|  |
| --- |
| Ultimate Software |
| SQL Framework Training |
| Interface Team |

|  |
| --- |
| Last Updated:  8-9-2019 |

Contents

[Chapter 0: Introduction 3](#_Toc19774451)

[Chapter 1: What is the Framework? Switchbox? 10](#_Toc19774452)

[Chapter 2: UltiPro Export Screens, Tables, and Queries 12](#_Toc19774453)

[Chapter 3: DSI Tables and SPs 19](#_Toc19774454)

[Chapter 4: Common Tables and Queries 21](#_Toc19774455)

[Chapter 5: Interface Developer Steps (New Projects) 31](#_Toc19774456)

[Chapter 6: Key Tables/Views Used in Exports 31](#_Toc19774457)

[Chapter 7: Framework Tips and Tricks 41](#_Toc19774458)

[Chapter 8: Scheduling Interfaces outside of the Web Portal 50](#_Toc19774459)

[Chapter 9: User Documentation 54](#_Toc19774460)

[Chapter 10: Interface Team SQL Standards (v2) 56](#_Toc19774461)

[Chapter 11: Web vs. Back Office Exports – Coding Differences 58](#_Toc19774462)

[Chapter 12: Upgrading Switchbox v1 to v2 60](#_Toc19774463)

[Chapter 13: After Collect Stored Procedure 62](#_Toc19774464)

[Chapter 14: Reporting from Platform Configurable Fields 64](#_Toc19774465)

# Chapter 0: Introduction

**What are Interfaces?**

An interface creates a bridge between our database and any tool, system, application, or company outside of Ultimate Software. An interface can send data out of an UltiPro database or import into an UltiPro database.

**Terms We Use Daily:**

|  |  |
| --- | --- |
| **Term** | **Meaning** |
| SOW/Spec | Statement of Work, or Spec documents. |
| IA | Integration Analyst. Our trusted friends who gather requirements and create the Spec documents. Previously referred to as BA (Business Analyst). |
| PIA | Primary Integration Analyst. Previously referred to as PBA. |
| SC | System Consultant / Functional Consultant. |
| PM | Project Manager. |
| QA | Quality-assurance testing, usually done by the IA or PIA. |
| UAT | User-Acceptance Testing. |
| UDES | UltiPro Data Exchange Service. They set up processes to exchange data between Ultimate Software and third parties. |
| I Drive | An internal folder on Ultimate’s server to which the customer has access. Files are not meant to be transmitted out of Ultimate. |
| J Drive | Frequently also referred to as the **UDES drive**. Files saved in the “Exports” folder in the J drive are meant to be transmitted out of Ultimate or be available for download via SFTP. An UDES rule must be set up in order for the transmission to take place.  J Drive paths will contain the phrase “data\_exchange”. |
| BDM | Benefit Deduction Module. Designed to standardize and simplify how benefit deductions are processed in exports. |
| 834 File | A standard file format for electronically exchanging health plan enrollment data. |
| UCN | UltiPro Carrier Network. For a large list of vendors and common file layouts, we can use a tool called JARVIS to quickly customize and deploy an export. |
| USI | UltiPro Standard Interfaces. A tool that allows us to configure an 834 file from a standard template. We have templates for many vendors. |
| Switchbox | Designed to standardize and simplify how custom interfaces are processed. Features include activity tracking/logging, automatic data manipulation, file generation and conversions. |

**SQL Crash Course:**

JobCode (Table):

|  |  |  |  |
| --- | --- | --- | --- |
| JbcJobCode | JbcDesc | JbcSalaryGrade | … |
| A001 | Customer Service Rep | C1 | … |
| B300 | Store Manager | M2 | … |
| B250 | President | E8 | … |

* **SELECT:**

SELECT JbcJobCode, JbcDesc, JbcSalaryGrade  
FROM dbo.JobCode WITH (NOLOCK)  
WHERE JbcJobCode = 'A001';

* **UPDATE:**

UPDATE dbo.JobCode  
SET JbcDesc = 'Customer Service Representative'

WHERE JbcJobCode = 'A001';

* **DELETE:**

DELETE FROM dbo.JobCode  
WHERE JbcJobCode = 'B300';

* **BEGIN TRAN / ROLLBACK TRAN / COMMIT TRAN:**

BEGIN TRAN T1  
DELETE FROM dbo.JobCode;

ROLLBACK TRAN T1;

-----------------------------------------

BEGIN TRAN T2  
DELETE FROM dbo.JobCode   
WHERE JbcJobCode = 'B300';

COMMIT TRAN T2;

EmpComp (Table):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EecEmpNo | EecEmplStatus | EecJobCode | EecEEID | … |
| 11111 | A | A001 | BK33T40000K0 | … |
| 22222 | A | A001 | BK33T40000K0 | … |
| 33333 | T | B300 | BK33T40000K0 | … |
| 44444 | A | B300 | BK33T601O0K0 | …. |

Contacts (View):

|  |  |  |  |
| --- | --- | --- | --- |
| ConEEID | ConRelationship | ConSSN | … |
| BEGOODATSQL1 | SPS | 999999991 | … |
| BEGOODATSQL2 | SPS | 999999992 | … |

* **Joins**

SELECT EecEmpNo, EecJobCode, JbcDesc

FROM dbo.EmpComp WITH (NOLOCK)

JOIN dbo.JobCode WITH (NOLOCK) ON JbcJobCode = EecJobCode  
WHERE EecEmplStatus = 'A';

This will return everyone in EmpComp, regardless of whether they have a spouse:

SELECT EecEmpNo, ConSSN AS SpouseSSN

FROM dbo.EmpComp WITH (NOLOCK)

LEFT JOIN dbo.Contacts WITH (NOLOCK) ON ConEEID = EecEEID

AND ConRelationship = 'SPS';

This will only return those who have a spouse:

SELECT EecEmpNo, ConSSN AS SpouseSSN

FROM dbo.EmpComp WITH (NOLOCK)

JOIN dbo.Contacts WITH (NOLOCK) ON ConEEID = EecEEID  
AND ConRelationship = 'SPS';

* **CASE Statement**

CASE WHEN EecEmplStatus = 'A' THEN 'Active' ELSE 'Inactive' END

CASE EecEmplStatus  
WHEN 'A' THEN 'Active'  
WHEN 'L' THEN 'On Leave'  
WHEN 'T' THEN 'Terminated'  
END

* **Stored Procedure:**

A set of SQL statements to do many great things – creating and populating tables, performing calculations or complex logic, updating tables, reading or producing files, etc.

* **Function:**

Performs an action and returns a value. Some common functions we use include:

GETDATE() -- returns the current date and time

CONVERT(VARCHAR,GETDATE(),112) -- returns today formatted as YYYYMMDD

CAST(num AS VARCHAR(10)) –- casts a numeric value to a string

ISNUMERIC(str) –- returns 1 if string is numeric, 0 otherwise

ISNULL(value,'') –- returns the second value if the first value is NULL

NULLIF(str,'Z') -- returns NULL if str is equal to 'Z'

COALESCE(value,'','') –- returns the "next" value if the first value is NULL

RTRIM(str) –- trim spaces on the right

LTRIM() –- trim spaces on the left

SUBSTRING(str,start,length) –- grab a partial string

SUBSTRING('united states',3,2)

LEFT(str,X) –- grab the left X characters of a string

RIGHT(str,X) –- grab the right X characters of a string

LEN(str) -- return length of a string

REPLACE(str,',','') -- replace all commas with blank

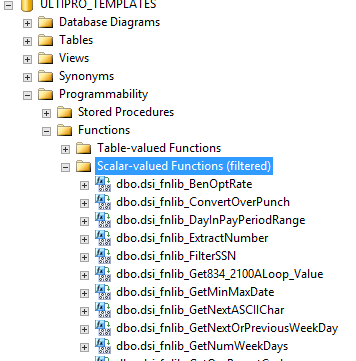
CHARINDEX('c',str) -- returns the first position of character(s) in string, or 0 if not found.

SUM(field) -- sum values in field from multiple records.

MIN(field) – return minimum in field from multiple records.

MAX(field) – return maximum in field from multiple records.

We also have a lot of custom functions that we use in our interfaces. They’re all stored in ULTIPRO\_TEMPLATES. The ones officially added in the function library will contain the word “fnlib” in the function name.



* **SQL Operators**

LIKE 'J%'

NOT LIKE 'John%'

BETWEEN @StartDate AND @EndDate, BETWEEN @MinNum AND @MaxNum

EXISTS(subquery)

NOT EXISTS(subquery)

WHERE EepNameMiddle IS NULL

IS NOT NULL

* **Compare Values**

We should always compare data of the same type or insert/update into a column data of the same type.

The statement below might work most of the time but is not a good practice. What if @SalaryString does not contain a numeric value?

IF @SalaryString >= 11880.00 THEN PRINT 'Above Poverty'

The statement below is better because we ensure that @SalaryString contains a numeric value. If it doesn’t, we would default it to 0.00 before comparing.

IF CAST(

(CASE ISNUMERIC(@SalaryString) = 1 THEN @SalaryString  
ELSE '0.00' END)

AS NUMERIC(10,2)) >= 11880.00 THEN PRINT 'Above Poverty'

Must handle NULL values properly.

WHERE EecLeaveReason <> 'SALCON'

vs.

WHERE ISNULL(EecLeaveReason,'') <> 'SALCON'

Be careful when comparing strings

'50000.00' and '50000' are two different string values.

* **String Concatenation**

Again, always check for NULL values. In addition, trim spaces before concatenation.

LTRIM(RTRIM(ISNULL(EepNameFirst,'')) + ' ' + LTRIM(RTRIM(ISNULL(EepNameLast,'')));

* **WHILE Loop**

WHILE @LoopCount <= @RowCount

BEGIN

-- Some logic that needs to happen in each iteration

SET @LoopCount = @LoopCount + 1;

END;

* **Subqueries**

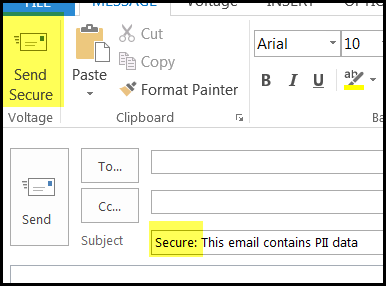
Make sure your subqueries only return one value, by using MAX, MIN, TOP 1 and ORDER BY.

drvPayDate = (SELECT MAX(PdhPayDate) FROM dbo.PDedHist WITH (NOLOCK))

drvCellPhone = (SELECT TOP 1 EfoPhoneNumber FROM dbo.EmpMPhon WITH (NOLOCK) WHERE EfoEEID = xEEID AND EfoIsPrivate = 'N' ORDER BY AuditKey DESC)

* **Code Format**
* SQL reserved words should be capitalized
  + SELECT, FROM, WHERE, ISNULL, CONVERT, etc.
* Database objects should use camelCase
  + drvFirstName, drvAddressLine1, etc.
* All database objects should include the schema with the object name
  + dbo.Company
* Table Hints – It’s recommended to use WITH (NOLOCK) on all SELECT queries to improve performance.
  + SELECT \* FROM dbo.Contacts WITH (NOLOCK);
* Terminate all SQL statements with a semi-colon.
* **PII Info**

Never send an email with personal, sensitive information. Always encrypt files or send them in secure emails. For Ultimate Software employees, be sure to include “Secure:“ in front of the email subject and click on the “Send Secure” button to force the email to be sent as a secure email.



# Chapter 1: What is the Framework? Switchbox?

The framework is the foundation code for the interfaces we build in UltiPro. A framework export can easily be identified by a call to the Switchbox stored procedure. The framework ties a number of tables and stored procedures together to work towards creating an interface to or from UltiPro.

Difference between the framework and switchbox – Sometimes these terms can be swapped when referring to custom code. Just to clarify, the framework is the whole foundation. The swithchbox is just one of the stored procedures. It is a pretty important one, but more of a gatekeeper.

Switchbox – The switchbox is the initial stored procedure that is called. The switchbox code is built to remove the redundant code from an interface. As an example the switchbox does a the following tasks

* The BCP statement
* 834 file formatting
* Reads in the UltiPro export screen fields
* Manages the file name and the output location
* Converts files to UTF-16 or UTF-8 format
* Consistent interface locator (Formatcode)
* Automatic Data Manipulation – Sorting, Uppercase, Override Record Counts
* Create flexibility with export sessions (Exportcode)

**Tables referenced**

AscDefH – Interface format definition (one record per interface)

AscDefF – File field definition (length, formatting, etc. – multiple records per interface)

AscExp – Export session information (multiple records per interface)

U\_dsi\_Configuration - Main Configuration Table

U\_dsi\_SqlClauses - Tables used for associated RecordSet

U\_dsi\_RecordSetDetails - Used to skip RecordSets by Export Code

U\_dsi\_Translations\_v2 – Used for translation/mapping

U\_dsi\_CompList\_v2 – Company filter list

U\_dsi\_Parameters – Custom table storing the last execution parameter data

U\_<formatcode>\_File – custom table storing the export data

U\_dsi\_InterfaceActivityLog - Locking Mechanism, Retain last 180 days of execution data

U\_dsi\_InterfaceProcesses - Track each process 'step' within Switchbox

**Stored Procedures**

U\_dsi\_DataDefinition\_v2 – Build INSERT statement into \_File table based on AscDefF (data definition).

U\_dsi\_CreateRecordSet\_v2 – Builds FROM/WHERE Clause and Sorting of Records. Calls dbo.dsi\_sp\_DataDefinition\_v2 SP.

U\_dsi\_Collect\_v2 – Handles skipping of RecordSets, calls dbo.dsi\_sp\_CreateRecordSet\_v2 SP.

U\_dsi\_builddrivertables\_FORMATCODE – Your stored procedure with the resulting formatcode.

U\_dsi\_sp\_TestSwitchBox\_v2 – Used by Custom Interface Team for running a Custom Interface by FormatCode, ExportCode in SSMS. When you are calling the switchbox in an executable line (like for scheduling) or just running the file via SQL, you just need the formatcode and export code defined. If you had to call the switchbox itself, there would be a lot more variables you need to define. (Don’t let the name fool you).

**Formatcode and Exportcode**

Formatcode – The unique identifier for the export across all the Framework tables. Vendor specific not client specific. If this is an export the formatcode must start with an E. If an import then I, and an update process then U. MUST USE ALL CAPS.

Exportcode – Provides flexibility within the formatcodes. Essentially is another session that provides variability for the user to use. Common exports codes are OEACTIVE/OEPASSIVE, SCHEDULED/ONDEMAND, PRODUCTION/TEST, FULL/CHANGES, and sessions for different GL accruals.

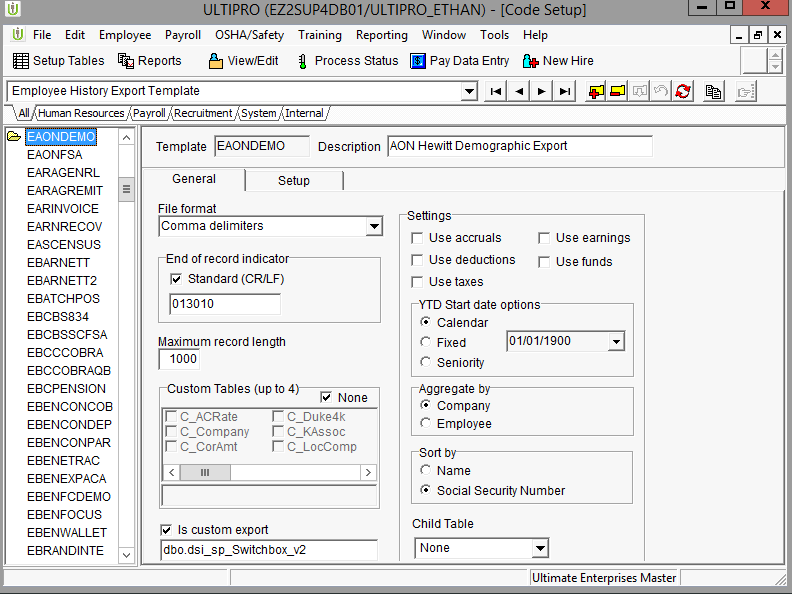
**Example: Concur Export**

****

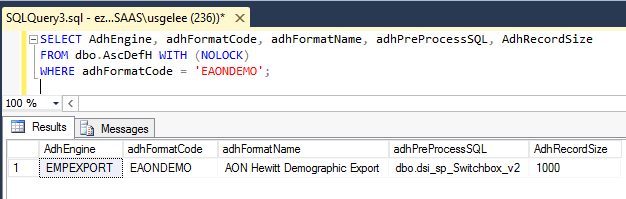
# Chapter 2: UltiPro Export Screens, Tables, and Queries

**Setup Tables:**

**Employee History Export Template – General Tab**



**Employee History Export Template – General Tab Table and Query**



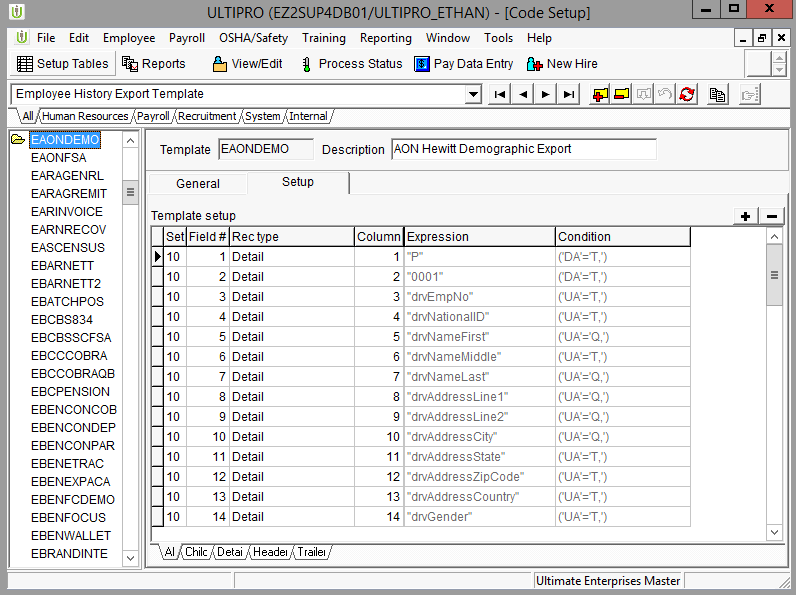
**Query**

SELECT AdhEngine, adhFormatCode, adhFormatName, adhPreProcessSQL, AdhRecordSize

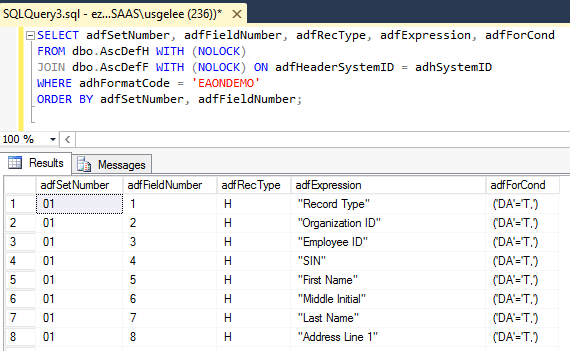
FROM dbo.AscDefH WITH (NOLOCK)

WHERE adhFormatCode = 'EAONDEMO';

**Employee History Export Template – Setup Tab**



**Employee History Export Template – Setup Tab Table and Query**



**Query**

SELECT adfSetNumber, adfFieldNumber, adfRecType, adfExpression, adfForCond

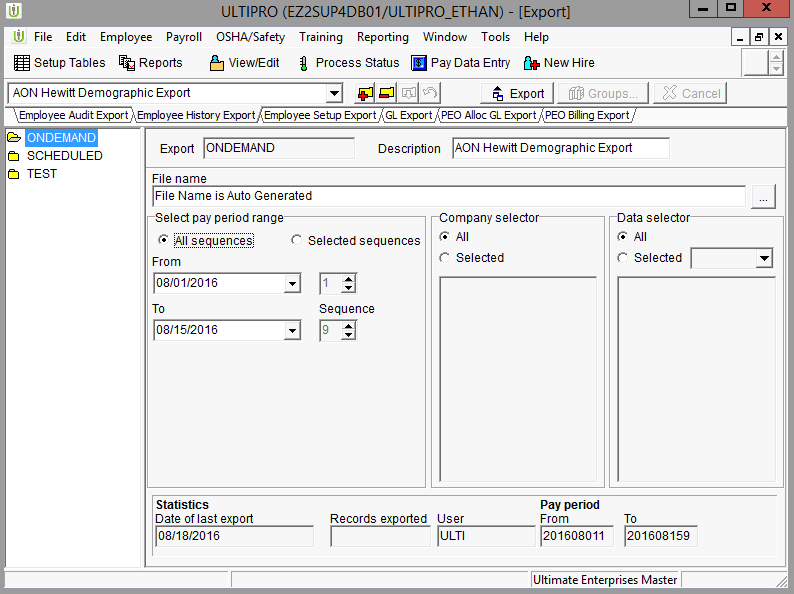
FROM dbo.AscDefH WITH (NOLOCK)

JOIN dbo.AscDefF WITH (NOLOCK) ON adfHeaderSystemID = adhSystemID

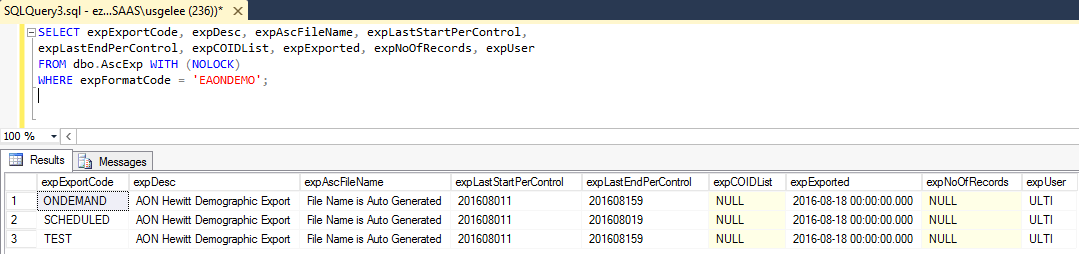
WHERE adhFormatCode = 'EAONDEMO'

ORDER BY adfSetNumber, adfFieldNumber;

**Export > Employee History Export - Screen**



**Export > Employee History Export - Query and Table**



**Query**

SELECT expExportCode, expDesc, expAscFileName, expLastStartPerControl,

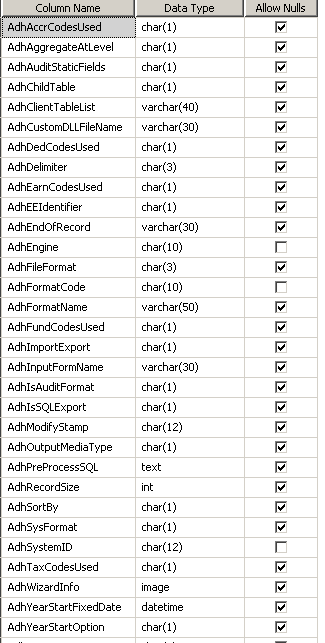
expLastEndPerControl, expCOIDList, expExported, expNoOfRecords, expUser

FROM dbo.AscExp WITH (NOLOCK)

WHERE expFormatCode = 'EAONDEMO';

**Table Links**

* AscDefH
  + Holds general information about export such as file format (Delimited, Fixed Length)
  + Link – AdhSystemID and AdhFormatCode



* AscDefF
  + Holds export file layout information
  + Link – AdfHeaderSystemID



* AscExp
  + Holds export session information
    - Export Session
    - Date/Percontrol parameters
    - Employee selection criteria
  + Link – ExpExportCode, ExpSystemID and ExpFormatCode

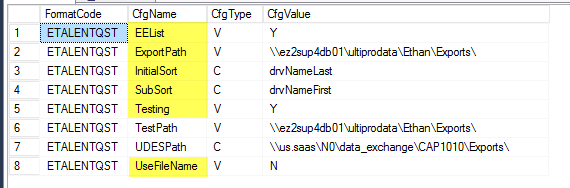


# Chapter 3: DSI Tables and SPs

**Tables**

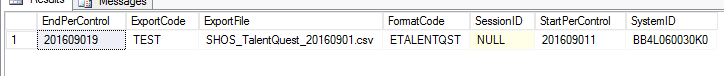
U\_dsi\_Configuration

* Table holding specific export configuration details. Can be used by the developer for additional items like test file path, last time export ran, and file count. 6 Required Entries highlighted.



U\_dsi\_Parameters

* Holds export session parameters. Updates during run time and will only have 1 record per formatcode
  + Dates (percontrols)
  + File name
  + ExportCode used



U\_dsi\_RecordSetDetails

* Indicates if record set is to be used and if the record set is a change or full record set

U\_dsi\_SelectByList

* Houses the values passed in from the data selector on the export screen.

U\_dsi\_SQLClauses

* Houses the table being used for a specific record set

U\_”Format Code”\_File

* Houses file data output, ready for file. The file table is the last stop and can be updated with an AfterCollect. A view will reference this table using the sorts in U\_dsi\_Configuration.

U\_dsi\_InterfaceActivityLog

* Used mainly for the locking mechanism to allow simultaneous runs for payroll automation.
* Run details. Great to use for debugging and troubleshooting

U\_dsi\_InterfaceProcesses

* Tracks each process "step" within the switchbox.
* Added for additional debugging and troubleshooting.

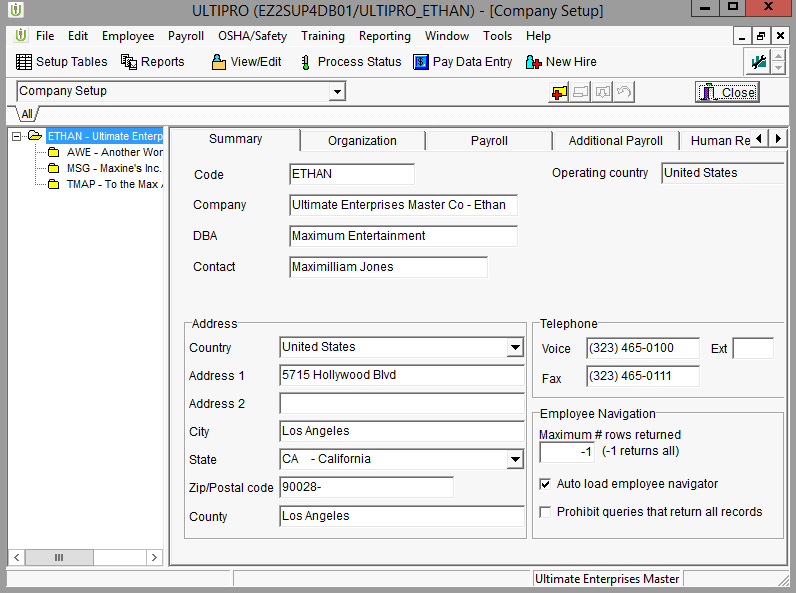
# Chapter 4: Common Core UltiPro Tables and Queries

* Company
  + Contains company setup information
  + CmpCOID - Unique Company ID

SELECT \*

FROM dbo.Company WITH (NOLOCK);

*Company Table accessed by application*

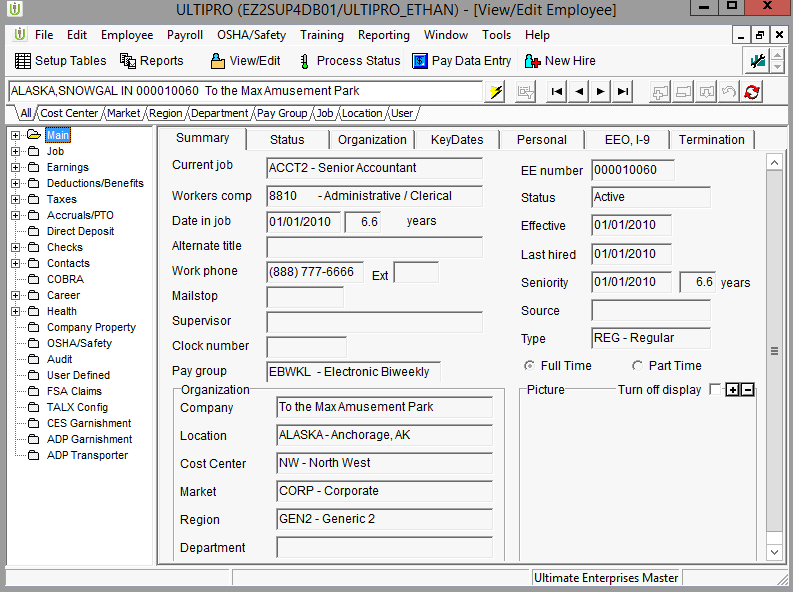


* EmpComp
  + Contains Employee company information
  + Multiple records can exist (EE existing in multiple companies)
  + Link
    - EecEEID – Unique Employee ID assigned by the system
    - EecCOID – Unique Company ID

SELECT \*

FROM dbo.EmpComp WITH (NOLOCK);

*EmpComp table accessed by application*

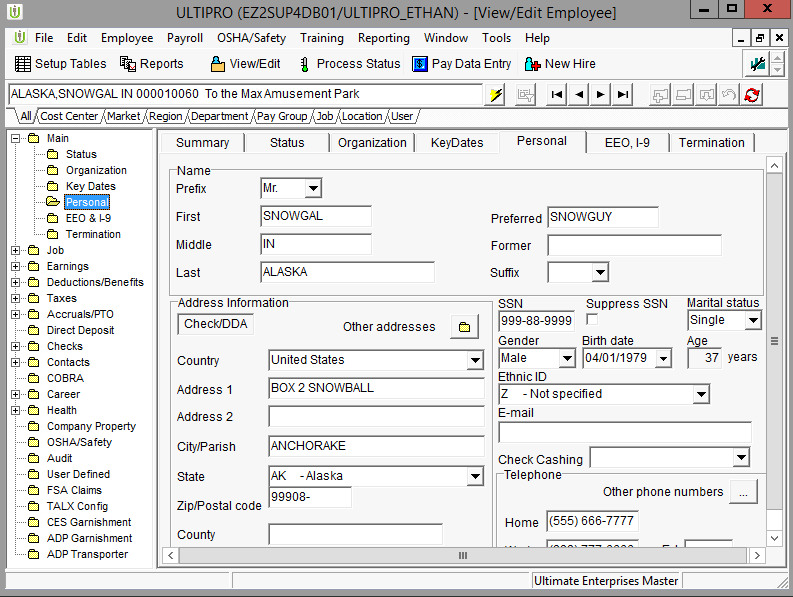


* EmpPers
  + Contains Employee personal information
  + One record per employee
  + This is now a view
  + Link
    - EepEEID – Unique Employee ID assigned by the system
    - EepHomeCOID – Don’t use unless you have to

SELECT \*

FROM dbo.EmpPers WITH (NOLOCK);

*EmpPers table accessed by application*

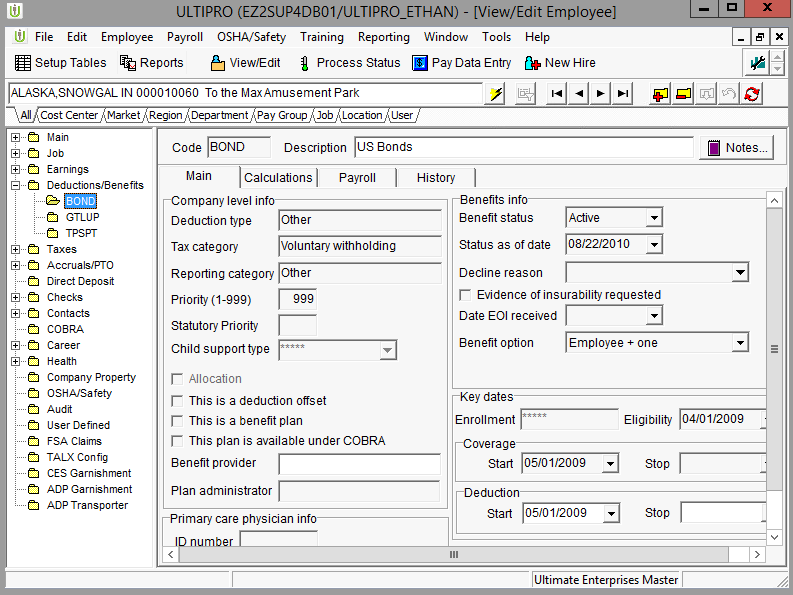


* EmpDed
  + Contains Employee deduction information
  + This is a view
  + Multiple records can exist (multiple deduction codes)
  + Link
    - EedEEID – Unique Employee ID assigned by the system
    - EedCOID – Unique Company ID
    - EedDedCode – Deduction Code

SELECT \*

FROM dbo.EmpDed WITH (NOLOCK);

*EmpDed table accessed by application.*

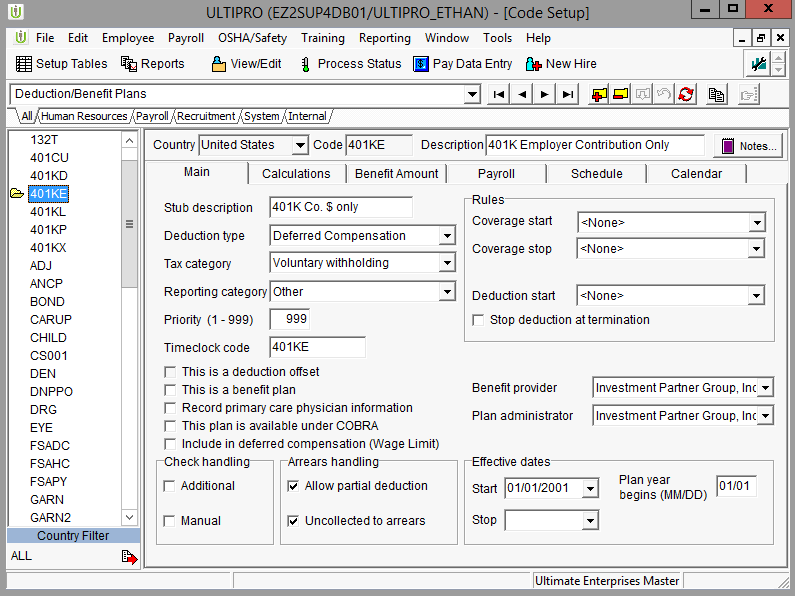


* DedCode
  + Contains Deduction setup information, Company level
  + Link
    - DedDedCode – Deduction Code

SELECT \*

FROM dbo.DedCode WITH (NOLOCK);

*DedCode table accessed by application.*

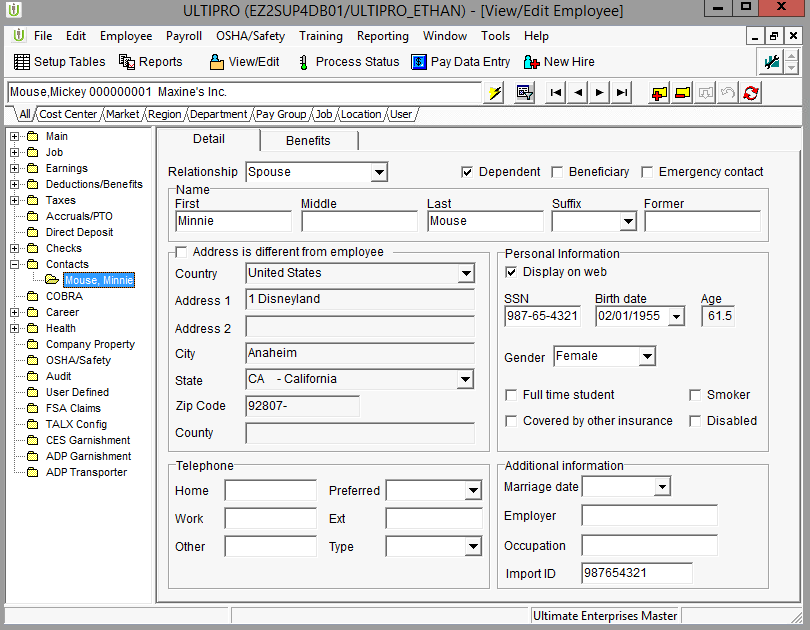


* Contacts
  + Contains dependent personal information
    - Wife, Child, etc…
  + One record per dependent, emergency contact etc…
  + It is a view
  + Link
    - ConEEID – Unique Employee ID assigned by the system
    - ConSystemID = Unique Dependent ID assigned by the system

SELECT \*

FROM dbo.Contacts WITH (NOLOCK);

*Contacts table accessed by application.*

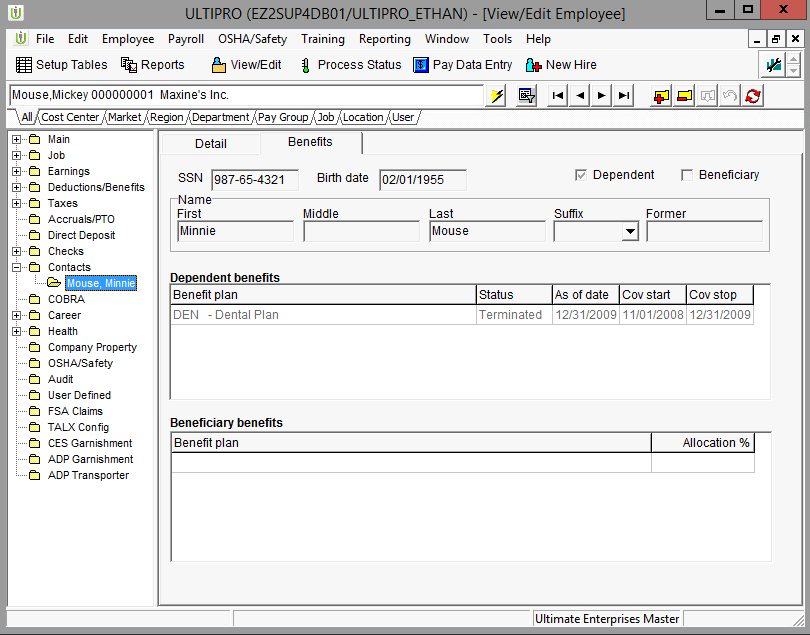


* DepBPlan
  + Contains dependent benefit information
  + Multiple records can exist (multiple deduction codes)
  + Note this doesn’t have a coid (company). Need to be mindful when building an interface. Sometimes duplicates can arise if an employee has two sets of deductions in different coids.
  + Link
    - DbnEEID – Unique Employee ID assigned by the system
    - DbnDepRecID - Unique Dependent ID assigned by the system
      * Links to ConSystemID

SELECT \*

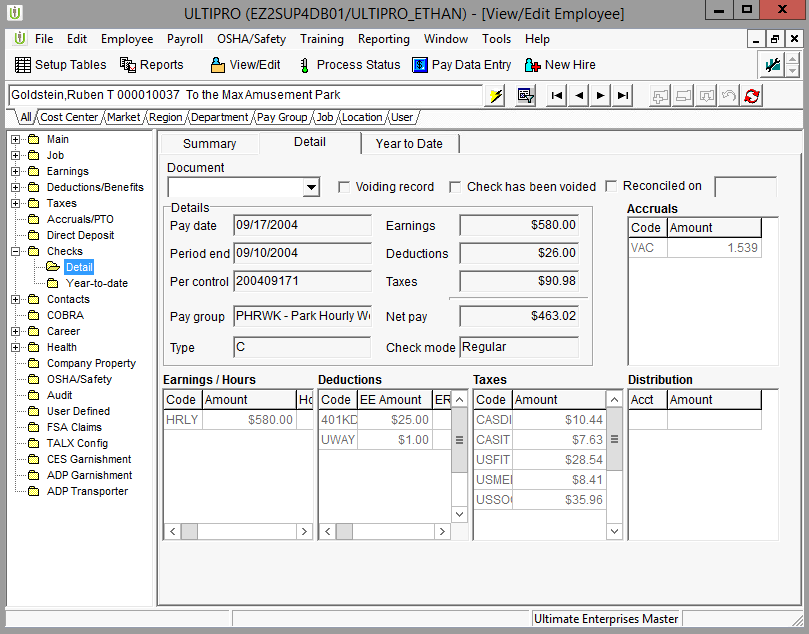
FROM dbo.DepBPlan WITH (NOLOCK);

*DepBPlan table accessed by application.*



* iPayRegKeys and PayReg (view)
  + Contains Employee Check History
  + Multiple records can exist (multiple checks)
  + Link
    - PrgEEID – Unique Employee ID assigned by the system
    - PrgCOID - Unique Company ID assigned by the system
    - PrgGenNumber – Unique ID assigned to an individual check
      * Check history tables all use GenNumber for linking between tables
* iPearHist and PEarHist (view)
  + Contains Employee Earnings Check History
  + Multiple records can exist (multiple checks and earnings codes)
  + Link
    - PehGenNumber – Unique ID assigned to an individual check
      * Check history tables all use GenNumber for linking between tables
* iPDedHist and PDedHist (view)
  + Contains Employee Deduction Check History
  + Multiple records can exist (multiple checks and deduction codes)
  + Link
    - PdhGenNumber – Unique ID assigned to an individual check
      * Check history tables all use GenNumber for linking between tables
* iPTaxHist and PTaxHist (view)
  + Contains Employee Tax Check History
  + Multiple records can exist (multiple checks and tax codes)
  + Link
    - PthGenNumber – Unique ID assigned to an individual check
      * Check history tables all use GenNumber for linking between tables

*Check History tables accessed by application.*



* **Example Queries**

What do these queries do?

SELECT EepEEID, EepSSN, EecDateOfOriginalHire, EepNameLast, EepNameFirst

FROM dbo.EmpPers WITH (NOLOCK)

JOIN dbo.EmpComp WITH (NOLOCK) ON EepEEID = EecEEID

WHERE EecEmplStatus = 'A';

SELECT EepEEID, EepSSN, EedDedCode, EepNameLast, EepNameFirst

FROM dbo.EmpPers WITH (NOLOCK)

JOIN dbo.EmpComp WITH (NOLOCK) ON EepEEID = EecEEID

JOIN dbo.EmpDed WITH (NOLOCK) ON EecEEID = EedEEID AND EecCOID = EedCOID

JOIN dbo.DedCode WITH (NOLOCK) ON DedDedCode = EedDedCode AND DedDedType = 'MED'

WHERE EecEmplStatus = 'A';

SELECT EepEEID, CmpCompanyName, EepNameLast, EepNameFirst, EecEmpNo

FROM dbo.EmpPers WITH (NOLOCK)

JOIN dbo.EmpComp WITH (NOLOCK) ON EepEEID = EecEEID

JOIN dbo.Company WITH (NOLOCK) ON CmpCOID = EecCOID

WHERE EecEmplStatus = 'A';

SELECT PrgEEID, PrgCOID, PdhDedCode, PdhEECurAmt, PrgPeriodStartDate, PrgPeriodEndDate

FROM dbo.PayReg WITH (NOLOCK)

JOIN dbo.PDedHist WITH (NOLOCK) ON PrgGenNumber = PdhGenNumber

WHERE PrgPerControl = '201608151';

SELECT EepNameFirst, EepNameLast, ConNameFirst, ConNameLast, ConRelationship

FROM dbo.EmpPers WITH (NOLOCK)

LEFT JOIN dbo.Contacts WITH (NOLOCK)

ON ConEEID = EepEEID AND ConIsEmergencyContact = 'Y'

ORDER BY EepnameLast,EepNameFirst;

# Chapter 5: Interface Developer Steps (New Projects)

1. Find the SOW.
2. Read and analyze SOW.
3. If there are any questions loop back with the IA.
4. Build Interface Loader Spreadsheet.
5. Copy Spreadsheet to Sandbox to import using Interface Loader
6. Use dsi\_sp\_InterfaceLoader\_v3 to load Spreadsheet (dsi\_sp\_InterfaceLoader\_v3 'FormatCode', 'FolderName').
7. Run dbo.dsi\_sp\_TestSwitchbox\_V2 'Formatcode', 'Exportcode' to start testing.
8. TEST the exported file in the development sandbox.
9. Run RipOut Script (\_dsi\_usp\_exportripout 'FormatCode').
10. Deploy on customer system.
11. Test with client data.
12. Prepare documentation and email notify the client or the IA the interface is ready for testing.
13. Run RipOut and archive ripout script.

**Exercise 1: The Vine Export**

** **

**Exercise 2: The Vine Export (v2.04)**



**Exercise 3: Benefit Strategies FSA Export (Part 1)**



# Chapter 6: Key Tables/Views Used in Exports

|  |  |  |  |
| --- | --- | --- | --- |
| **Object Name** | **Description** | **Columns Often Used in Joins** | **Columns Often Used** |
| CompMast | Master Company Information | CmmCOID |  |
| Company | Component Company Information | CmpCOID | CmpCompanyCode,CmpCompanyName,CmpFedTaxID |
| Codes | Miscellaneous Translations | CodCode, CodTable | CodDesc |
| TrmReasn | Termination Reasons | TchCode | TchDesc, TchType |
| JobChRsn | Job Change Reasons | JchCode | JchDesc |
| BenChRsn | Benefit Change Reasons | BchCode | BchDesc, BchIsCOBRAQualifiedEvent |
| Location(vgbl) | Location Information | LocCode | LocDesc, LocAddress info |
| OrgLevel | Organization Information | OrgCode, OrgLvl | OrgDesc |
| JobCode | Job Code Information | JbcJobCode | JbcDesc, JbcFLSAType, JbcLongDesc |
| EarnCode | Earning Code Setup Information | ErnRarnCode | ErnLongDesc, ErnInclInDefComp |
| DedCode(veff) | Deduction Code Setup Information | DedDedCode | DedDedType, DedLongDesc, DedIsBenefit |
| TaxCode(v) | Tax Code Setup Information | CtcTaxCode | CtcTypeOfTax, CtcTaxCodeDesc |
| PayGroup | Paygroup Information | PgrPayGroup | PgrPeriodControl, PgrPeriodStartDate, PgrPeriodEndDate |
| PgPayPer | Pay Period Information | PgpPayGroup, PgpPeriodControl | PgpPeriodEndDate, PgpPeriodStartDate |
| EmpPers(vgbl) | Employee Personal Information | EepEEID, (EepHomeCOID) | EepSSN, EepNameLast, EepNameFirst |
| EmpComp | Employee Job Information | EecEEID, EecCOID | EecAnnSalary, EecJobCode, EecHourlyPayRate, EecOrgLvl1 |
| EmpEarn | Employee Earnings Information | EeeEEID, EeeCOID,EeeEarnCode | EeeAmt |
| EmpDed(veff) | Employee Deduction Information | EedEEID, EedCOID, EedDedCode | EedEEAmt, EedERAmt, EedBenStartDate, EedBenStopDate,EedBenStatus,EedBenOption |
| EmpTax | Employee Tax Information | EetEEID, EetCOID, EetTaxCode | EetFilingStatus, EetExemptFromTax |
| EmpAccr | Employee Accrual Information | EacEEID, EacCOID,EacAccrCode | EacAccrAllowedCurBal, EacAccrTakenCurBal |
| EmpDirDp | Employee Direct Deposit Information | EddEEID, EddCOID | EddAcct, EddAmtOrPct |
| EmpHJob | Employee Job History | EjhEEID, EjhCOID, EjhJobEffDate, EjhDateTimeCreated | EjhAnnSalary, EjhEEType, EjhJobCode,EjhIsRateChange |
| EmpHDed | Employee Deduction History | EdhEEID, EdhCOID, EdhDedCode, EdhEffDate | EdhChangeReason |
| EmpIntl | Int’l employee info | EinEEID | EinNationalID |
| Contacts(vgbl) | Contact Information | ConEEID, ConSystemID | ConIsDependent, ConSSN, ConDateOfBirth |
| DepBPlan(veff) | Dependent Deduction Information | DbnEEID, DbnDepRecID, DbnDedCode (note don’t use DbnCOID) | DbnBenStopDate, DbnBenStatus, DbnBenStartDate |
| BnfBPlan(veff) | Beneficiary Information | BfpEEID, BfpConRecID,BfpDedCode | BfpPctToAlloc |
| EmpMAddr(vgbl) | Employee Multiple Addresses | EadEEID, EadAddressTypeCode | EadAddrSystemID |
| Address(vgbl) | Employee Multiple Address Detail | Addeeid, Addaddrsystemid | AddAddressLine1, AddAddressLine2, AddAddressCity, AddAddressState |
| EmpMPhon(vgbl) | Employee Multiple Phone Numbers | EfoEEID, EfoPhoneType | EfoPhoneNumber |
| PayReg(v) | Payroll Register | PrgEEID, PrgCOID,PrgPerControl, PrgGenNumber | PrgGenNumber, PrgNameLast,PrgNameFirst,  PrgPeriodStartDate,PrgPeriodEndDate,PrgPayDate |
| PEarHist(v) | Payroll Earnings History | PehEEID, PehCOID, PehPerControl, PehGenNumber | PehEarnCode, PehCurAmt |
| PDedHist(v) | Payroll Deduction History | PdhEEID, PdhCOID, PdhPerControl, PdhGenNumber | PdhDedCode, PdhEECurAmt, PdhERCurAmt |
| PTaxHist(v) | Payroll Tax History | PthEEID, PthCOID PthPerControl, PthGenNumber | PthTaxCode, PthCurTaxAmt |
| M\_Batch | Master Batch Table (current payroll) | MbtEEID, MbtCOID, MbtGenNumber | MbtPeriodEndDate, MbtPeriodStartDate,MbtTotEarn |
| E\_Batch | Earnings Batch Table (current Payroll) | EbtEEID, EbtCOID, EbtGenNumber | EbtEarnCode |
| D\_Batch | Deduction Batch Table (current payroll) | DbtEEID, DbtCOID, DbtGenNumber | DbtDedCode |
| T\_Batch | Tax Batch Table (current Payroll) | TbtEEID, TbtCOID, TbtGenNumber | TbtTaxCode |
| A\_Batch | Accrual Batch Table (current Payroll) | AbtEEID, AbtCOID, AbtGenNumber | AbtaAcrAllowedCurBal |
| B\_Batch | Bank Batch Table(current Payroll) | BbtEEID, BbtCOID, BbtGeNnumber |  |

For more info about each of these tables and views, including common SQL queries, please refer to the Data Dictionary.

**Key**

V – View

Gbl – Global

Eff – Effective dated

Globalization Information

|  |  |  |
| --- | --- | --- |
| Table | Description | Views Using Table |
|  |  |  |
| Iemppers | Employee Personal Information | Emppers |
| Icontacts | Contact Information | Contacts |
| Ilocation | Location Information | Locations |
| Iprovider | Provider Information | Providers |
| Person | Person Information | Emppers, Contacts |
| PersonNames | Person Name Information | Emppers, Contacts |
| PhoneNumbers | Phone Number Data | Emppers, Contacts, Location, Provider,Empmphon |
| Addresses | Address Data | Emppers, Contacts, Location, Provider,Empmaddr,Address |
| Empaddr | Employee Multiple Addresses | Empmaddr,Address |
| Iempmphon | Employee Multiple Phone Numbers | Empmphon |

Rules for Tables Included in Globalization

* You should use the views and not the tables
* The Inserts/Updates/Deletes to the Global tables are audited
* The audit records generated for the global records are cloned to create audit records for the views

Effective Dating Information

|  |  |  |
| --- | --- | --- |
| Table | Description | Views Using Table |
|  |  |  |
| Dedtypeti | Deduction Type Invariant Information | Dedtype, Dedtypefull |
| Dedtypetv | Deduction Type Variant Information | Dedtype, Dedtypefull |
| Dedcodeti | Deduction Code Invariant Information | Dedcode, Dedcodefull |
| Dedcodetv | Deduction Code Variant Information | Dedcode, Dedcodefull |
| Benprogti | Benefit Program Invariant Information | Benprog, Benprogfull |
| Benprogtv | Benefit Program Variant Information | Benprog, Benprogfull |
| Empdedti | Employee Deduction Variant Information | Empded, Empdedfull |
| Empdedtv | Employee Deduction Invariant Information | Empded, Empdedfull |
| Depbplanti | Dependent Deduction Invariant Information | Depbplan, Depbplanfull |
| Depbplantv | Dependent Deduction Variant Information | Depbplan, Depbplanfull |
| Bnfbplanti | Beneficiary Invariant Information | Bnfbplan, Bnfbplanfull |
| Bnfbplantv | Beneficiary Variant Information | Bnfbplan, Bnfbplanfull |

Rules for Effective Dated Tables

* DO NOT USE EFFECTIVE DATED TABLES IN QUERIES – ONLY ACCESS VIA THE VIEWS
  + Effective dated tables are still changing
  + Updates/Deletes/Inserts to effective dated tables are not audited. Auditing is done via the updates/deletes/Inserts to the views.
* The view with 'FULL' at the end will return all rows in the tables
* The view without 'FULL' at the end will return the row of data in effect at the time of the connection date
* The connection date defaults to the current date
* If you need to set the connection date to a future or past date you can use the function

DECLARE @conndate datetime;

SET @conndate = '09/01/2016';

EXEC MASTER.dbo.CXN\_SET\_SESSION\_DATE @conndate;

**Audit Information**

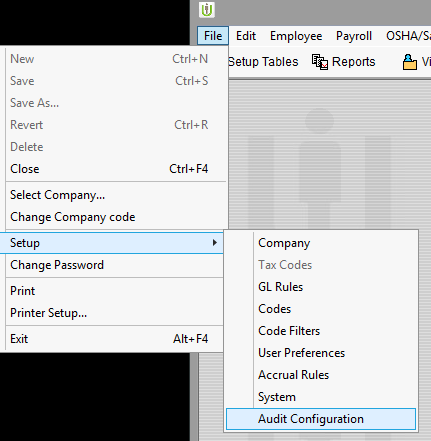
* We need to look at the audit view when we need to report changes only on the export. This includes a purely changes-only file, or it can be a full file for all non-terminated employees, but the vendor only wants terminated employees to be reported once.
* Use the view **vw\_AuditData** when accessing audit data
* Auddatetime is the date/time the transaction was entered
* Audtablename is the table the audit record is for
* Audfieldname is the field the audit record is for
* Join to the base table based on the Auditkeys (audkey1value,audkey2value,audkey3value)
* In addition to looking up this information in the Back Office, we can also query **dbo.vw\_AuditConfiguration** to see which fields are being tracked for audit

Examples of Some Key Audit Table Join Information

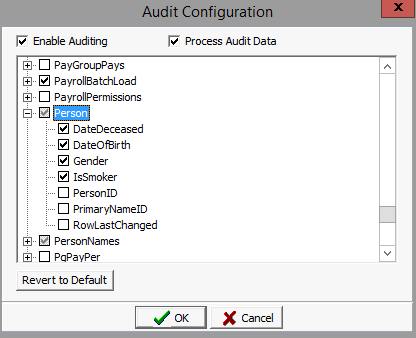
|  |  |  |  |
| --- | --- | --- | --- |
| Audit Tablename | Audkey1value | Audkey2Value | Audkey3value |
|  |  |  |  |
| Emppers | Eepeeid |  |  |
| Empcomp | Eeceeid | Eeccoid |  |
| Empearn | Eeeeeid | Eeecoid | Eeeearncode |
| Empded | Eedeeid | Eedcoid | Eeddedcode |
| Emptax | Eeteeid | Eetcoid | Eettaxcode |
| Contacts | Coneeid |  | Consystemid |
| Depbplan | Dbneeid | Dbndedcode | Dbnsystemid |

* To find audit keys you can look at the tables, audit triggers or you can look in the vw\_auditdata to see what fields are referred to in audkey1label, audkey2label and audkey3label
* Audit also needs to be turned on for the fields you wish to audit. This can be done in the Back Office:

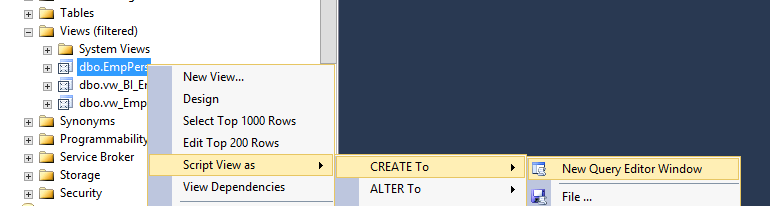
File -> Setup -> Audit Configuration



On the next screen, make sure the fields you want to audit are checked.



Note that table names are listed, not views. You will not see views that you frequently use. For example, EmpPers will not appear on this list. If you’re not sure which tables and fields are being used in a particular view, you can review the view definition in SQL Server Management Studio.



# Chapter 7: Framework Tips and Tricks

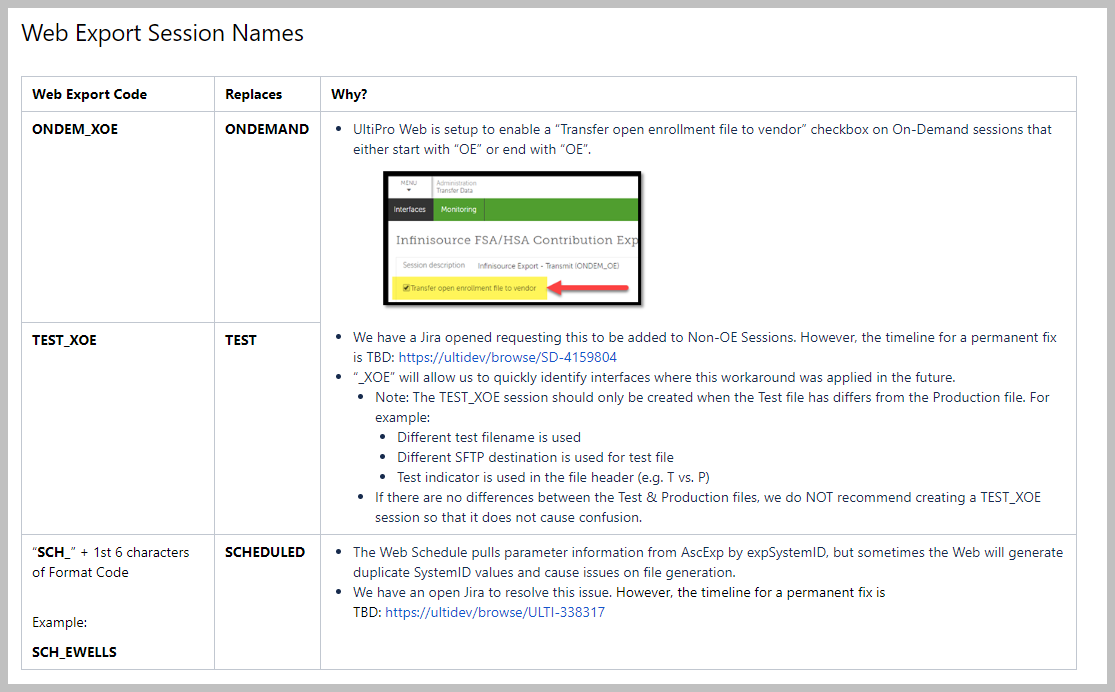
1. Create the Employee History Setup Template Layout using the Excel Layout Spreadsheet. The layout for the export will be contained in the SOW, created by the Integration Analyst.

**TIP:** When copying and pasting from the SOW into the Expression column, use Paste Special and select Text. This keeps the SOW formatting separate from the spreadsheet formatting.

**TIP:** Make the SetNo column sequential, H01, D10, D11, T90. This allows the program to sort the header, detail, and trailer records in the proper order if required. Header Records should be 01-09, Detail Records 10-89, and Trailer Records 90-99.

**TIP:** Remove any single quotes around literals. They are not required by the Framework. Remember single quotes in Word are different from ticks in SQL.

1. If they’re not created after loading the spreadsheet, additional export sessions can be created by going to Tools > Exports > Employee History Export.



1. Add or update the required values in the U\_Dsi\_Configuration table. Do not hardcoded file paths (database name and server name included) in the stored procedure. Clients cannot save to their C drive.

**TIP:** The file path must be in the UNC Format (\\us.saas\e1\...)

1. Populate the U\_Dsi\_SQLClauses table.

**TIP:** An entry is required for each for each record set (H01, D10, T90, …) retrieving data from the database. Check ascdeff to see how many combinations of record types and set nos. That is how many records are to be in SQLClauses.

**TIP:** Do NOT put FROM in the FromClause field. Just the table name.

**TIP:** Do NOT put WHERE in the WhereClause field. Just the criteria of the WHERE clause. Rarely used anyway.

**TIP:** If the recordset does not need data from SQL. Then place NONE in the FROM clause.

**TIP:** If a record set in the Export’s setup template is not required to be placed on the export file, place the value IGNORE in the WhereClause field within U\_Dsi\_SqlClauses table. This will exclude the record set from the file.

1. Create or Alter the view dsi\_vw<FORMATCODE>\_Export if necessary

**TIP:** The standard code is:

SELECT TOP 2000000

Data FROM dbo.U\_<FORMATCODE>\_File WITH (NOLOCK)

ORDER BY SUBSTRING(RecordSet,2,2), InitialSort, Subsort;

**TIP:** Example, if the file requires all upper case alpha characters use dsi\_Configuration and set UPPER to Y

**TIP:** Sorting a View with the key words TOP <Number> PERCENT in SQL 2005 no longer sorts the data correctly. TOP PERCENT no longer is supported by Microsoft within Views. Use TOP 2000000 instead.

1. You will most likely need to clean the EEList of any previous company transfers. Below is the best way to do it.

----Clean EE List

DELETE FROM dbo.U\_EFORMAT\_EEList

WHERE xCoID <> dbo.dsi\_BDM\_fn\_GetCurrentCoID(xEEID)

AND xEEID IN (SELECT xEEID FROM dbo.U\_EFORMAT\_EEList GROUP BY xEEID HAVING COUNT(1) > 1);

Also, be sure to utilize the function when you attempt to report employee level information in subqueries or join. For example, here’s a subquery that reports the employee’s supervisor’s Employee Number.

SELECT

drvEmpNo = EMP.EecEmpNo

,drvSupervisorEmpNo = (SELECT SUPER.EecEmpNo FROM dbo.EmpComp AS SUPER WITH (NOLOCK) WHERE SUPER.EecEEID = EMP.EecSupervisorID

AND SUPER.EecCOID = dbo.dsi\_BDM\_fn\_GetCurrentCOID(SUPER.EecEEID))

FROM dbo.EmpComp AS EMP WITH (NOLOCK)

1. If you are getting a -1 by the switchbox double check that SQL has access to save the data to the folder.

exec master..xp\_cmdShell 'dir \\<ExportPath>\' ;

exec master..xp\_cmdShell 'echo > \\<ExportPath>\Test.txt' ;

exec master..xp\_cmdShell 'dir \\<ExportPath>\Test.txt' ;

exec master..xp\_cmdShell 'del \\<ExportPath>\Test.txt' ;

exec master..xp\_cmdShell 'bcp "select top 5 \* from ultipro\_<server>.dbo.empPers with (nolock)" queryout \\<ExportPath>\EmpPers.bcp -n -T'

1. If you need to keep track of how many times the export has run for a file name.

IF @Exportcode = 'SCHEDULED'

BEGIN

SELECT @LastRun = CAST((CASE WHEN ISDATE(CfgValue) = 1 THEN CfgValue ELSE GETDATE() END) AS DATETIME);

FROM dbo.U\_dsi\_Configuration WITH (NOLOCK) WHERE FormatCode = 'EAMRAPPAY' AND CfgName = 'LastRun';

UPDATE dbo.U\_dsi\_Configuration  
SET CfgValue = CONVERT(VARCHAR,@EndDate,121)  
WHERE FormatCode = 'EAMRAPPAY' AND CfgName = 'LastRun';

SET @StartDate = @LastRun;

END;

cid:image001.png@01D20383.35C7C790

1. If you need to find the table name for a file

--Find Table

SELECT DISTINCT a.name,b.name

FROM dbo.syscolumns b

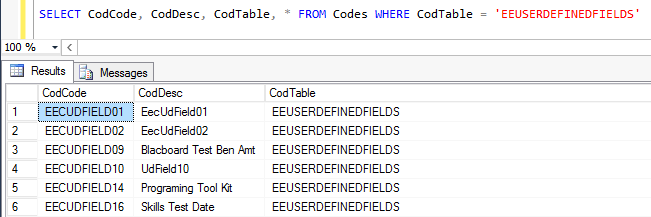
JOIN sysobjects a ON a.id = b.id

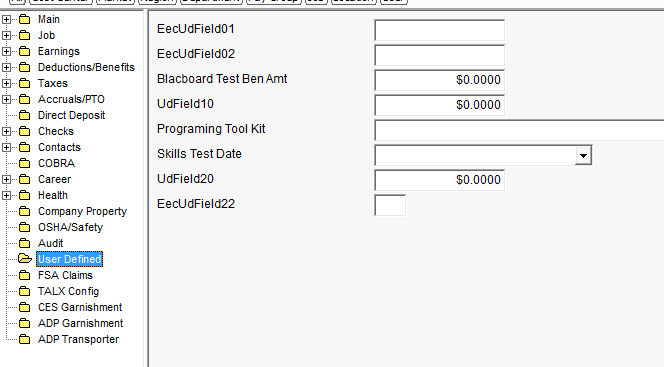
WHERE b.name LIKE '%flsa%'

AND a.xtype = 'U';

1. User-defined fields in EmpComp can have custom labels. The custom labels are stored in the Codes table.

SELECT CodCode, CodDesc, CodTable, \* FROM Codes WHERE CodTable = 'EEUSERDEFINEDFIELDS'





1. Switchbox Error Codes:

|  |  |  |
| --- | --- | --- |
| Output Record Value | Root Cause | Resolution |
| -1 | File cannot be generated | 1. File Path is incorrect in U\_dsi\_Configuration table. 2. File Path Permissions (Access is Denied). |
| -2 | Summary File cannot be generated | 1. Summary File Path is incorrect in U\_dsi\_Configuration table. 2. Summary File Path Permissions (Access is Denied). |
| -3 | Archive File cannot be generated | 1. Archive File Path is incorrect in U\_dsi\_Configuration table. 2. Archive File Path Permissions (Access is Denied).   This only applied to Switchbox V2. |
| -33 | Initial Setup in U\_dsi\_Configuration table does not exist | Switchbox will create 'default' values in U\_dsi\_Configuration table for the following fields: EEList, ExportPath, OEPath, InitialSort, SubSort, Testing, UseFileName  Re-run the file will resolve the issue, but it's recommended to update the values before re-running the file. |
| -44 | The 'SelectByField' does not match. | Correct the 'SelectByField' in AscExp. |
| *-66* | *The Export TEMPLATE is locked when the AscExp.ExpSystemID is NULL/Blank* | *This only applied to Switchbox V1. You must correct any errors and run 'ZAP' to unlock the Session.* |
| -88 | Max Characters for Fixed Length File is over 8000 characters (AscDefH.AdhRecordSize > 8000 and AscDefH.AdhFileFormat = 'SDF') | Fixed Length/Fixed Width files over 8000 characters are not supported.  You may need to use Fixed Length/Variable Width files by setting AscDefH.AdhFileFormat = 'CDE'.  This only applied to Switchbox V2. |
| *-99* | *Unlock a Locked Session* | *This only applied to Switchbox V1. This will unlock a locked session.* |

1. To convert a file to UTF-16 format (only available in Switchbox V2):

DECLARE @FormatCode VARCHAR(10), @sCmd VARCHAR(200), @IsUTF VARCHAR(100)

SET @FormatCode = 'EFORMATCODE'

SET @IsUTF = 'Y'

SET @sCmd = '

INSERT INTO dbo.U\_Dsi\_Configuration

SELECT ''' + @FormatCode + ''', ''IsUTF'', ''V'', ''' + @IsUTF + '''

'

EXEC (@sCmd)

1. To convert a file to UTF-8 format (only available in Switchbox V2):

DECLARE @FormatCode VARCHAR(10), @sCmd VARCHAR(1000), @IsUTF8 VARCHAR(1), @UTF8Path VARCHAR(1000)

SET @FormatCode = 'EFORMATCODE'

SET @IsUTF8 = 'Y'

SET @UTF8Path = 'Enter a valid path to be used temporarily for conversion'

SET @sCmd = '

INSERT INTO dbo.U\_Dsi\_Configuration

SELECT ''' + @FormatCode + ''', ''IsUTF8'', ''V'', ''' + @IsUTF8 + '''

INSERT INTO dbo.U\_Dsi\_Configuration

SELECT ''' + @FormatCode + ''', ''UTF8Path'', ''V'', ''' + @UTF8Path + '''

'

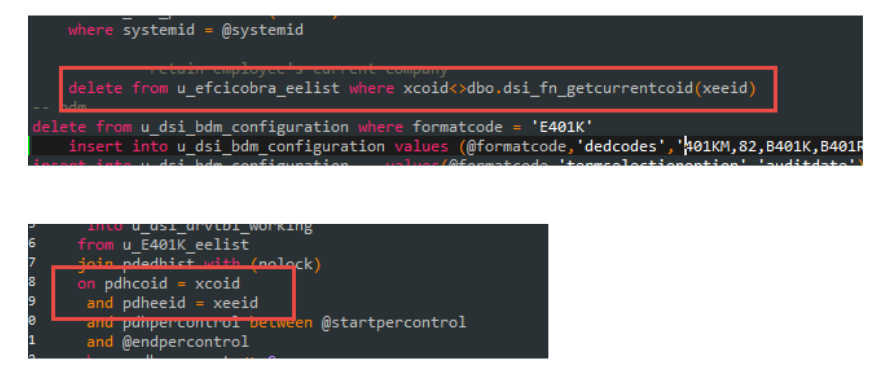
EXEC (@sCmd)

* The UTF-8 conversion process will pull all files in the UTF8Path folder and convert them, then move the converted files into the ExportPath folder. This will allow for UTF-8 conversion of multiple files generated via AfterCollect, as long as the files are generated to the UTF8Path folder.
* If you are using an after collect, include -w for widechar in your BCP statement.

SET @BCPStr = 'bcp "SELECT Data FROM ' + DB\_NAME() + '..' + @SourceTableName + '" QUERYOUT "' + @FileName + '" -S' + @@SERVERNAME + ' -T -w -t'

* You can validate the file is in UTF-8 Format with an Advanced Text Editor like Notepad++.

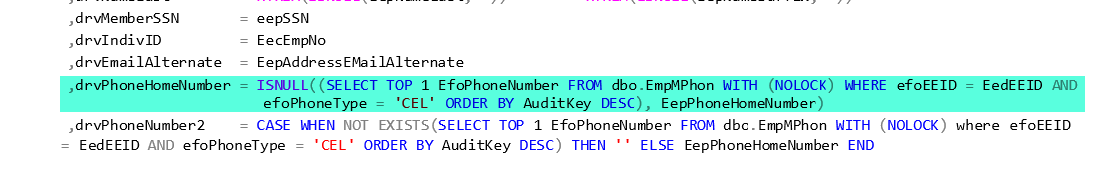
1. The export reduces EEList to current companies, but the payroll working table joins on COID. If the employee jumps companies any YTD contributions will be wrong



1. All table references must have a "dbo." qualifier; all joins should use the WITH (NOLOCK) hint

Also, watch for dbo.dbo’s... these can happen when a TC has done a search-and-replace, and will blow up when the customer tries to run it in BackOffice

1. Only call the **BDM** on exports that actually need it... if none of the fields in your Spec start with EED (or better, BDM) you’re not using benefits. If not using EED or BDM, do not have a BDM call on Payroll files. It didn’t hurt previously but it now is, as it making SP's run longer than they should(performance issue).
2. Avoid inline Select Statements when possible. Instead, make use of “Left Joins” better or available function from the function library:



SELECT EepEEID, EepPhoneHomeNumber, EepAddressEmail, dbo.dsi\_fnlib\_GetPhoneNumber(EepEEID, 'CEL') FROM dbo.EmpPers

1. Use the PaddZero function when padding amounts. If you need to turn **123.45** into **0000012345**, use the dbo.dsi\_fnPadZero function:

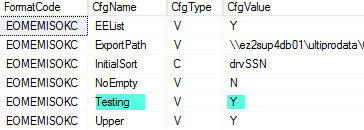
dbo.dsi\_fnPadZero(@numIn,@Length,@Decimal)

1. Use FieldNames or descriptive drv names. Ie. drvDateInService,drvAge as opposed to drvField1,drvField2.
2. drv Field Names should be descriptive of the vendor’s field name description rather than the UltiPro field name

**PREFER: drvDepartment = EecOrgLvl2**

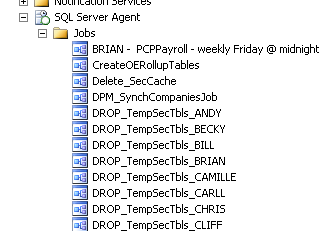
NOT: drvOrgLvl2 = EecOrgLvl2

1. References to the audit tables should not be made directly in the driver table. A working table should be created at the beginning of the stored procedure to gather all audit data needed for the file. The working audit table can then be referenced using a in left join when creating the driver table. Inline select statements should be avoided when calling audit data.
2. Terminate all SQL statements with semi-colons ;
3. Clean the EELIST before you run the BDM. Reducing the size of the EELIST table will improve performance
4. **Troubleshooting**: Set “Testing” Flag in U\_dsi\_Configuration table to **Y** in order to see the current process output to see approximate point of error. Set flag to N when completed:

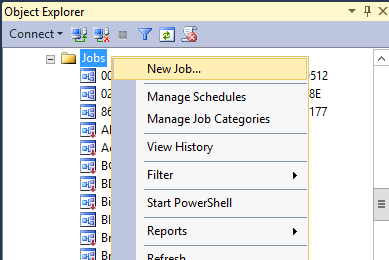


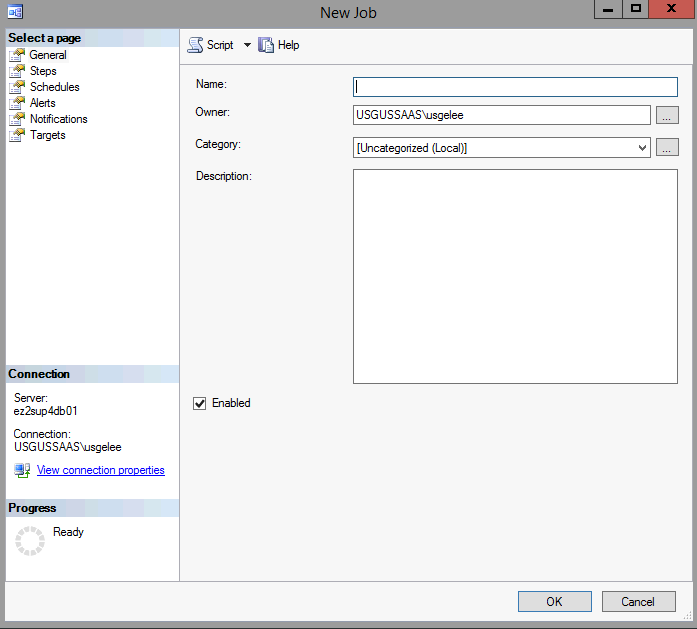
# Chapter 8: Scheduling Interfaces outside of the Web Portal

* Go to SQL Server Agent



* Right mouse click and and go to New Job





* In the name add the company - export name (export code) - frequency and schedule of the export.
* Change the Owner to **ssis\_user**. If the Owner field isn’t enabled, run the query below:

---------------------

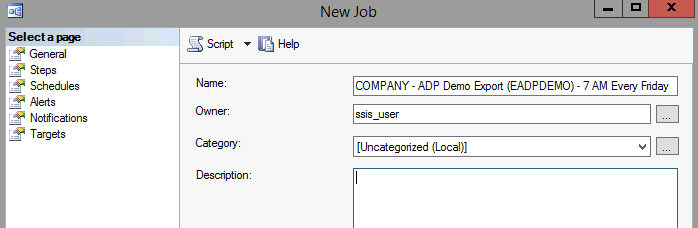
-- Set job owner to ssis\_user for Production jobs

EXEC msdb..usg\_set\_job\_owner @job\_name = '', @set\_owner\_to\_self = 0;

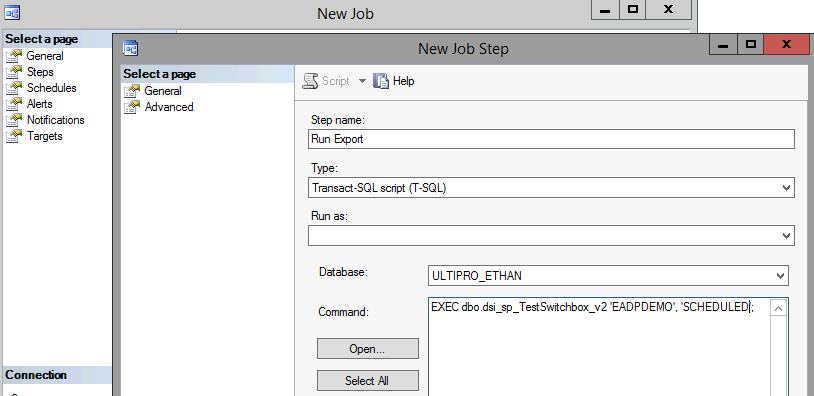
-- Set job owner to self, to make changes in SQL Job Scheduler. Remember to re-run -- the above script to change ownership to ssis\_user when complete

EXEC msdb..usg\_set\_job\_owner @job\_name = '', @set\_owner\_to\_self = 1;

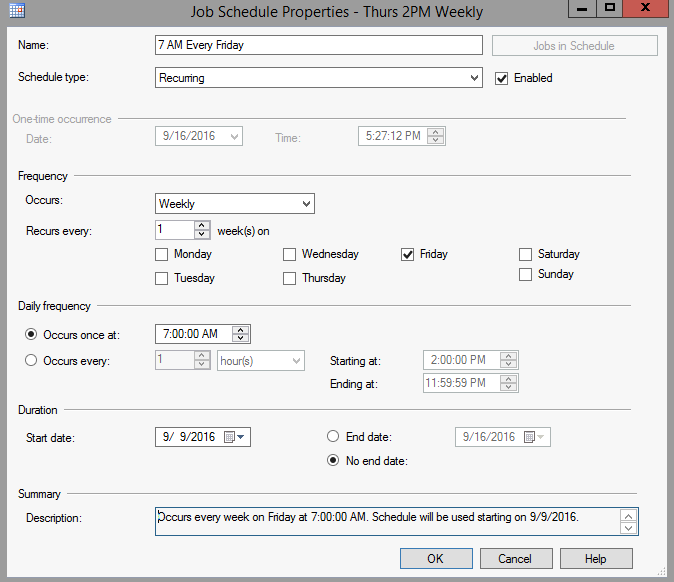
* Description is not needed unless we are calling DTS packages or the Name field doesn’t give another user the full picture.



* Next go to Steps. The step name is not needed unless there are multiple steps in the SQL job. If there are not multiple steps then just add Run. The Type should be T-SQL and Run as shouldn’t be filled in. The Database will need to be the correct client database. Next paste the test switchbox code with the formatcode and export code in the command line.



* Go to the Scheduled page and enter the frequency and configure appropriately.



Note: Every day between 3 AM and 5 AM Eastern is the maintenance window. We cannot schedule a job during this time.

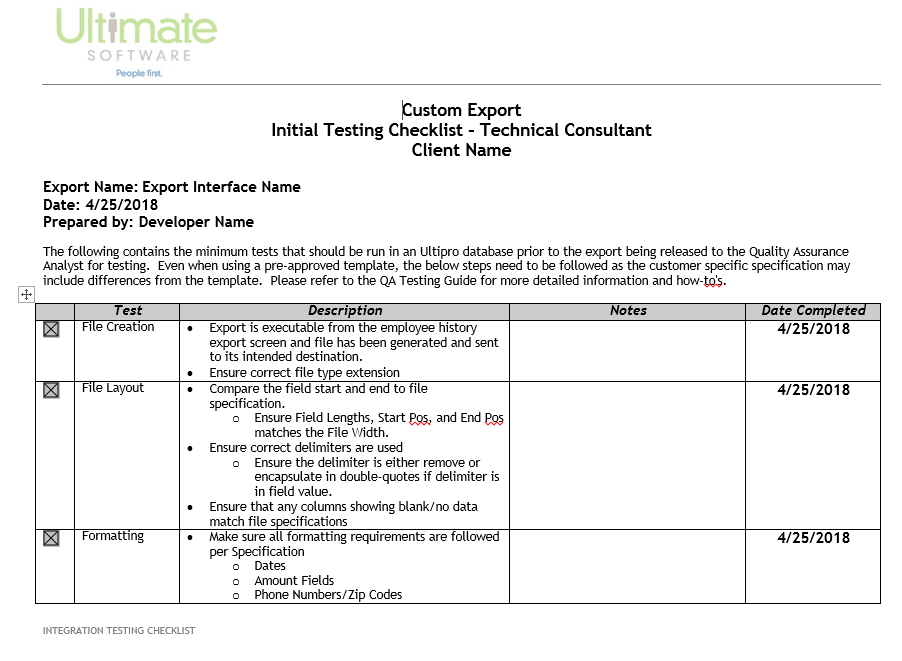
# Chapter 9: User Documentation

For brand new interfaces we develop for UDE customers, we have to create three documents before we release the interface for UAT.

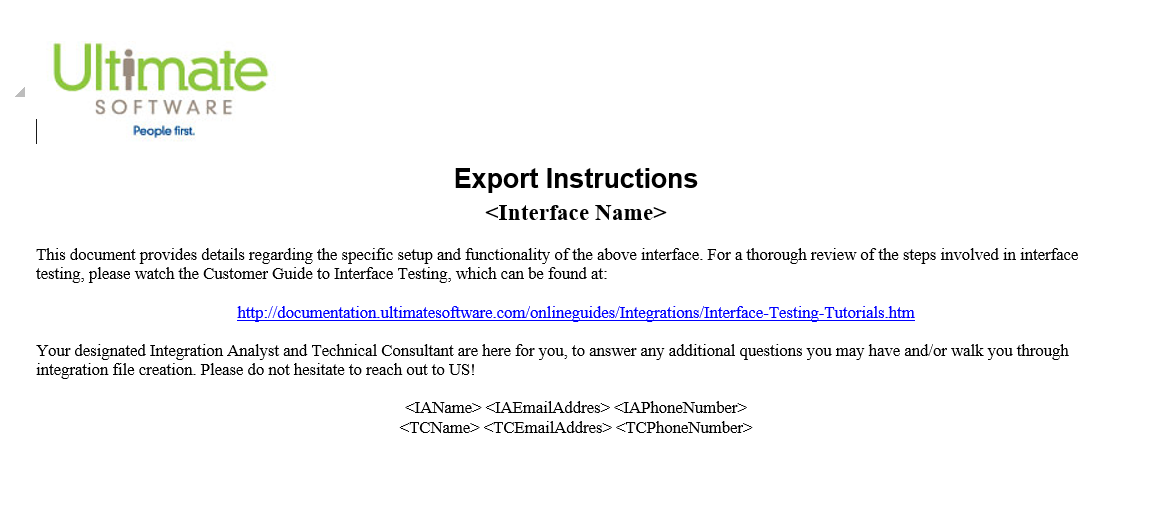
**Customer Acceptance**



**Test Plan**



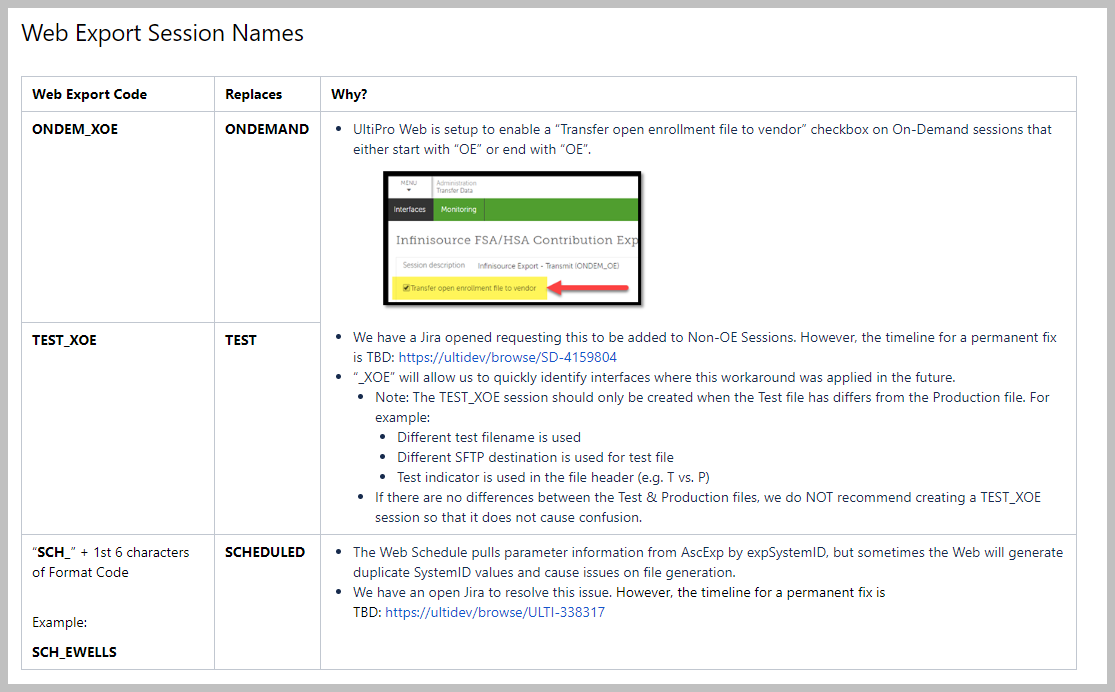
**Export Instructions**



# Chapter 10: Interface Team SQL Standards (v2)

**SQL General Development Standards**

* Formatcode should follow the below naming convention and be in ALL CAPS
  + E - Exports. Example EMED834
  + U – Updates. Example UDOFHIRE
  + I - Imports. Example IBENEFITS
* Export codes all need some standards. Use the naming convention below



* Add logic to determine whether the TEST/TEST\_XOE session is being run or the Testing parameter in the configuration table is still set to Y. Report the test/production flags or values (if required on the file layout) accordingly, instead of hardcoding those values.
* The naming convention of the SQL Script is [ObjectName].sql eg. U\_<formatcode>\_EmployeeStockOptions.sql or dsi\_sp\_BuildDriverTables\_EMED.sql for Subversion
* Use ‘dbo.’ before table names.
* Use WITH (NOLOCKS) on all select statement tables.
* Make your code easy to read. Indent your code and use camel case for object names (EmpPers, EmpComp, EecOrgLvl2, etc.). For reserved keywords, use upper case (SELECT, FROM, CASE, WHEN, IF, etc.).
* Future-proof your logic. Imagine the potential changes that the customer might ask for in the future and program your logic accordingly.
* **DO NOT** make updates to the USI, BDM, Switchbox\_v2 or Testswitchbox\_v2
* When changes are being made, be sure to include some comments. When implementing complicated or unusual logic in our stored procedure, also include some comments to explain the reasoning behind it.
* If you use a "BEGIN TRAN" statement check the transaction count and perform a “ROLLBACK TRAN” if the transaction failed. If changes look good, then be sure to “COMMIT TRAN.” Do not leave transactions uncommitted.
* For the creation of SQL Objects, Data Inserts, Linked Servers etc. always first check “IF EXISTS”, and do a drop if required. Do not DROP Tables in Table Create scripts. All scripts should be able to be run multiple times without any negative impact.

IF OBJECT\_ID('U\_ESTARBUILD\_drvTbl','U') IS NOT NULL

DROP TABLE dbo.U\_ESTARBUILD\_drvTbl;

IF OBJECT\_ID('dsi\_sp\_AfterCollect\_ESTARBUILD') IS NOT NULL

DROP PROCEDURE dbo.dsi\_sp\_AfterCollect\_ESTARBUILD;

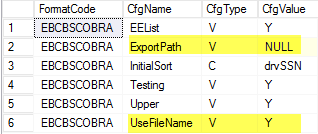
* Avoid the usage of CURSORS where possible, use WHILE loops instead
* Avoid using Temp Tables #TMP tables. This makes debugging the code much more difficult. Use working tables so they retain data being used on the latest export files, in case we need to research on issues.

# Chapter 11: Web vs. Back Office Exports – Coding Differences

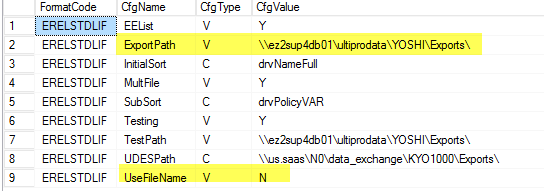
**U\_Dsi\_Configuration Table**

Two parameters highlighted below need to be set differently.

**Web**:



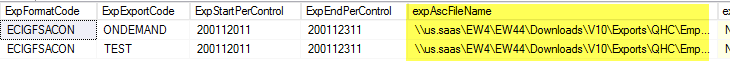
**Back Office**:

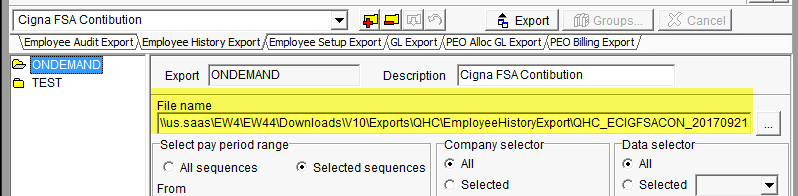


**AscExp Table**

ExpAscFileName must be set to the path and file name for the interface to be run via the Web.

**Web**:





The path is set automatically to match the client’s environment, if @IsWeb is set to Y in the rip-out script:

EXEC dbo.\_dsi\_usp\_ExportRipOut @FormatCode = 'ECIGFSACON', @AllObjects = 'Y', @IsWeb = 'Y'

**Back Office**:

We control the file name in the BuildDriver stored procedure and the file path in the U\_Dsi\_Configuration table, so we just set ExpAscFileName to “File Name is Auto Generated” to remind the client.

**CustomTemplates Table**

**Web**: We also need to run this additional statement to make the export format visible on the web:

BEGIN TRAN

DECLARE @FormatCode VARCHAR(10);

SET @FormatCode = 'EFORMATCODE';

INSERT INTO dbo.CustomTemplates(CreationDate,Engine,EngineCode,IsActive,ModifiedDate)

SELECT CreationDate = GETDATE()

,Engine = AdhEngine

,EngineCode = AdhFormatCode

,IsActive = 1

,ModifiedDate = GETDATE()

FROM dbo.AscDefH WITH (NOLOCK)

WHERE AdhFormatCode = @FormatCode AND adhFormatCode <> 'EFORMATCODE';

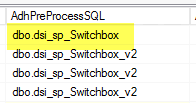
COMMIT TRAN;

Note – For an client who wants to run an export on-demand on the web, we must create an export code that has “OE” in it to trick the Web product that this is an open-enrollment session. This enables the option to whether to send the file through “File Delivery” (UDES).

# Chapter 12: Upgrading Switchbox v1 to v2

Occasionally, when you’re asked to modify an existing interface, you might find that the interface is still using switchbox v1. You should update it to use v2 so we can track the run history and also give the export more features.

Step 1: Update AscDefH.AdhPreProcessSQL for the format code to use **dbo.dsi\_sp\_Switchbox\_v2**.

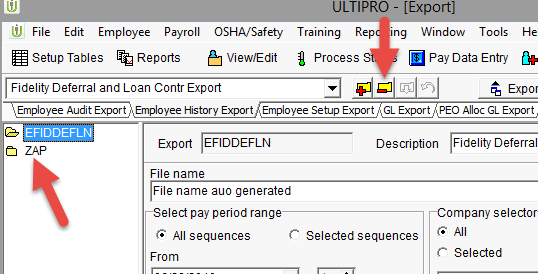


Alternatively, you can run an update script:

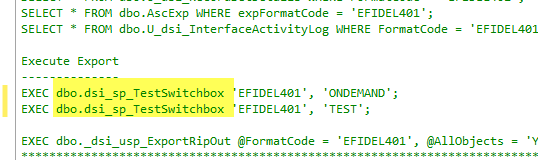
UPDATE dbo.AscDefH SET AdhPreProcessSQL = 'dbo.dsi\_sp\_switchbox\_v2'

WHERE AdhFormatCode = 'EFORMATCODE'

Step 2: Delete the ZAP export session. This export session is not needed or used in v2.



Step 3: Update all references to dsi\_sp\_TestSwitchbox in the stored procedure to use **dsi\_sp\_TestSwitchbox\_v2** for easy execution from SQL server. Remove the reference to the ZAP export session, if it is present.



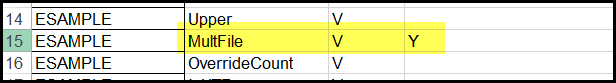
Step 4: If there’s a SQL scheduled job set up, update the Job Step Commands to use the dsi\_sp\_TestSwitchBox\_v2 function as well.

# Chapter 13: After Collect Stored Procedure

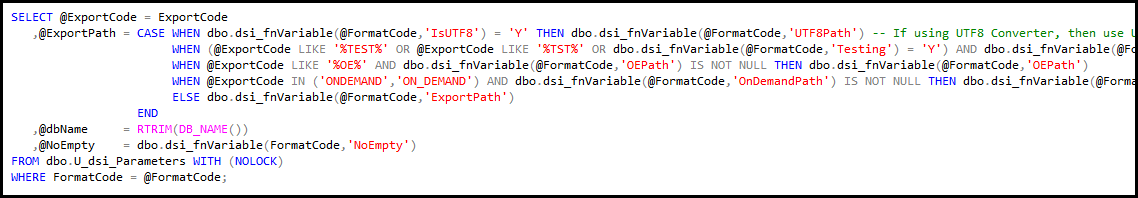
When we build an export for a UD customer that needs to produce more than one file, UD TCs create multiple exports (different format codes), one for each file. On the UDE side, we can implement a different solution called **After-Collect** to help split the data in our driver table(s) into multiple files. SwitchBox checks to see if an After-Collect stored procedure exists. If so, it will run this stored procedure after the typical BuildDriver stored procedure to further manipulate the data and allow us to split the data into different files.

Here are the steps:

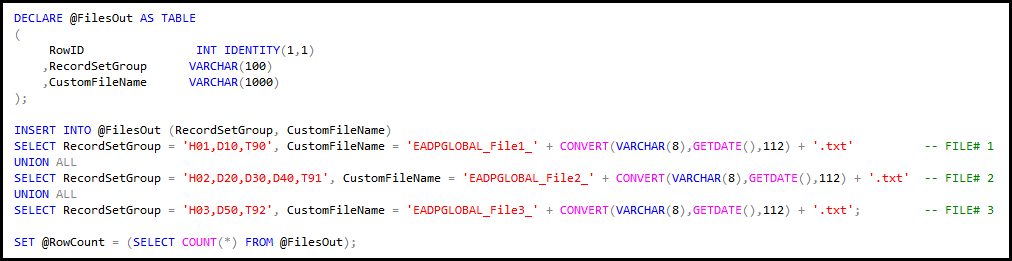
1. Set the MultiFile parameter to Y on the interface spreadsheet. This tells the Interface Loader to generate the template After-Collect stored procedure code for us.



1. After the spreadsheet is loaded, the code generated will include an additional stored procedure called **dsi\_sp\_AfterCollect\_EFORMAT**.
2. Open the After-Collect stored procedure, and you will see logic that determines the @ExportPath variable based on the Export Code. Modify this logic as needed.



1. Review the @FileOut table variable. This is where each file will be defined – specify the record set and the file name. Add additional lines below for additional files.



Note: Sometimes we need to separate records from the same record set/driver table into multiple files. In that case, list the record sets that contain the records in each file, i.e. D10 for file 1, D10 for file 2, D10 for file 3, etc.

1. Once the record sets for each file are defined, After-Collect will loop through the @FileOut table variable, select records from those record sets, and BCP out those records to the corresponding file name. When necessary, tweak the @QueryInfo variable to select only certain records from a record set, or to order the records differently. The @sCmd variable can also be tweaked if a different BCP statement is needed.



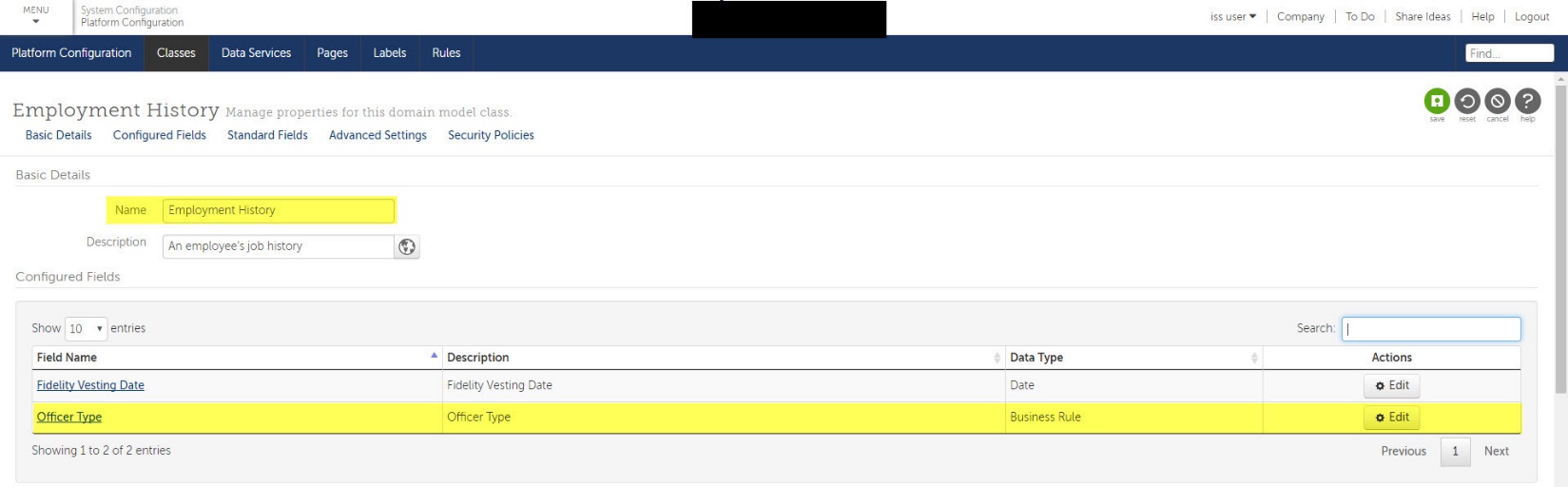
# Chapter 14: Reporting from Platform Configurable Fields

**Platform Configurable (PC) Fields**

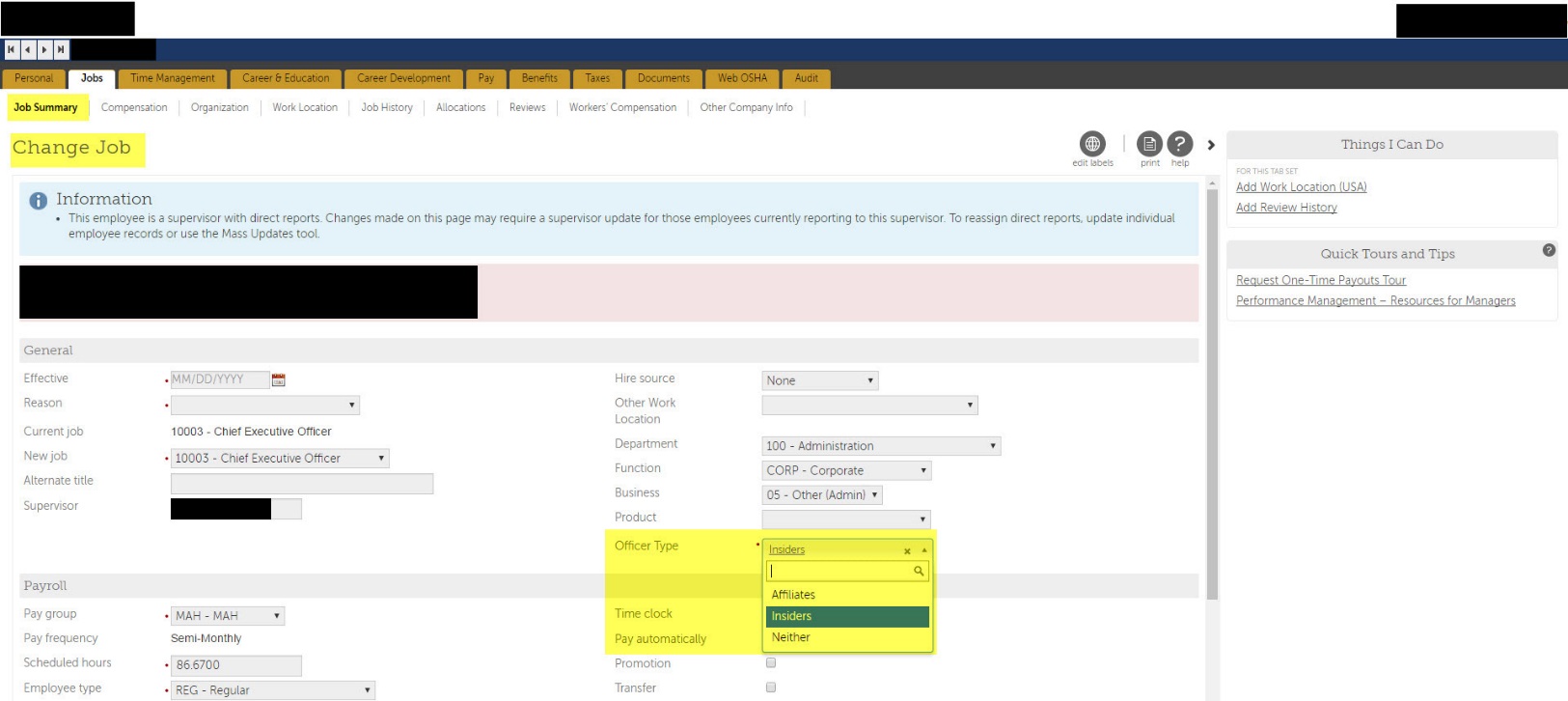
The latest feature made available in the Web Portal are platform configurable fields, where the customer has the ability to create a custom field that's associated to a "Class".  An example of a Class is the "Employment History" record.  This custom field is then added to a Web Portal screen of the customer's choice, usually an Add/Change Wizard screen.  The custom field can then be populated with information the customer needs to track in UltiPro.

You can find the Platform Configurable Field Setup in the Web Portal by going to Menu > System Configurations > Platform Configurations. Under the Class tab, then it will list all the available Classes that a PC Field can be created under.

Below is an example of the Configurable Fields setup under the Employment History Class:



Below is an example of the same Platform Configurable (PC) Field called **"Officer Type"** on the Change Job screen of an Employee's Job History record:



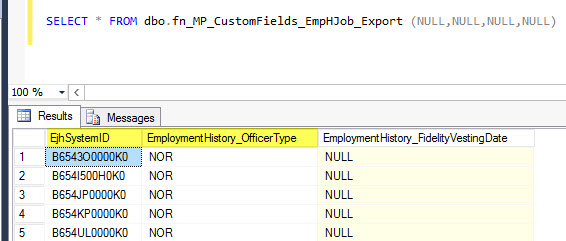
As you will see in the "Officer Type", the Platform Configurable Field was setup as a drop-down list with values that the customer defined.  Normally user defined Codes and Descriptions are stored in the dbo.Codes table in UltiPro Database where the CodTable starts with "CO\_" and ends with the name of the Platform Configurable Field setup in the database.

**Identifying the Platform Configurable Field Name in Database**

First, we will need to identify the actual Platform Configurable Field name that's setup in the database.  It's recommended to use the Table-Value functions provided by Development Team for the associated Platform Configurable Class.  In this case, it is the Employee Job History table (EmpHJob).

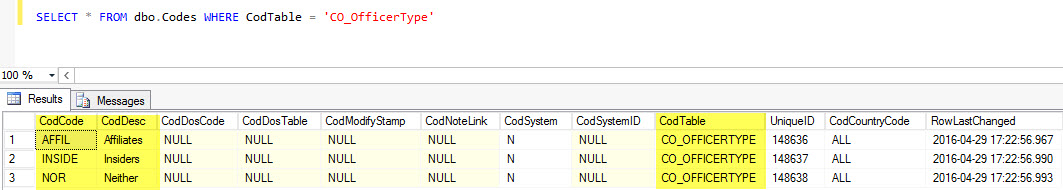
The Table-Value Functions for Platform Configurable Fields use the naming convention: dbo.fn\_MP\_CustomFields\_[TableName]\_Export (NULL,NULL,NULL,NULL).  For this example, it would be dbo.fn\_MP\_CustomFields\_EmpHJob\_Export (NULL,NULL,NULL,NULL).

Below is an example of the results produced:



As you will see the Platform Configurable Field will normally have an Identified Field (EjhSystemID) to link on, then the PC Field will be based on the Class and Field Name (EmploymentHistory\_OfficerType).  You will notice the Class is normally based on the Class Name and the Field Name are defined in the Platform Configuration Setup in the System Configurations.

Now we want to tie the PC Field back to the dbo.Codes table to get the corresponding Description, so we would use the Platform Configurable Field Name (OfficerType) as part of the CodTable to get "CO\_OfficerType".  This will produce the following results:

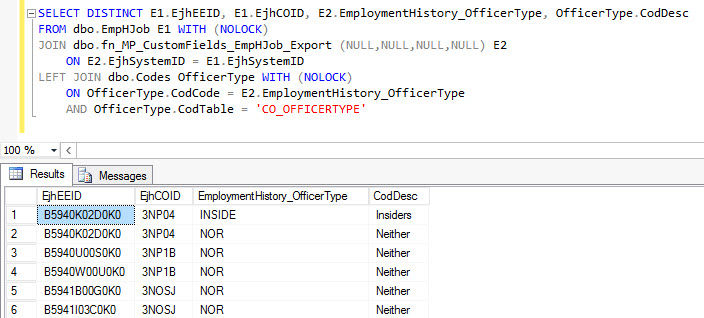


**Putting It All Together - Sample Query**

Lastly, we have identified the Platform Configurable Field Table-Value Function and Codes table queries.  We want to join them together to an Employee Record so that we can properly display the Description related to that Employee.

As you recall, the PC Field Table-Value Function usually has an Identifier (ID) Field used for joining.  In this instance, it would be the EjhSystemID which is directly related to the EmpHJob.EjhSystemID in the database.  We will need to join the PC Table-Value Function: (dbo.fn\_MP\_CustomFields\_EmpHJob\_Export (NULL,NULL,NULL,NULL) to the Employee Job History table: dbo.EmpHJob, then join to the dbo.Codes table where CodTable = 'CO\_OfficerType'.

Below is a Sample Query that produces the results:



Finally, we are able to link the Employee's record to the Officer Type and Description from the Platform Configurable Field!

To see the other available PC functions set up for this customer, you can simply navigate to the Table-valued Functions folder on the SQL server and filter by “MP\_”.

