sunday, May 31, 2015		*Enter audit end data. Audit must last at least 1 week frm its start date*
6	AuditInstructionsEN	<HTML>All of the employees in this list need to have entitlements for which you are the authorizer. Please review these entitlements at your earliest convenience. For each employee, please click the appropriate button to confirm or revoke the entitlement. Note that choosing revoke will not remove access until the audit has closed and the responses been reviewed by the admin team. <BR>For any question related to this audit, please contact eauditadmin@acme.com</HTML>		*Enter the instructions that people you are audting will see on the audit screen - English. HTML markup allowed*
7	AuditInstructionsFR	Instructions in French here		*Enter the instructions that people you are audting will see on the audit screen - French. HTML markup allowed*
8	UDEAprimaryFieldsEN	{"Emp ID","Full Name","Emp Type","System ID"}		*List of UDEA primary fields names (labels) in English. JSON format*
9	UDEAprimaryFieldsFR	UDEA in French		*List of UDEA primary fields names (labels) in French JSON format*
10	UDEAsecondaryFieldsEN	{"Work Location","Province","Last login time to system"}		*List of UDEA secondary fields names (labels) in English JSON format*
11	UDEAsecondaryFieldsFR	UDEA in French		*List of UDEA secondary fields names (labels) in French JSON format*
12	AuditByManager	0		*If '1', then the audit is done by the manager of the employee being reviewed and the PrimaryKey field in entities must be an employee ID*
13	DataLoadType	Excel-load		*Either Excel-Load for data loaded via this template or Adapter-Load for data loaded by an adapter script*
14	eMACCyclesTimelineID	0		*TELUS-specific field to connect audit with CSO emailing tool*
15	UseEMAC	0		*TELUS-specific field to connect audit with CSO emailing tool*
16	AuditManager	François Bégin		*The name/ID of the person responsible to manager this partcular audit. The Audit manager has extended rights to the audit (delegation, reporting, etc.)*
17				

Appendix D – AuditData sheet of the eAUDIT Excel template



	A	B	C	D	E	F	G	H	I	J
1	PrimaryKey	Entitlement	Authorizers	UDEAprimary	UDEAprimary	UDEAprimar	UDEAprimary	UDEAsecond	UDEAsecond	UDEAseconda
2	123456	Guest of Ticketing System	200001;200002	123456	Melia Habel	Contractor	MHabel1	USA	California	
3	123457	Regular User of Ticketing System	200001;200002	123457	Seema Casiano	Regular	SCasiano1	Canada	Alberta	2015-04-12
4	123458	Regular User of Ticketing System	200001;200002	123458	Jack Shankles	Regular	JShankles1	Canada	Alberta	2015-04-12
5	123459	Guest of Ticketing System	200001;200002	123459	Sal Rackham	Contractor	SRackham1	USA	California	2015-04-20
6	123460	Admin of Ticketing System	200010	123460	Bambi Hildebrand	Contractor	BHildebrand1	USA	California	2015-04-20
7	123461	Regular User of Ticketing System	200001;200002	123461	Kristan Denker	Contractor	KDenker1	USA	California	2015-04-20
8	123462	Regular User of Ticketing System	200001;200002	123462	Cecila Harpe	Contractor	CHarpe3	USA	California	2015-04-20
9	123463	Guest of Ticketing System	200001;200002	123463	Rebbeca Puga	Regular	RPuga1	Canada	Alberta	2015-04-12
10	123464	Guest of Ticketing System	200001;200002	123464	Tomoko Mcferrin	Regular	TMcferrin1	Canada	Alberta	2015-04-22
11	123465	Guest of Ticketing System	200001;200002	123465	Tenesha Buggs	Contractor	TBuggs1	USA	California	2015-04-12
12	123466	Admin of Ticketing System	200010	123466	Eleonore Vanslyke	Regular	EVanslyke1	Canada	Alberta	2015-04-12
13	123467	Regular User of Ticketing System	200001;200002	123467	Giovanni Varona	Regular	GVarona1	Canada	Alberta	
14	123468	Regular User of Ticketing System	200001;200002	123468	Randee Noga	Regular	RNoga1	Canada	Alberta	2015-04-12
15	123469	Regular User of Ticketing System	200001;200002	123469	Barrett Rackley	Regular	BRackley1	Canada	Alberta	
16	123470	Regular User of Ticketing System	200001;200002	123470	Monet Dehner	Regular	MDehner1	Canada	Alberta	2015-04-12
17	123471	Guest of Ticketing System	200001;200002	123471	Sonia Learn	Regular	SLearn2	Canada	Alberta	2015-04-12
18	123472	Guest of Ticketing System	200001;200002	123472	Andy Ellman	Regular	AEllman1	Canada	Alberta	
19	123473	Guest of Ticketing System	200001;200002	123473	Hayley Niemann	Contractor	HNiemann1	USA	California	2015-04-20
20	123474	Regular User of Ticketing System	200001;200002	123474	Samuel Ryer	Contractor	SRyer1	USA	California	2015-04-20
21	123475	Guest of Ticketing System	200001;200002	123475	Frederic Caton	Contractor	FCaton1	USA	California	2015-04-20
22	123476	Guest of Ticketing System	200001;200002	123476	Leslie Raysor	Contractor	LRaysor1	USA	California	2015-04-20
23	123477	Admin of Ticketing System	200010	123477	Iris Stcyr	Regular	IStcyr1	Canada	Alberta	2015-04-12
24	123478	Regular User of Ticketing System	200001;200002	123478	Jonelle Voisine	Regular	JVoisine1	Canada	Québec	2015-04-12
25	123479	Regular User of Ticketing System	200001;200002	123479	Su Rosenthal	Regular	SRosenthal1	Canada	Québec	2015-04-12
26	123480	Regular User of Ticketing System	200001;200002	123480	Kenna Stegman	Regular	KStegman2	Canada	Québec	2015-04-12
27	123481	Guest of Ticketing System	200001;200002	123481	Sherley Procter	Regular	SProcter1	Canada	Québec	2015-04-12
28	123482	Guest of Ticketing System	200001;200002	123482	Awilda Glickman	Regular	AGlickman1	Canada	Québec	2015-04-12
29	123483	Admin of Ticketing System	200010	123483	Alonso Herwig	Regular	AHerwig1	Canada	Québec	2015-04-12
30	123484	User of Backend System	200001;200003	123484	Wm Gossage	Regular	WGossage2	Canada	Québec	2015-04-12
31	123457	User of Backend System	200001;200003	123457	Seema Casiano	Regular	SCasiano1	Canada	Alberta	2015-04-12
32	123458	User of Backend System	200001;200003	123458	Jack Shankles	Regular	JShankles1	Canada	Alberta	2015-04-12
33	123461	User of Backend System	200001;200003	123461	Kristan Denker	Contractor	KDenker1	USA	California	2015-04-20
34	123462	User of Backend System	200001;200003	123462	Cecila Harpe	Contractor	CHarpe3	USA	California	2015-04-20
35	123467	User of Backend System	200001;200003	123467	Giovanni Varona	Regular	GVarona1	Canada	Alberta	2015-04-12
36	123468	User of Backend System	200001;200003	123468	Randee Noga	Regular	RNoga1	Canada	Alberta	2015-04-12
37	123469	User of Backend System	200001;200003	123469	Barrett Rackley	Regular	BRackley1	Canada	Alberta	2015-04-12
38	123470	User of Backend System	200001;200003	123470	Monet Dehner	Regular	MDehner1	Canada	Alberta	2015-04-12
39	123474	User of Backend System	200001;200003	123474	Samuel Ryer	Contractor	SRyer1	USA	California	2015-04-20
40	123478	User of Backend System	200001;200003	123478	Jonelle Voisine	Regular	JVoisine1	Canada	Québec	2015-04-12
41	123479	User of Backend System	200001;200003	123479	Su Rosenthal	Regular	SRosenthal1	Canada	Québec	2015-04-12
42	123480	User of Backend System	200001;200003	123480	Kenna Stegman	Regular	KStegman2	Canada	Québec	2015-04-12
43	123483	Admin of Backend System	200020	123483	Alonso Herwig	Regular	AHerwig1	Canada	Québec	2015-04-12
44	123484	Admin of Backend System	200020	123484	Wm Gossage	Regular	WGossage2	Canada	Québec	2015-04-12
45										
46										

François Bégin, francois.begin@telus.com

Appendix E – Controller that maps a request to an authorizer view of audit data

```
    @RequestMapping({
      "/audit/{auditID}"
})
 public ModelAndView displayAuditDetailsPage(@PathVariable String auditID)
      throws ObjectNotFoundException,
SQLException {
      ModelAndView mv = new ModelAndView("audit");
      //TODO: validate permission
      IAuditDao auditDao = getAuditDao();
      mv.addObject(USER_NAME,
      SDDIHelper.getCurrentTeamMemberNameFromRequest(request));
      mv.addObject(USER_TID,
      SDDIHelper.getCurrentTeamMemberTIDFromRequest(request));
      mv.addObject("AuditDetail", auditDao.getAuditDetailsByID(auditID));
      mv.addObject("EntityList", auditDao.getEntitiesByAuditAndAuthorizer(auditID,
      SDDIHelper.getCurrentTeamMemberNumericalPIDFromRequest(request)));
      return mv;
}
```

François Bégin, francois.begin@telus.com

Appendix F – SQL query getEntitiesByAuditIDAndAuthorizer to retrieve entities of a specific audit and specific authorizer

```
<statement id="getEntitiesByAuditIDAndAuthorizer"
resultMap="resultEntity"
parameterClass="map">
SELECT
        e.idEntities,e.PrimaryKey,
e.UDEAprimaryFieldsValues,
e.UDEAsecondaryFieldsValues,
a.Entitlement,
resp.RecordedResponse
            FROM Entities as e
            INNER JOIN Audit as a
ON (e.Audit_idAudit=a.idAudit
AND a.AuditTypeReference_idAuditTypeReference = #auditType#)
            LEFT JOIN Authorizers as auth
ON (auth.Audit_idAudit=a.idAudit)
            LEFT JOIN EntitiesAuditResponses as resp
ON e.idEntities = resp.Entities_idEntities
            WHERE auth.AuthorizerEmpID=#authorizerID#
        </statement>
```

François Bégin, francois.begin@telus.com

Appendix G – audit.jsp: main page of the eAUDIT review engine

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec"%>
<script>
var numOfPrimaryColumns = ${fn:length(AuditDetail.primaryFields)};
$(document).ready(function() {
        var rootPath = '${pageContext.request.contextPath}';
        var data = [];
        var secondaryFields = [
                        <c:forEach items="${AuditDetail.secondaryFields}"
        var="pfield2" varStatus="loop">
                                ${pfield2}<c:if test="${!loop.last}">,</c:if>
                        </c:forEach>
                ];

        <c:forEach items="${EntityList}" var="entity">
                data.push([ '<img
        src="${pageContext.request.contextPath}/img/details_open.png">',
                <c:forEach items="${entity.primaryFields}" var="fieldValue" >
                        ${fieldValue},
                </c:forEach>
                        "${entity.entitlement}",
                        '<div class="ui-corner-all list_icon text_icon grey action_yes"
        title="Keep"><span class="ui-icon ui-icon-check"></span></div><div
```

```
class="ui-corner-all list_icon text_icon grey action_no" title="Revoke"><span
class="ui-iconui-icon-closethick"></span></div>',
                '<div id="${entity.entityID}_details">' +
                <c:forEach items="${entity.secondaryFields}" var="fieldValue"
varStatus="loop">
                        '<div class="content_name">'+
secondaryFields[${loop.index}] +':</div><div class="content_value">' +
${fieldValue} + '</div>' +
                </c:forEach>
                '</div>'
                                ]);
</c:forEach>

var oTable = $('#dataTable').dataTable({
        bJQueryUI: true,
        "aaData": data,
        "aoColumns": [
                { "sTitle": "", "sWidth": "20px" },
                <c:forEach items="${AuditDetail.primaryFields}" var="pfield">
                { "sTitle": ${pfield} },
                </c:forEach>
                { "sTitle": "Entitlement", "sWidth": "250px" },
                { "sTitle": "Actions", "sWidth": "40px" },
                { "sTitle": "details" }
                        ],
        "aoColumnDefs": [
                                        { "bSearchable": false, "bSortable": false,
"aTargets": [ 0 ] },
                                { "bSearchable": false, "bSortable": false,
"aTargets": [ numOfPrimaryColumns + 2 ] },
                    { "bSearchable": false,"bVisible":
false,"bSortable": false, "aTargets": [ numOfPrimaryColumns + 3 ] }
               ],
     "iDisplayLength": 1000,
     sPaginationType : "full_numbers",
     oLanguage : {
          sProcessing : "<spring:message code="app.datatable.text1" />",
          sLengthMenu : "<spring:message code="app.datatable.text2" />",
          sZeroRecords : "<spring:message code="app.datatable.text3"
/>",
          sInfo : "<spring:message code="app.datatable.text4" />",
          sInfoEmpty : "<spring:message code="app.datatable.text5" />",
          sInfoFiltered : "<spring:message code="app.datatable.text6" />",
          sInfoPostFix : "<spring:message code="app.datatable.text7" />",
          sSearch : "<spring:message code="app.datatable.text8" />",
          sUrl : "<spring:message code="app.datatable.text9" />",
          oPaginate : {
               sFirst : "<spring:message code="app.datatable.text10"
/>",
               sPrevious : "<spring:message
code="app.datatable.text11" />",
               sNext : "<spring:message code="app.datatable.text12"
/>",
               sLast : "<spring:message code="app.datatable.text13"
/>",
          }
     }
});
```

```
/* Add event listener for opening and closing details       * Note that the indicator
for showing which row is open is not controlled by DataTables,       * rather it is
done here       */
$('#dataTable tbody td img').click(function() {
          //var nTr = $(this).parents('tr')[0];
          var nTr = this.parentNode.parentNode;
          if ( this.src.match('details_close') ){ /* This row is already open - close it
*/
                    this.src =
"${pageContext.request.contextPath}/img/details_open.png";
                    oTable.fnClose(nTr);
          } else { /* Open this row */
                    this.src =
"${pageContext.request.contextPath}/img/details_close.png";

                    oTable.fnOpen(nTr,fnFormatDetails(oTable,nTr),'details');
          }
});

$( "div.action_yes" ).mouseover(function() {
          $(this).addClass('green').removeClass('grey');
}).mouseout(function(){
          $(this).addClass('grey').removeClass('green');
}).click(function(){
          $(this).addClass('greenSelected').removeClass('grey');

          $(this).parent().find('div.action_no').removeClass('redSelected').addClass(
'grey');
});
```

```
        $( "div.action_no" ).mouseover(function() {
                $(this).addClass('red').removeClass('grey');
        }).mouseout(function(){
                $(this).addClass('grey').removeClass('red');
        }).click(function(){
                $(this).addClass('redSelected').removeClass('grey');

                $(this).parent().find('div.action_yes').removeClass('greenSelected').addCl
        ass('grey');
                });
});

/* Formating function for row details */
function fnFormatDetails(table, nTr) {
        var aData = table.fnGetData(nTr);
        return aData[numOfPrimaryColumns+3];
}
</script>

<div><h3>${AuditDetail.auditName}</h3></div>
<div><h5>${AuditDetail.auditDescription}</h5></div>
<div id="tableContainer" style="margin-left:auto; margin-right:auto; width:
        auto;clear:both;">
        <table id="dataTable" style="width:100%;"></table>
</div>
```