UT Business Contracts Application Documentation

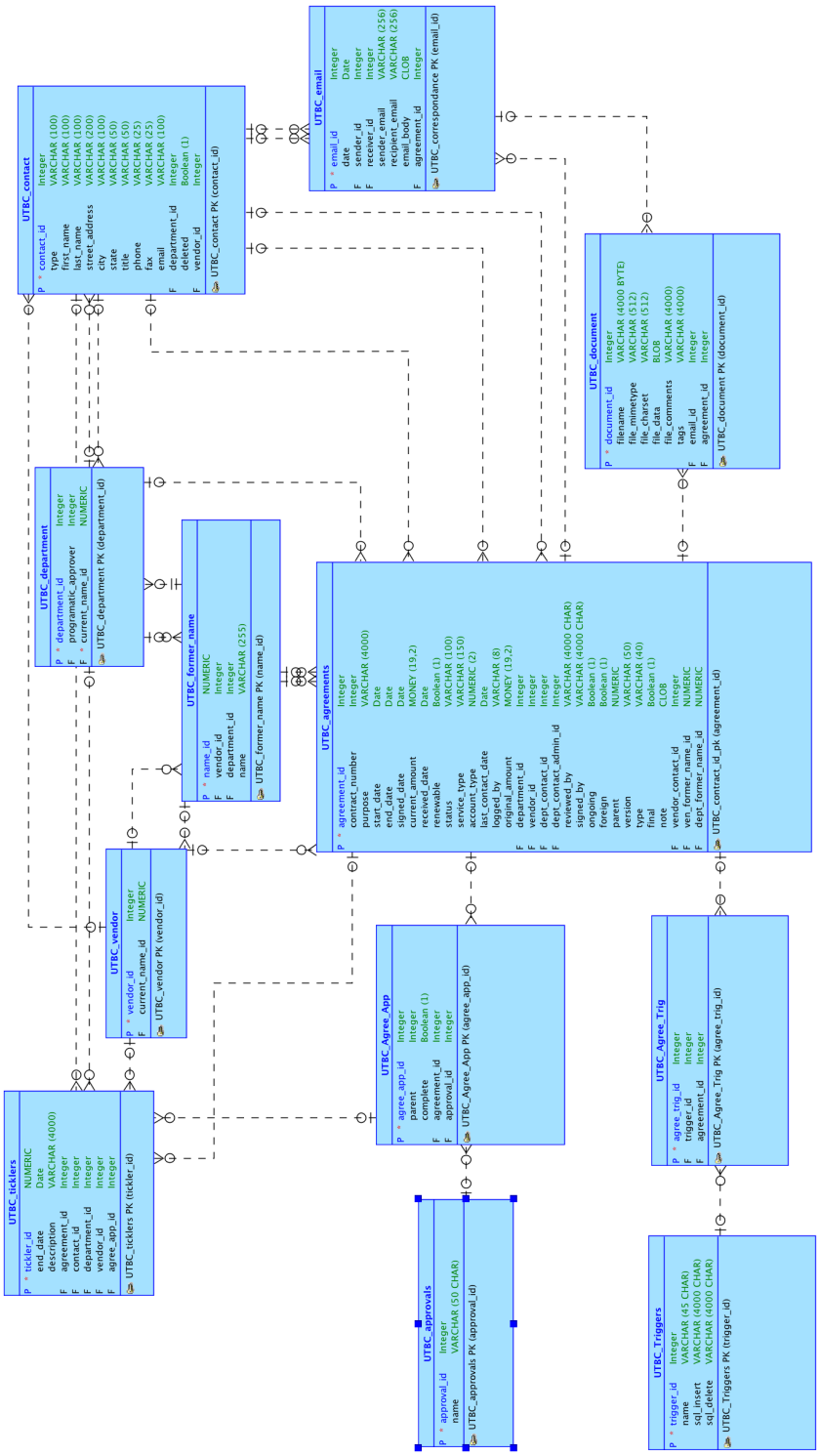
Colin Murray (Team Leader) Calvin Aisenbrey Alex Landaverde
Chris Metcalf Ashley Ng

August 12, 2014

Abstract

Put abstract here

Contents


UTBC_contact
UTBC_email
UTBC_department
UTBC_former_name
UTBC_agreements
UTBC_document
UTBC_vendor
UTBC_ticklers
UTBC_Agree_App
UTBC_Agree_Trig
UTBC_approvals
UTBC_Triggers

Chapter 1

Data model

1.1 Contract Table

1.2 UTBC_approvals and UTBC_Agree_App Tables

Chris Alex

1.3 UTBC_triggers and UTBC_Agree_Trig Tables

Chris Alex

Chapter 2

Application

2.1 Home Page

The home page consists of three different tables; contracts, ticklers, and BOR dates. The tables only displays contracts that are renewable and are expiring in the next 60 days. The table also only displays contracts that pertain to the user logged in using the where clause

Listing 2.1: Contracts table where clause for user

```
1 WHERE reviewed_by APEX_UTIL.GET_SESSION_STATE('APP_USER')
```

The next table is ticklers (or reminders), which only displays ticklers pertaining to the user that is logged in as well as public ticklers for everyone to see. We used the following where clause to do this

Listing 2.2: Tickler table where clause

```
1 WHERE t.created_by = APEX_UTIL.GET_SESSION_STATE('APP_USER')
2 OR PUBLIC_T = 'Y'
```

In this ticklers table, two tables are joined; the UTBC_ticklers and UTBC_contact tables. That way the user can easily email the corresponding person about the tickler. The next table is BOR dates (or board of regents dates). This table has no connect do our data model and is a stand alone table. The table has two attributes, due date and meeting date. The due date is the day the office will have to get certain paperwork submitted to be on the associated meeting date.

2.2 Agreements Home Page

2.2.1 Agreements Tree

Colin

2.2.2 Agreements Search Bar

Colin

2.2.3 Agreements Sorting

Calvin

2.2.4 Agreements Form

Ashley

2.3 Agreements Details Page

2.3.1 Purpose region

Calvin

2.3.2 Related Region

colin

2.3.3 Notes Region

calvin

2.3.4 Add Sub-Agreement, Add Version

The add sub-agreements button takes the agreement_id of the viewed agreement, and passes it to the item P23_agreement_parent on the form. The top level contract's agreement parent should be null because it has not parent, and setting the agreement parent relates the sub-agreement to it's parent contract.

There are two add version buttons, the first one allows you to create a new version based on the max version for the agreement. We pass five different values; the first one is we pass what the next version(&P4_Next_version) would be into P23_version. The &P4_next_version just uses a query to get the max version of the contract

Listing 2.3: P4_NEXT_value query

```
1 select MAX(VERSION)+1
2 from UTBC_AGREEMENTS
3 where VERSION_PARENT = (
4 select VERSION_PARENT
```

```
5 from UTBC_AGREEMENTS
6 where AGREEMENT_ID = :P4_AGREEMENT_ID)
```

The next item is the contract number, which connects the version to it's proper contract. The next assignment is the version parent, and this is just to keep things organized. The next item is the version_id value that get's it's value from &P4_max_agreement which uses a query to get what the agreement id of the max version.

Listing 2.4: P4_max_agreement query

```
1 select agreement_id
2 from UTBC_AGREEMENTS
3 where version = :P4_max_version
4 and version_parent = :P4_version_parent
```

The form then uses this value to populate itself. Normally a form would have an "apply changes button" if the primary key is not null. To work around this the &P4_blank value is null and that is passed into the P23_agreement_id, so that the version id can be the one to populate the form; this way the form is going to be a create.

Other version button

2.3.5 Approvals Region

Chris

2.4 Other Forms

2.4.1 Vendor Form

2.4.2 Department Form

2.4.3 Tickler Form

2.4.4 Document Form

The documents form uses a file drop plug-in that was downloaded at apex-plugin.com, and was created by Damien Antipa. The file drop plugin automatically does an insert into the table (as seen from the following code) once you drop the file into the file drop window. This causes an issue for the user because they have yet to finished completing the document form, or if the user uploaded the incorrect document, or decides to cancel the form. To get around this, the document_id was returned in the PL/SQL statement after the insert

Listing 2.5: PL/SQL on filedrop item

```
DECLARE
  c_collection_name constant varchar2(200):='CLOB_CONTENT';
  l_blobBLOB;
  l_mime                                    varchar2(50);
BEGIN
  SELECT apex_web_service.clobbase642blob(
        substr(clob001, ',')+1, length(clob001)))
  INTO l_blob
  from apex_collections
  where collection_name = c_collection_name

  insert into UTBC_document(FILE_DATA, filename,
        file_mimetype, AGREEMENT_ID)
  values(l_blob, wwv_flow.g_x01, wwv_flow.g_x02, :P28_agreement_id)
  returning document_id into :P28_document_id;
END;
```

If the user now decided that they want to cancel the form, a dynamic action fires when the user clicks the cancel button, and executes PL/SQL code

Listing 2.6: close dialog PL/SQL

```
BEGIN
delete from UTBC_document where document_id = :P28_document_id;
END;
```

Because the file browse and file drop both have different ways of inserting the blob into the table, two different create buttons were made. One handling the file browse and another handling the file drop, but only one is ever displayed at a time because of a IS NOT NULL or IS NULL condition on the item P28_FILENAME_SHOW. The item is initially null, and then will be filled with the text "*filename* uploaded" once the file drop has done the insert. One of the create buttons is the original create button that handles the file browse option. The other create button for the file drop option has PL/SQL code that fires when the corresponding create button is clicked.

Listing 2.7: update statement for filedrop

```
BEGIN
        update UTBC_document
        SEt
```

```
4           file_comments = :P28_file_comments,
5           tags = :P28_tags,
6           where document_id = :P28_document_id;
7   END;
```

At this point there is no indication when a file is successfully uploaded/inserted with the file drop. The was handled using a dynamic action event that came with the plug-in, upload ended. To do this, a dynamic action was created on the upload ended event on the file drop item. A false action was created to fire on page load, to hide the P28_FILENAME_SHOW item. Next, several true actions were set up, the first being to hide the file drop window. The next was to set this P28_FILENAME_SHOW with the filename. A cancel upload button will also be shown after upload. This button is incase the user has accidentally selected the incorrect file. When the button is clicked, a dynamic action fires that will delete the corresponding file, show the file drop region again, and then set the P28_FILENAME_SHOW item back to null.

2.5 Vendor, Department, and Contacts Page

These three pages are just interactive reports on their corresponding tables. The pencil icon allows you to edit the selected row with a modal. While, the magnifying glass takes you to a simple details page for the item. The state fields for the vendors and contacts are dynamic LOVs from the UTBC_state table, as well as the country field for the UTBC_country table. Our contacts table stores vendor and department contact so there is a type fields on the contact form that distinguishes the two.