Degree Works

Technical Guide

Release 5.0.3.1
March 2020

Notices

Ellucian
2003 Edmund Halley Drive
Reston, VA 20191
United States of America

Contents

Introduction

Document Organization

This Degree Works Technical Guide has been divided into the following sections:

Introduction. The Introduction provides you with a general impression of how the software works, how the various pieces are interconnected, and explains special topics in a focused manner.

Special Topics. The Special topics section discusses a variety of topics that are required to help you use Degree Works effectively.

Scribe. The Scribe User Guide documentation explains how to enter degree requirements using the Degree Works language and Scribe application.

Web Interface. The Degree Works on the Web documentation explains how users can process degree audits, exceptions, and notes.

Database Tables. The list and descriptions of all tables used within Degree Works.

Special Scripts. This section lists the tools that usually are not accessed through a user interface. These scripts are used in conjunction with other processes or are used for very special purposes and should only be used by your IT staff.

Security. The security options available within Degree Works.

Central Authentication Service single sign-on. This section details the Central Authentication Service (CAS) that can be used to integrate Degree Works with portals and other Web applications.

System Administration. The various tasks that are required to administer Degree Works.

System Performance. This section provides information about system performance management, configuration options to manage performance, and troubleshooting guidelines.

Overview

The Degree Works product allows an institution to automate the degree auditing and student advising processes. It provides a means and method for entering degree requirements from a college catalog into the computer and analyzes a student's academic progress, providing output stating what institutional requirements have been met, and what still needs completion.

The users of the requirement definition input process may include registrar's staff, academic administrators, and data center technical staff. The consumers of the results of the degree audit process will represent an even wider spectrum, including students, admission applicants, academic advisors, academic administrators, and registrar's staff.

This wide range of client types was an important operative consideration in the design of Degree Works.

The goals of the Degree Works design are:

1. Provide accurate information for students and advisors about degree progress.
2. Provide easy-to-read advisory format.
3. Provide easy-to-use requirement building process.
4. Provide powerful language for requirements definition.
5. Provide robust exception management.
6. Provide security for access.
7. Generate a cogent description of all degree requirements.
8. Report requirements that have been satisfied and those that are yet to be completed in a concise manner with 100% accuracy (where accuracy means that a student's courses are applied to appropriate requirements).
9. Inform the registrar that degree requirements are nearing completion or are completed.
10. Provide "what if" capability for on-line viewing of potential evaluations for changes of degree, major, minor, concentration, or placement evaluation for new transfer students.
11. Provide fast and efficient processing of evaluations.
12. Generate individual and institutional-level output, such as requirements worksheets and course demand data.
13. Include query capability for students through a campus network.
14. Be versatile and provide configuration settings to handle opposing client requests.
15. Provide robust exception handling, including but not limited to, the definition of custom requirements for a student.

A Degree Works **language** consisting of certain keywords and syntactical rules is used to define institutional requirements into the computer. The institutional requirements are entered using a program called **Scribe**, resulting in a series of **requirement blocks**. These blocks are then read by the first of the two Degree Works processing engines, the Parser Engine. The **Parser Engine** validates the requirement blocks, assuring they are lexically and syntactically correct so that they may be properly interpreted by the Auditor Engine.

The second Degree Works processing engine is the **Auditor Engine**. The Auditor Engine reconciles student academic data from the student information system with the requirement blocks that have been built by users and then validated by the Parser Engine. The Auditor Engine actually evaluates the student course data against the appropriate requirement blocks, determining which academic requirements have been satisfied and which await completion. The audit results are stored in a database for reporting and review by the institutional staff.

The third Degree Works processing engine is the **Output Engine**. The Output Engine interprets the audit results and produces printed and online audit reports. Online viewing of audit results is accomplished using **Degree Works on the Web**, which may be made available to Registrar, faculty or students. It is through Degree Works on the Web that exceptions and substitutions are entered as well as advisor notes. Security to control "who accesses what" is enforced by Degree Works on the Web.

DAP, Degree Audit Process, is an abbreviation used internally for Degree Works. The database name is "dabdb" and the program names begin with "DAP". Throughout this document, "DAP" may used as shorthand for "Degree Works". Compared to prior versions of "DAP", Degree Works is focused on developing a user-friendly language and robust Parser Engine to articulate the wide range of academic requirements among institutions of higher education. Concentrated efforts were made to construct an Auditor Engine flexible enough to evaluate consistently a wide range of diverse institutional procedures and policies. Finally, improvements have been made to the screen interface and hard-copy output in order to better meet the auditing and advising needs of end-users.

Degree Requirements

The Parser Engine translates the requirements language into a format to be used for audit. The Parser Engine is a program called Dapparse (DAP13) with the following characteristics:

- Resides on the host computer
- Accepts files of requirements (created with Scribe)
- Parses requirements, i.e. translates the rules into syntax and remarks files
- Catches errors by validating disciplines and courses
- Returns error files if any errors are encountered during the parse
- Stores the syntax and remarks files on the host computer
- The parsed text is saved in the DAP database on the host computer

Scribe is the input mechanism for entering degree requirements. You can either enter the requirements yourself or contract with Ellucian to enter them for you. In either case, you must gather together the course catalogs and other documents that define your institution's degree requirements.

Requirement Blocks
Degree Works stores requirements in requirement blocks. Blocks are user-defined and are likely to include degree, major, minor, and concentration, but may also include sets of requirements that are unique to an institution (e.g., community service). The blocks will not necessarily be hierarchical, and links among blocks may be established but are not required. The order of the blocks and the linkages between the blocks are determined by the user. There is no limit to the number of blocks that may be used to construct the requirements for a degree program.

Degree Works determines which blocks to use for a particular student by checking standard data on the student record for degree, major, minor, or concentration, or by client-defined data on the student record to indicate special activities or programs (e.g., ROTC). Blocks may be global as well (e.g., general education), so a link to student data may not be required.

Users are able to define the relationships among requirement blocks. Some will be sequential and hierarchical; some will be linked one to another (e.g., a concentration that cannot exist without an associated major); some will be conditional on student data that will determine blocks and point to other blocks (e.g., ROTC); and some will be isolated and required of all students regardless of their characteristics (e.g., a requirement that all students perform some kind of community service).

The requirement blocks may be defined for student attributes which may include school, college, department, major, minor, concentration, location, catalog year, and other client-defined characteristics. Custom sets of requirements by student may also be defined.
Unlimited courses are allowed within requirement blocks and unlimited nesting of requirements is allowed within a block.

The Scribe language for defining requirements lets users specify requirements by number of classes, by number of credits, or by a combination of both.

The definition of courses that can be taken to fulfill a requirement rule allows for the robust use of the wildcard (@) and also for use of a range operator (:) for course numbers. The wildcard (@) indicates one or more occurrences of any character.

Example: MATH@ = all classes in the discipline MATH
Example: MATH1@ = all classes in the discipline MATH were the first digit of the course number is a "1"
Example: MATH100:199 indicates all courses from MATH100 through MATH199.

The definition of that portion of the course key that indicates the course discipline may require the use of a delimiter. If no course delimiter is specified then Degree Works assumes discipline is an alpha code.

Example: ART1200 = ART
Example: ART1 200 = ART1
Example: ARTA200 = ARTA
Example: ART A200 = ART

Degree Works allows for non-course requirements. These requirements may include theses, performances, attendance at required events, work experiences and should be allowed individually, in groups by student characteristics, or by requirement block.

There is an unlimited text capability for annotating the requirement blocks. This text can be used, in whole or part, to provide readable reports where requirements are described. The text may include comments internal to the requirement block or may be text for use on user reports.

Degree Works provides a mechanism for pointing multiple catalogs to the same requirement block to avoid duplicating blocks when requirements do not change. This is accommodated by identifying the catalog year as a range of begin/end years rather than as a single year.

Auditor Engine

The Auditor Engine reads the student's academic data, assembles the requirement blocks, performs the audit, and stores the audit results in the DAP database. The Auditor Engine is a program called Dapaudit (DAP14) with the following characteristics:

- Resides on the host computer
- Accepts student data from the host (degree data, class data)
- Accepts "what-if" student data from the end-user
- Selects requirement blocks to be audited based on the degree data
- Processes the audit using the audit algorithm and site-defined configuration settings
- Stores the audit results in the Degree Works database on the host computer

The following student data is passed to Degree Works from the student database. These elements constitute the minimum data needed by the Auditor Engine but additional "custom" data may also be passed to the Auditor for evaluation or subsequent inclusion on audit reports.

From the student course record:

```
ID
school
college
course discipline
course number
credits
grade (letter and number)
grade points (grade number * credits)
term
course type (resident, transfer)
grade type (regular, pass/fail)
credits type (e.g., academic, clep, ap)
location (site of class)
transfer xref (local course equivalence)
class status (added, dropped, withdrawn, repeated)
```

From the student academic record

```
ID
school
degree(s)
  catalog year
program(s)
  catalog year
college(s)
  catalog year
major(s)
  catalog year
minor(s)
  catalog year
concentration(s)
  catalog year
specialization(s)
  catalog year
liberal learning(s)
  catalog year
non-course requirement data
  (e.g., exams, performance, language, comments)
client-defined data (e.g., ROTC, religion)
```

From the course catalog record:

```
course discipline
course number
catalog year
course equivalents
```

Student data drives the Auditor Engine. There are standard characteristics that would be expected from any student database including degree, major(s), concentration(s), minor(s), catalog year(s). There may be client-specific characteristics that also drive the Auditor Engine, including (but not limited to) ROTC and religion.

Requirement blocks are assembled for processing by searching for blocks that match student attributes, looking first for the degree block. Blocks are matched first by ID. If no blocks matched by ID, other student attributes are combined to find blocks with matching attributes. Configuration settings control the processing of the requirements.

The Auditor Engine matches all resident courses, transfer courses, and non-course activities to the requirement blocks that have been assembled. Resident courses can be evaluated in the context of the selected course catalog, taking into account changes in course identifiers (e.g., recycled courses/AKA's by catalog year). Courses may be excluded from analysis by the Auditor

Engine based on credit type or grade type, (e.g., academic bankruptcy classes or classes that do not carry academic credits).

Degree Works can handle transfer courses that do not map directly to a resident course but do map to a range of courses or to any course in a discipline. Direct equivalencies are not required, for example, transfer courses that are different in credit value, courses that are a series of two at one institution and three at another, or situations when course transfer will be allowed but no direct equivalent exists.

Exception handling in Degree Works lets the user lock-in evaluation decisions so that any subsequent running of an audit will not undo them. These decisions might include substitutions of specific courses for specific requirements, waivers of classes, exemptions from certain credits or requirements. Implicit in this is the option to "unlock" such decisions.

Degree Works supports an option to "split" credits from a single course and apply the remaining credits to other requirements. This is done either through requirement definition in Scribe or through a special kind of exception.

It is possible within Degree Works for the user to indicate that a requirement is complete using Exceptions even when the Auditor Engine cannot complete it through.

Degree Works allows variations in the processing of repeated or retaken courses. Courses that may be repeated for credit up to a maximum number of credits or courses must be counted appropriately, and those courses may or may not be limited to one per term. Courses that may not be repeated for credit but are repeated to improve a student's GPA can be identified and the GPA can be calculated correctly. The rules for applying credits from repeated classes and the rules for calculating GPA for repeated classes are site-defined within a set of repeat policies provided by Degree Works.

For purposes of evaluation, the Auditor Engine assumes all courses are applied to requirements in an exclusive manner, i.e., one class and the associated credits apply to one requirement. The nonexclusive application of classes can then be specified with the appropriate limits. The range of users' needs on this issue is great. It includes everything from "you can use anything wherever it fits" to "you can't use anything twice". The middle ground is "no more than 10 credits that applied toward the major requirements can also be applied toward minor requirements."

For the processing of student records against requirement data, the Auditor Engine uses a "best fit" algorithm. To accomplish this, the Auditor Engine may have to perform a number of passes through the course information. For example, on the first pass, all courses that fit a requirement are placed. Classes that can be applied to multiple rules then needed to be weeded out, keeping the "best fit". The fit is made against requirement blocks based on a client-defined order.

To identify those students nearing requirements completion, the Auditor Engine calculates the overall percent complete and percent complete for each requirements block (e.g. Major, Minor) The Notes capability in Degree Works provides unlimited text capabilities for recording notations on the student's record. This includes, but is not limited to, special circumstances, advisor notes, reasons for exceptions, who made decisions and when.

Grade Point Average is calculated overall and by requirement block. The cumulative GPA calculated by Degree Works may be different from that calculated by the student system due to Degree Works ability to process repeats and courses that exceed limits. Users have the ability to exclude specific courses from block GPA calculations via the NOTGPA reserved word.

Output Engine

The Output Engine reads the audit results from the database amd formats the results into an XML or JSON document. The Output Engine consists mainly of one subroutine, dapextract (DAP15) with the following characteristics:

- Resides on the classic server
- Extracts the requested audit from the Degree Works database
- Formats the audit results as an XML or JSON tree

The presentation layer takes in the XML or JSON tree and formats it using XSL or some other tool. In the Dashboard there are several Degree Works stylesheets that create audit worksheets while in the Responsive Dashboard the react-js code creates the worksheets from the JSON audit. In batch mode FOP is used to convert the XML trees into PDF.

Scribe

The Scribe Language is used to define the requirements rules. Each Reserved Word has rules about its use.

The Scribe application is used to enter the degree requirements using the Scribe language. A specialized "word processing" window is used to allow the user to enter the requirements using a natural language. Templates in the help guide can be dragged-n-dropped into the edit window to make rule creation easier. After entering the requirements, they are parsed for syntactical and lexical errors. If no errors occur then the requirements are saved to the Degree Works database with database "tags" that describe the degree requirements.

Transfer Equivalency

Transfer Equivalency is the Transfer Equivalency Coursework Articulation Data Entry Interface. Transfer Equivalency is a module allowing schools to set up transfer equivalencies, enter transfer course data from transfer transcripts, and articulate the transcripts against the mapping created for each transfer institution. Transfer Equivalency interface has the following functions:

- create mappings of course equivalents from transfer institutions
- create transcript information for specific students, including courses and test scores.
- perform an articulation of classes processed.
- perform a degree audit on articulation results, audit classes in student system, and view a degree audit report.

Transit

Transit provides a variety of batch reporting capabilities, including various audit reports, and enrollment demand statistics. The Transit application is typically accessible only to authorized registrar staff.

Degree Works Dashboard

The Degree Works Dashboard is used to perform the following functions while enforcing security about who can execute which functions:

- run a new degree audit against the student's real academic data
- run a what-if audit against the student's requested academic data
- run a look ahead audit based on projected class data
- review a previous degree audit
- view audit results, selecting the desired output format
- view historic audits
- view student data audits for troubleshooting purposes
- exception management
- student academic planner function
- notes entry record
- enter petitions for exceptions
- GPA calculators.
- Student Educational Planner

Curriculum Planning Assistant

The CPA tool allows administrators a macro view of the student audit results. Audit results are placed in the database in a format that may be queried using standard tools such as Crystal, Hyperion, Cognos, etc. The audit results are placed in the **dap_result_dtl** table with class records being stored in the **dap_resclass_dtl** and noncourse records being stored in **dap_resnoncr_dtl** table.

You can use any SQL-complaint tool to access the CPA data. Reports are supplied as part of the Ellucian ODS but no reports are supplied as part of the Degree Works toolset.

Please also see the Advanced Reporting Technical Guide for more information.

Controller

Controller is a web application used to manage user access to Degree Works functionality. Typically, only a few staff members in the Registrar's or Information Technology office are granted access. It allows users of Degree Works, to codify, store, maintain, and validate data values and stores these values or codes in discrete units called Tables within an overall structure called the Universal Code eXtension (UCX).

Glossary

This glossary presents terms that are used throughout this document. It is important that the reader be introduced to these terms early so they are presented at the beginning of the document. However, this section is also intended to be used as a reference while reading the rest of the document.

The Scribe language consists of words that describe degree requirements. Many of the words are "reserved" because they have a special meaning in the Scribe language. These reserved words, also called keywords, are not case-sensitive. The examples in this document may show the Scribe keywords in upper-case but mixed or lower-case is also acceptable.

The Scribe language is not dependent on indentation. The end-user should develop a consistent style of indentation for readability. When Ellucian Scribes a catalog for a client, a consistent look is delivered.

Throughout this document, optional letters and words appear within square brackets. For example, CLASS[ES] means that either CLASS or CLASSES can be used -- both are valid in the Scribe language.

In this glossary, words in italics refer to another entry in the glossary.

The basic components of the Scribe language are:

Block a set of degree requirements written in the Scribe language. Requirement blocks are defined by the user. Most colleges will have blocks for degree, major and minor. A block consists of a block header, followed optionally by one or more rules or remarks, followed by "END.". Example: General Education requirements consist of 6 credits of Math, 6 credits of English, 6 credits of a foreign language, and a maximum of 6 pass-fail credits. In the Scribe language, this requirement is:

```
Begin
MaxPassFail 6 Credits
;
6 Credits In MATH @ Label “Math requirement”;
6 Credits In ENGL @ Label “English requirement”;
6 Credits In FRE @, GER @, SPA @, RUS @, CHI @
  Label “Language requirement”;
End.
```

Block Header contains the degree requirements that apply to all courses satisfying rules in the block. Keywords in the block header describe the block as a whole and are applied to the entire block, not to specific rules. The block header is composed of the word BEGIN, followed by optional block qualifiers (such as minimum grade, maximum transfer credits), followed by a semicolon. Example:

```
Begin
MinGPA 2.0
MaxTransfer 30 Credits
;
```

Block Qualifier a Scribe keyword that describes a degree requirement that applies to all courses satisfying rules in the block, such as minimum GPA or maximum number of transfer courses. Each block header can have zero or more block qualifiers. Example:

```
MaxTransfer 30 Credits
```

Block Type	the primary database tag, a characteristic of the block. It describes what kind of requirements are in the block. Examples: `degree=BA` `major=MATH` `minor=BUS.` In these examples, the block types are DEGREE, MAJOR, and MINOR respectively. The block type values are BA, MATH, BUS respectively.
Client-defined Code	a string of up to 12 alphanumeric characters that represents a valid value for a piece of student data. A code is a shortened representation of a literal, e.g. WA is the code for Washington state. Client-defined codes are created by each Degree Works customer and are unique to the institution, but are shared by all offices and staff across the institution.
Comment	a string of free-text following a pound sign or exclamation point. Comments are entered into a requirements block as an annotation that explains something to the person maintaining the requirements block. Comments are never printed on audit output -- they are for internal use only. Example: `3 CREDITS IN BIO @;  #doublecheck with biology department`
Course List	a list of required classes, where each class is represented by a course key. Each course key consists of an academic discipline and a course number. Either the discipline or the course number can include a wildcard symbol (@). Example: `BIO 100, CHE 115, PHY 1@;`
Custom Block	a requirements block that is defined for non-standard student data or for a specific student. A custom block either has a database tag of ID for a particular student or has a block type of OTHER. Examples: custom requirements for ID=12938593, requirements block for community service (block type is OTHER and block type value is COMMSERV).
Custom Data	non-standard student information. Custom data is not part of Degree Works's standard data for a student but an institution may have degree requirements that are based on non-standard student data. Examples: ROTC, religion.
Database Tag	a characteristic of the requirement block that is stored in the DAP database. These characteristics are used to match the requirements to the students. The database tags are: beginning catalog year, ending catalog year, college, concentration, degree, student ID, liberal learning, major, second major, minor, other, program, school, and specialization. One of these database tags is designated by the user as the primary database tag, the block type. The database tags are entered when the degree requirements are added to the DAP database. They are related to the Scribe language only because the language refers to BLOCKTYPE and BLOCK, which tell the Auditor Engine which primary database tag to use when finding a requirement block for a student.
Goal	A field of study (such as MAJOR, MINOR, PROGRAM). A student can have a virtually unlimited number of goals stored on their student record in Degree Works.

GPA

Grade Point Average, sum of grade points divided by total graded credits earned. The GPA is calculated to 3 decimal places, with a maximum of 999.999. Examples:

```
3.125, 2.5.
```

Integer

a whole number between 0 and 999 inclusive. Examples:

```
1, 222, 35.
```

Keyword

a Scribe reserved word. Scribe keywords are part of the Scribe language and are restricted as to how they can be used. Examples:

```
Begin, MinGPA, MaxTransfer
```

Linked Block

a requirements block referenced in another block (in a BLOCK rule). The requirements in the linked block are later added by the Auditor Engine in place of the BLOCK rule. A linked block is useful when requirements need to be repeated in multiple blocks. For example, if both the Bachelor of Arts degree and the Bachelor of Science degree have the same foreign language requirements then three Scribe blocks could be created:

```
Begin
120 Credits;
# Bachelor of Arts degree requirements block
6 Credits In ART @;
1 Block (OTHER=FORLANG);
End.

Begin
126 Credits;
# Bachelor of Science degree requirements block
6 Credits In MATH @;
1 Block (OTHER=FORLANG);
End.

Begin
MinGrade 2.0;
# Foreign language requirements block
6 Credits In FRE @, GER @, SPA @, RUS @, CHI @;
End.
```

Non-course

a degree requirement, such as a recital, chapel, test, or thesis, that is not a course for which a student registers but whose completion is recorded in the student information system. Example:

```
1 NonCourse (RECITAL).
```

Real

a number between 0 and 999.999 inclusive. A real number is either an integer or an integer followed by a decimal point and up to 3 decimal digits. The decimal point is only required if there is a number after the decimal point. Examples:

```
1, 2.0, 3.125, 25.375, 120.001.
```

Relational Operator	a character that signifies comparison of two values. In the Scribe language, relational operators are: = (equal), > (greater than), < (less than), >= (greater than or equal), <= (less than or equal), <> (not equal). Examples: `DEGREE = BA, Catalog_Year >= 9495.`
Remark	a free-text string that describes a requirement using natural language similar to the language of the college catalog. Example: `REMARK "Must take either 6 credits in a foreign language or pass a test."`
Requirement Block	see *Block*.
Rule	a specific degree requirement. A rule is a Scribe language statement of one of the degree requirements. The rule contains a list of courses, blocks, block types, or non-courses, followed by zero or more qualifiers, (such as minimum grade, maximum number of credits per term), and terminated by a semicolon. For example, the requirement in the catalog is 6 credits of upper-division mathematics. The Scribe rule is: 6 Credits In MATH 300:499.
Rule Qualifier	a Scribe keyword that describes a degree requirement that applies to all courses satisfying a particular rule. Rule qualifiers are Scribe reserved words that indicate properties of the courses used to satisfy a rule, such as minimum grade or maximum number of transfer courses. Each rule can have zero or more rule qualifiers. Example: MinGrade 2.0.
Standard Data	the standard student information supplied to Degree Works by the student system. Typically this data includes ID, Name, SSN, degree, majors, minors, concentrations, specializations, college, school, program, catalog years, and course data.
Token	a single word or a string within quotes in a requirements block. Tokens are separated by one or more spaces, tabs or newline characters. Examples: `"This is a remark"` is one token. `3 CREDITS IN ENG 100;` is six tokens (3, CREDITS, IN, ENG, 100, and semicolon)
Token Class	a category of tokens that have a similar purpose in the Scribe language. For example, the 'and-or' token class consists of the tokens And and Or, and they both serve as connectors within the language.
UCX	a collection of common code values stored in the Universal Code eXtension file. These codes are used by Degree Works to validate codes used in the requirements for schools, college, concentration, degree, liberal learning, major, minor, program and specialization.

Wildcard	a symbol representing one or more occurrences of any alphanumeric character (i.e. A-Z, a-z, 0-9). The wildcard in Degree Works is the "@" sign. The wildcard is used for pattern matching of disciplines and course numbers. For example, "BIO @L" represents any BIO courses with course numbers ending in "L". PE@ 1@ represents any course with a discipline beginning with "PE" and a course number beginning with "1".

Special Topics

To help you use Degree Works effectively, there are a variety of special topics that need to be discussed and elaborated:

Adding Custom Data Items

Custom data items are pieces of data that are not part of the standard data items passed from the student system to Degree Works Custom data items are also data that are available for use in an IF expression in Scribe. They are defined in UCX table UCX-SCR002. The primary use of custom data items in Degree Works is in an IF expression, where a requirement varies based on student data. For example, if Catholic students must take RLGN 300 and non-Catholic students must take RLGN 305 then the requirement could be written as follows:

```
If (RELIGION = CA) Then
    1 Class in RLGN 300 Label “Catechism”
Else
    1 Class in RLGN 305 Label “Religion for the masses”;
```

The above requirement will parse with an error indicating RELIGION is invalid. In order to add RELIGION or another piece of data as a custom data item, follow the steps outlined below.

Step 1:

Determine where the custom data item is stored in the student system. For example, RELIGION could be stored on a user-def field on the RAD-PRIMARY-MST or in the RAD-CUSTOM-DTL – which is the most logical place to store custom data.

Step 2:

If you want to use the data item in an IF expression in Scribe then it must also be added to UCX-SCR002.

Step 3:

Check UCX-SCR002 to see that the data item (e.g. RELIGION) is not part of the custom data items. If it is, then use the code from UCX-SCR002 in the IF expression and skip to step 6 of these instructions.

Step 4:

If the data item is not in UCX-SCR002 then add the data item to UCX-SCR002. Choose a name for the data item that is from one to twelve characters long, e.g. RELIGION. This name is the Degree Works name of the custom data item. It is usually upper-case. Find out the element number of the data item by looking in UCX-SYS999. For example, the element number for the Custom-Value is R323. Use Controller to add the new code to UCX table UCX-SCR002. Create a new record with a code of “RELIGION” or whatever name you want to use in Scribe. In the Description field – enter something meaningful (e.g. "Religious affiliation"), The Data Element field– should contain your value from UCX-SYS999 – in this case R323. Since this value is coming from a “Dtl” record we need to specify which record to read. This is done by filling out the Edit Element information. Here we want the record with RELIGION in the Custom-Code field – which is element R322. The Type field should be filled with EV and the value should be “RELIGION”.

See the UCX documentation of table UCX-SCR002 for examples.

Adding NonCourse Data Items

NonCourse data items are pieces of data from the student system that are non-standard requirements, not a traditional course but something required for graduation. They are defined in UCX-SCR003. Examples of NonCourse data items are music recital, art gallery show, chapel attendance, and placement exams.

```
1 NonCourse (RECITAL)
  ProxyAdvice "At least one recital is required"
  Label RECITAL "Recital";

1 NonCourse (MATHEXAM > 12)
  ProxyAdvice "A math exam with a score better than 12 is needed"
  Label MATHEXAM "Math exam";
```

The above requirements will parse with errors if the NonCourse code in parentheses is invalid.

To add MATHEXAM or another piece of data as a NonCourse data item, follow the steps outlined below.

Step 1:
Determine where the noncourse data is stored in the student system. For example, MATHEXAM might be stored in the RAD-CUSTOM-DTL, but often noncourses are stored in the RAD-NONCRSE-DTL. You can check the Student Data Report to verify where the data is being housed for your students.

Step 2:
If the data item is not in UCX-SCR003 then add the data item to UCX-SCR003. Choose a name for the data item that is not longer than twelve characters, e.g. MATHEXAM. This name is the Degree Works name of the noncourse data item. It should be upper-case. If the data is not on the RAD-NONCRSE-DTL, find out the element number of the data item by looking in UCX-SYS999. Use element number 0000 if the value is on the RAD-NONCRSE-DTL.

Use Controller to add the new code to UCX-SCR003. Add a new record and enter the Degree Works name for the data item, e.g. MATHEXAM, as the key. In the Description field enter a description of the data item (e.g. "Math Placement Exam"), followed by the element number in the Data Element field. The Edit Element, Type and Value fields are used if the value you are getting is from a “detail” record.

See the UCX documentation of table UCX-SCR003 for examples.

Additional Advisee Filtering

Overview

This feature allows a user to be assigned the capability of having not only their advisees preloaded on the web screen, but also all students in a designated major. This is particularly useful for Department heads especially if they have no assigned advisees. This functionality could also be used to allow any advisor access to all students (majors) in their department in addition to their advisees, or instead of any advisees. This is controlled on a user by user basis via the advisee filter fields using Controller. Access to this functionality is granted via a service (SDDEPART) granted to the appropriate UserClass.

Services Affected

SDDEPART service – allows Advisees and students with the user's advisee filter(s) to be listed

It is automatically included as part of the DEPT UserClass.

SDSTUMY service was modified to also check the user's advisee filter(s) to preload students with the specified major in addition to the user's advisees. This allows advisors to be given this same capability as Department Heads if needed.

Setup Procedure

Step 1:
Be sure the appropriate user is given the necessary UserClass to allow access to this functionality.

Step 2:
Use Controller to define the advisee filters (up to 10 majors) for the user.

EXAMPLE:
In the case below, the user LOCKHART was given DRAM as an advisee filter. The user had a UserClass of ADV, which has the SDSTUMY service associated with it. So when LOCKHART logged into Degree Works, his advisee list was preloaded, and all students with a major of Drama (DRAM) were also added to the list.

If the user was not assigned the DRAM advisee filter, then the only students listed would be LOCKHART's advisees.

Degree Works Bridge

The Degree Works Bridge is a mechanism for loading the Degree Works data structure with relevant data from the student database. It has a well defined API-format that may be used by a university that is extracting data in a static fashion on a regular basis, or in a dynamic fashion on-demand. Degree Works also has a “native” integrated extract for selected student systems.

Static Bridge

Student data is typically moved from the University’s student database by using the batch process RAD11. This process can be scheduled to run on a regular periodic basis or can be launched on a manual basis. For more information about this process, please refer to the Bridge Interface Format Technical Handbook.

Dynamic Refresh

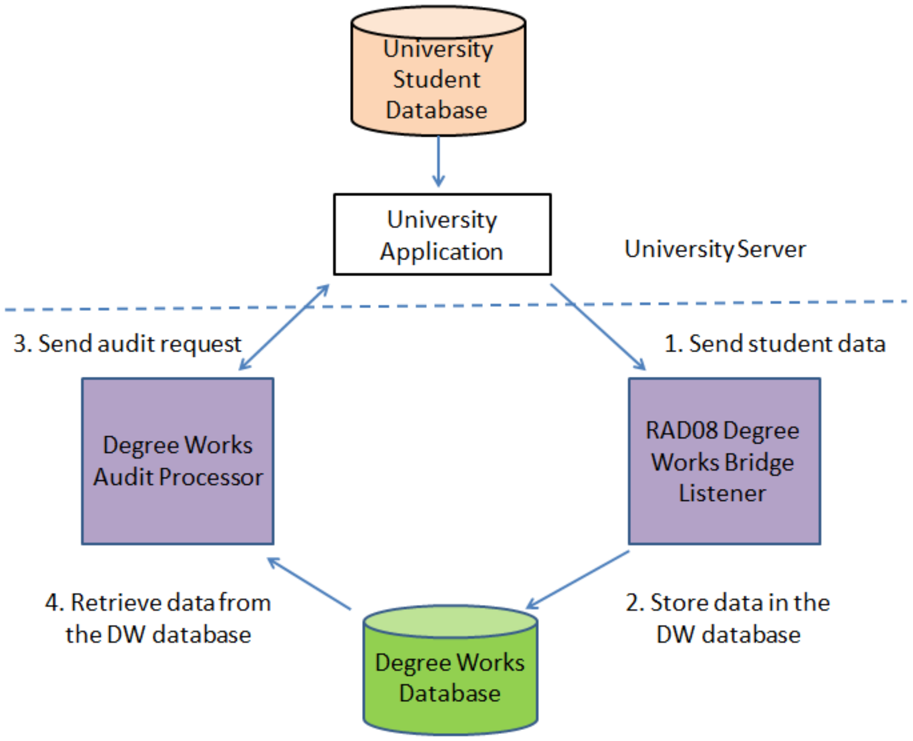
Although the recommended “best practice” is that the RAD11 static bridge be used to load student data in batch mode on a regular periodic basis, the RAD08 dynamic bridge may be used to load student data throughout the course of the day as events trigger the need. RAD08 is a daemon process that listens for incoming bridge requests and saves the incoming data to the Degree Works database. Once a request is sent to RAD08 and processed, subsequent run-audit requests will use the new data for the given student.

The University must write an application which issues the request for Refresh and sends it to RAD08 running on the Degree Works classic server.

The number of RAD08 processes that listen for and process bridge requests is controlled by the –C flag in the RAD08JOB. The –C flag controls the number of children the parent RAD08 will create to listen for requests. Since the parent also listens for and processes requests, the total number of RAD08 processes will be the number of children plus one.

You may decide to URL encode the request before sending it RAD08. If URL encoding is being used, the WEB84_URL_DECODE flag in rad08job must be set. Setting this flag tells the WEB84 subroutine that RAD08 calls to decode the request when breaking out the name-value pairs contained within.

Banner sites using the “native” Degree Works Integrated Interface should not use this RAD08 process. Please refer to the Banner Considerations document for more information on the integrated Banner dynamic refresh process.


University Student Database
University Application
University Server
3. Send audit request
1. Send student data
Degree Works Audit Processor
RAD08 Degree Works Bridge Listener
4. Retrieve data from the DW database
2. Store data in the DW database
Degree Works Database

Dynamic Refresh Process Flow

1. The University's application sends request to Degree Works to store student data in the Degree Works database.
2. Degree Works stores the student data in the Degree Works database.
3. The University's application sends a Web run audit request to Degree Works.
4. Degree Works reads the student data from the Degree Works database, processes the audit and returns the audit report to the user.

Refresh Request
The refresh request is in name-value pair format and contains the following information:

Name	Description	Length
ACTION	"REFRESH" for student data refresh	07
STUID	ID of the student for which the refresh is needed	10
RECCOUNT	Count of student records returned as REC name-value pairs	04

After the header information, student data records must be sent in the refresh request.

Name	Description	Length
REC	Repeated up to 256 times. Each REC name-value pair contains a record in Bridge-Interface-Format including a HEADER = 28 bytes, DATA = 972 bytes for a maximum total of 1000 bytes.	1000

Example Refresh request (not all student records are shown):

```
ACTION=″REFRESH″&STUID=″123456″&RECCOUNT=0003&
```

```
REC=”123456    R010PRIM        A 123456        Johnson, Joyce”&
…
REC=”123456    R020STUD        A 123456        000000    20001”&
…
REC="123456    R050BIOG        A 123456        542661234 19801231"&#
<$ENDMSG$>
```

An ampersand “&” separates each name-value pair, with an ampersand and pound-sign “&#” signaling the end of the data. The end of the entire response is signaled by “<$ENDMSG$>”. Each value may or may not be enclosed by quotation marks but quotation marks are required if the value may contain an ampersand. All data for a student must be bridged; do not bridge just the changed or new records.

To increase the efficiency of the data transfer over the network, trailing spaces from each REC name-value pair should be removed. If only the first 48 bytes contain actual data then the trailing spaces should be removed before placing the record in the REC name-value pair. Records must already be sorted by the bridge-indicator field (R010PRIM, R020STUD, etc) before sending them across the network.

Refresh Response
The RAD08 process returns a message containing status information concerning the processing of the refresh request. The name-value pairs are listed in the tables below. Note that “Length” is the maximum length in bytes.

The status information in the Refresh Response includes:

Name	Description	Length
STATUS	OK or FAIL. Return FAIL is error encountered.	04
ERROR	Error number specifying error encountered; e.g., 4321. Omit if no error encountered.	04
ERRMSG1	Error message string 1 describing error. Omit if no error.	80
ERRMSG2	Error message string 2 describing error. Omit if no error.	80

Example Refresh Response:

```
STATUS=OK&ERROR=""&ERRMSG1=""&ERRMSG2=""&ACTION=REFRESH&
STUID=123456&#<$FINISHED$>
```

Error numbers in the range of xxxx-xxxx indicate specific error conditions when attempting to process the refresh request. The error number returned along with the error message strings will be returned. The error messages may say “Student ID number not found in student system” or “Database error occurred when writing student classes”. An error is only returned if the data cannot be saved, is missing, or is invalid.

Refresh Thank You
After the University receives the status response a thank-you reply should be sent to the Degree Works refresh listener to indicate that the response has been read and it is ok to close the socket connection.

The thank-you message should be a simple “<$THANKYOU$>” value. When the Degree Works listener receives this message the current socket connection will be closed.

Equivalent Course Tracking

Equivalent courses are courses that changed discipline or number at some point in time. These equivalents are important to Degree Works when a student is being evaluated against a catalog year that uses a different course-key than the course-key on the student's class. If Degree Works does not know about the equivalency, the rule will never be satisfied because the course-key on the student's record does not match the course key Scribed in the rule. The solution to this problem is to set up the equivalence in the dap_eqv_crs_mst.

For example, John took ENGL 101 in the fall of 1990, took a year off and then returned to school as a member of the class of 1996. ENGL 101 changed to ENGL 115 in the fall of 1995. The course catalog requirements include ENGL 115 as a requirement, not ENGL 101 (which is now a different course). John's ENGL 101 course from 1990 should fulfill the ENGL 115 requirement of 1996.

In order for the Auditor Engine to automatically evaluate ENGL 101 as ENGL 115, an equivalent course record must be created. Once created, the equivalent courses serve as a map of course number and discipline changes. The equivalent course record is not specific to a student. The equivalent course record is stored in the dap_eqv_crs_mst.

Special Notes:

1. This document assumes your institution does not reuse course keys. A simple example of reusing course keys: MATH 103 was "Advanced Algebra II" during catalog year 19992000, then was changed to MATH 106. Then in 20002001 MATH 103 was reused to be a completely different course such as "Pre-Calculus I". If your institution has reused course keys, do not attempt to use the dap_eqv_crs_mst as described here; please contact Ellucian for assistance in setting this up.

2. The structure of the dap_eqv_crs_mst is:

class_dtl information ("When the course was taken")

TERM-CATALOG-YR	X12	Degree Works catalog year (UCX-STU035) mapped from the term the class was taken. Mapped from term to catalog year via UCX-STU016. The character "@" can be used as a wildcard. Using the wildcard here tells the auditor that it does not matter when the student took the class.
OLD-COURSE-DISCIPLINE	X12	Discipline from the course-key of the class taken (UCX-STU352).
OLD-COURSE-NUMBER	X12	Number from the course-key of the class taken. The character "@" can be used as a wildcard (primarily used if there is a discipline change, i.e. ENGL has changed to ENG).

Course Catalog Information (“When the equivalency exists”)

TARGET-CATALOG-YR	X12	Degree Works catalog year (UCX-STU035) mapped from the student's catalog year. Mapped from student catalog year to Degree Works catalog year via UCX-XXX358. The character “@” can be used as a wildcard. Using the wildcard here tells the auditor to apply this equivalency to all catalogs.
NEW-COURSE-DISCIPLINE	X12	Discipline from the course-key in the target catalog year – what the discipline is in requirements written for the target catalog year.
NEW-COURSE-NUMBER	X12	Course number from the course-key in the target catalog year – what the course number is in the requirement block written for the target catalog year. The character “@” can be used as a wildcard (primarily used if there is a discipline change, i.e. ENGL has changed to ENG).

Equivalent Course Tracking: Standard Setup

There are two steps you must follow to utilize the dap_eqv_crs_mst.

Step 1. Update all prior catalogs with the change.

If it is a discipline code change, find all instances of the discipline in the prior catalogs and change it to the new discipline code. If it is a specific class change, find all instances of the class in the prior catalogs and change it to the new class.

Step 2. Add the equivalency to the dap_eqv_crs_mst.

a) Bridge the records using the R190DEQV layout (Refer to Degree Works Bridge to Student Record Systems Technical Specifications). You may bridge the entire contents of the dap_eqv_crs_mst or bridge only new records as needed.

b) Use Controller’s UCX-CFG070 screen to maintain your equivalences. You should then run the dapucx2eqv script. Be careful about using both the bridge method and UCX-CFG070 to maintain equivalences.

Example 1: Discipline Code Change

The school has decided to change the course discipline code from ENGL to ENG. The school will either manually change the requirement blocks referencing ENGL to say ENG, or the school will set the UCX-CFG020 DAP13 Process Equivalences flag to Y and run DAP16 to reparse all of the blocks. With this flag set to Y, the underlying block requirements will now say ENG instead of ENGL.

For example, “1 Class in ENGL 101” will be changed to “1 Class in ENG 101”. This will happen through the laborious manual process or by running DAP16 with that flag enabled. Though the changes will not be reflected in Scribe when the block is viewed.

However, the auditor will now not apply any ENGL 101 classes to this rule. Setting up dap_eqv_crs_mst records is needed to tell Degree Works to apply ENGL classes against ENG rules.

Step 1. Update prior catalogs with the new course number:
In Scribe, locate any instance of ENGL and change it to ENG for every catalog – or rely on DAP16 with the UCX-CFG020 DAP13 Process Equivalences flag set.

Step 2. Add an entry to UCX-CFG070 in Controller
Catalog year class was taken = @
Old Course Discipline = ENGL
Old Course Number = @
Student's Catalog Year = @
New Course Discipline = ENG
New Course Number = @

Translation: "Degree Works will apply all ENGL courses against ENG requirements."

Example 2: Course Number Change
The school has decided to change the course number for the Math Statistical Analysis class. In the past, it had been listed as MATH 176. From now (catalog year 20042005) on, this class will be listed as MATH 200. The current catalog now contains rules such as "1 CLASS in MATH 200". Without a dap_eqv_crs_mst, Degree Works would not apply MATH 176 to the MATH 200 rule. With the dap_eqv_crs_mst, you are telling Degree Works that the classes are really the same and thus MATH 176 can apply to the MATH 200 rule.

Step 1. Update prior catalogs with the new course number:
In Scribe, locate any instance of MATH 176 and change it to MATH 200 for every catalog – or rely on DAP16 with the UCX-CFG020 DAP13 Process Equivalences flag set.

Step 2. Add an entry to UCX-CFG070 in Controller
Catalog year class was taken = @
Old Course Discipline = MATH
Old Course Number = 176
Student's Catalog Year = @
New Course Discipline = MATH
New Course Number = 200

Translation: "Degree Works will take every instance of MATH 176 and treat it as MATH 200."

UCX-CFG070 Equivalence Course Records

The UCX-CFG070 Equivalence Course Records table contains the equivalence records for mapping historic course keys to current course keys. This table provides an easy to use interface for maintaining the equivalence course records for use in Degree Works. Each record consists of a 30 byte key followed by a number of 12 byte fields for defining the course equivalence. The key can be any alphanumeric value but must be unique for each record. Wildcards can be used in the catalog year and course number fields. This is useful for changing course keys across multiple catalog years or for changing discipline codes globally. In the screen shot below, MLFRATH 100 taken in the 19901991 catalog year became FREN 1000 for students with a catalog year of 20012002.

The Note field is a 50 byte free text field which can be used for internal documentation. This information is not used in audits and is not displayed on any of the audit reports.

You can easily create new equivalence course records using the Controller data entry screen. New UCX-CFG070 records created in Controller will not be available for use in Degree Works until they are loaded into the dap_eqv_crs_mst table in the DAP database. To load new records into the DAP database, you should manually load the records by running the **dapucx2eqv** script. To load the dap_eqv_crs_mst records into the UCX-CFG070 table, use the **dapeqv2ucx** script. This script should only be run once to build the UCX-CFG070 records from the existing dap_eqv_crs_mst table records. You can also use the Bulk Operations function in Controller to load these records from a flat file.

Processing Equivalences into Scribed courses

When course numbers change (HIS 206 was renamed HIST 212) we need to make sure these changes are correctly applied to Degree Works. We have to make sure students taking the old course or the new course get credit for the particular requirement – and we need to make sure that students get the correct advice – which is to take the new course number. Scribers may go through historic blocks and make alter the requirements to list the new course number instead of the old one – but this can be a daunting task if there are a lot of changes.

Degree Works can alter the requirements listing the old course key and change it to the new course key based on the equivalence records that are in place.

To enable this feature you must do the following:

1. Set the UCX-CFG020 DAP13 Process Equivalences flag to Y.
2. Run DAP16 in Transit to reparse all of your blocks

Each time the equivalence table changes you should rerun DAP16 to pick up the changes and apply them to your rules.

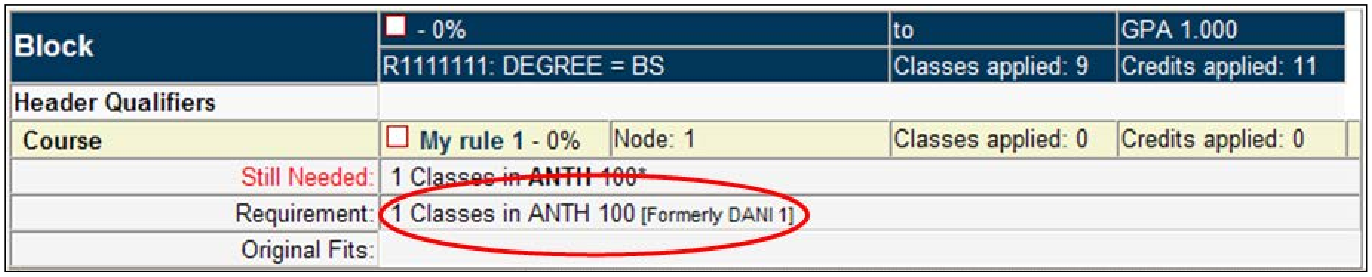
With these dap-eqv-crs-mst records in place in UCX-CFG070:

Catalog Year Taken	Old Course Discipline	Old Number	Student's Catalog Year	New Course Discipline	New Number
2002	DANI	1	@	ANTH	100
2003	DANI	1	@	ANTH	100
2004	DANI	1	@	ANTH	100

With this block in place:

```
      BEGIN
      ;
      1 Class in DANI 1
              Label "My rule 1";
      END.
```

The Diagnostics Report shows the Requirement with the equivalence information. The Requirement line shows what the rule looks like after the conversion has taken place and also the original course that was scribed.



Block	☐ - 0%		to	GPA 1.000
	R1111111: DEGREE = BS		Classes applied: 9	Credits applied: 11
Header Qualifiers				
Course	☐ My rule 1 - 0%	Node: 1	Classes applied: 0	Credits applied: 0
Still Needed:	1 Classes in ~~ANTH 100*~~			
Requirement:	1 Classes in ANTH 100 [Formerly DANI 1]			
Original Fits:				

Note: Although the Diagnostic Audit is showing the equivalence information in the Requirement line our standard Registrars Report worksheet will not. Should you choose to show the information, it is available – a stylesheet change is all that is needed.

Caveats:
Requirements containing ranges or wildcards such as these

```
        5 Credits in MATH 1@
        5 Credits in MATH 100:199
```

will not be processed following these rules. These requirements will have to be taken care of manually at this time.

Simple Scenario 1: Course A was offered from 2002 to 2005, but then in 2006 course A was renamed B.

These UCX-CFG070 records were built:

Catalog Year Taken	Old Course Discipline	Old Number	Student's Catalog Year	New Course Discipline	New Number
2002	A	001	@	B	001
2003	A	001	@	B	001
2004	A	001	@	B	001
2005	A	001	@	B	001

The 2002 scribed block looks like this:

```
      1 Class in A001, X001, Y001
```

Problem:

Under normal operations we that the student has taken A in 2002 and we rename it to B before trying to apply it to rules. When doing this against this block B does not fit since the rule still says A. For some schools it is a lot of work to go through and fix all of their rules to list the current course name.

Solution:

When saving the blocks the parser will make this conversion.

```
      1 Class in ~~A001~~ B001, X001, Y001
```

This change will be made to the saved syntax tree that is then pulled into the auditor.

When auditing a student who took this course when it was named A, the normal processing will take effect and A will be renamed to B and thus will apply to this rule. Students taking the course after 2005 will take it as B and it will apply normally also.

Complex Scenario 2: A changed to B but then B was changed to C

These UCX-CFG070 records were built:

Catalog Year Taken	Old Course Discipline	Old Number	Student’s Catalog Year	New Course Discipline	New Number
2002	A	001	@	B	001
2003	A	001	@	B	001
2004	B	001	@	C	001
2005	B	001	@	C	001

Yes, A was renamed to B but then B was renamed to C.

The 2002, 2003, 2004 and 2005 blocks should all list the rule using C:

```
      1 Class in ~~A001~~ C001, X001, Y001
      1 Class in ~~B001~~ C001, X001, Y001
```

This should be handled as required.

Complex Scenario 3: Reused course key

A (Intro to Art) was renamed to B but then A was reused for some other course (Sculpturing). When this happens, the logic is complicated. We will be looking up the course in the UCX-CFG074 table to find out if the course was reused. If the course was reused the parser will skip the processing of this course - these reused courses will have to be rescribed manually.

Processing Cross-Listings into Scribed courses

There are two options when dealing with scribing of cross-listed courses in Degree Works.

(1) Manually list all of the cross-listed pieces together in rules using Scribe:

```
      1 Class in MATH 101, PHIL 101, STAT 101;
```

(2) Manually list just one piece of a cross-listed set using Scribe rules. Use UCX-CFG073 plus the parser to automatically insert the rest of the cross-listed set inside curly braces {Hide…}:

```
      1 Class in MATH 101 {Hide PHIL 101, STAT 101};
```

To enable the second option perform the following:

(1) Set the UCX-CFG020 DAP13 Process Cross-Listings flag to Y.

(2) Setup UCX-CFG073 – For the example above two records would need to be added:
(1) MATH101 would be added as the UCX Key to UCX-CFG073 with the UCX Value of PHIL101 and
(2) MATH101 would be added as the UCX Key to UCX-CFG073 with the UCX Value of STAT101

Refer to the DGW Technical UCX Documentation for details on setting up UCX-CFG073 records:

For non-Banner clients this table must be loaded manually using Controller or inserted from a file containing records formatted correctly for UCX-CFG073 using the Bulk option of Controller.

For Banner clients this table may be automatically loaded by the Banner Extract EQUIV process if desired or it could be manually loaded. To automatically load UCX-CFG073:

1. Set the UCX-CFG020 BANNER Cross List in SCREQIV = Y.
2. Use Transit to launch RAD38 – the equivalance extract
3. UCX-CFG073 will be generated automatically if any cross listed course records are found in the SCREQIV table.

(3) Use Transit to launch DAP16 to reparse all blocks.

Each time the UCX-CFG073 cross-listings table changes DAP16 should be run to pick up the changes and apply them to your rules.

With these cross-listing in place in UCX-CFG073:

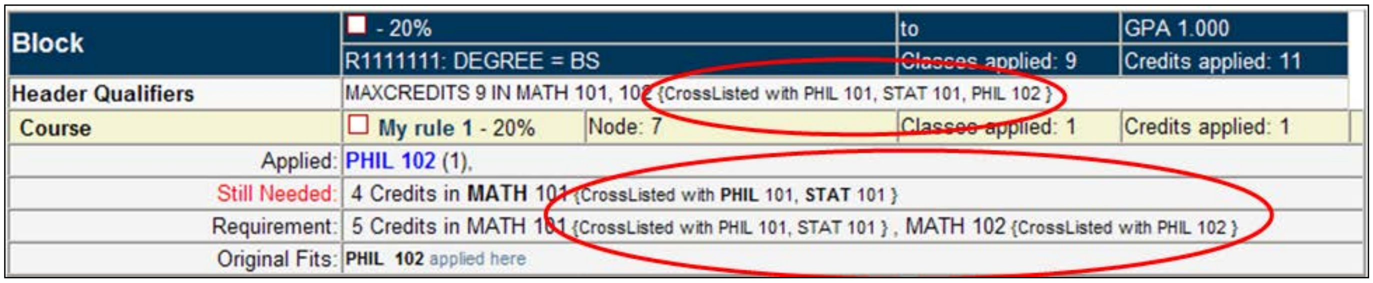
```
MATH 101     cross-listed with   PHIL   101
MATH 101     cross-listed with   STAT   101
MATH 102     cross-listed with   PHIL   102
```

With this block in place:

```
     BEGIN
     MaxCredits 9 in MATH 101, 102
     ;
     5 Credits in MATH 101 , 102
             Label "My rule 1";
     END.
```

The Diagnostics Report displays the cross-listed courses in three different places:

1. The Header Qualifiers line shows the MaxCredits qualifier with the cross-listed information.
2. The advice (Still Needed) line shows the cross-listed information.
3. The Requirement line shows what the real rule looks like – which is different from the Scribed block above.



Block	☐ - 20%		to	GPA 1.000
	R1111111: DEGREE = BS		Classes applied: 9	Credits applied: 11
Header Qualifiers	MAXCREDITS 9 IN MATH 101, 102 {CrossListed with PHIL 101, STAT 101, PHIL 102 }			
Course	☐ **My rule 1** - 20%	Node: 7	Classes applied: 1	Credits applied: 1
Applied:	PHIL 102 (1),			
Still Needed:	4 Credits in **MATH 101** {CrossListed with **PHIL** 101, **STAT** 101 }			
Requirement:	5 Credits in MATH 101 {CrossListed with PHIL 101, STAT 101 } , MATH 102 {CrossListed with PHIL 102 }			
Original Fits:	**PHIL 102** applied here			

Note: Although the Diagnostic Audit is showing the cross-listed information in the advice the standard Student View worksheet will not. However, if you choose to show the cross-listed information it is available – a stylesheet change is all that is needed.

If the same type of changes are to take place whenever PHIL 101 is scribed, UCX-CFG073 records must be created pointing PHIL 101 to MATH 101 and PHIL 101 to STAT 101. The same goes for STAT 101. In all, six records would be needed to cover these three courses in the cross-listing set.

Caveats:
The parser will not be creating header qualifiers to make sure the student will only get credit for the course once in case they are allowed to register for it under the two different names. A header qualifier like this will not be created – this should be handled by the registration system.

```
MaxClasses 1 in MATH 102, PHIL 102
```

These qualifiers may be added manually if required.

Requirements containing ranges or wildcards such as these

```
       5 Credits in MATH 1@
       5 Credits in MATH 100:199
```

will not be processed following these rules. These requirements will have to be taken care of manually at this time.

Financial Aid Audit

A new audit type has been added to Degree Works, so that a Financial Aid audit can be processed in addition to the Academic Audit. Rules for Financial Aid will be built using Scribe. Processing a student's financial aid data against these requirement blocks will verify if a student has met the aid requirements of specific awards, as defined in the rules of the block.

Please note that this Financial Aid audit has nothing to do with the CPoS (Course Program of Study) component of Banner. Nothing mentioned here is needed to get CPoS working.

This process is very similar to that of the Academic Audit. Some special notes pertinent to the Financial aid Audit follow:

AID KEYS:

SDAIDAUD – Allow Financial aid Tab
SDAIDREV – Allow viewing of Financial Aid audits
SDAIDRUN – Allow processing new Financial Aid audits
SDAIDHIS – a picklist of the historic audits for the student will appear
SDAIDDEL – delete historic Financial Aid audits
SDWEB50 – aid audit Worksheet
SDWEB51 – aid audit Worksheet combined with Academic Audit Worksheet

These keys are not assigned to any user class (shp group) – you need to assign these keys in SHPCFG

AUDIT NOTES:

1. In-progress checkbox is always checked as we required the current classes. Preregistered is always unchecked - we never want future classes for an Aid Audit.
2. Aid history obeys the same UCX-CFG020DAP14 History Depth setting when saving audits – if the depth is set to 3 you will end up with 3 degree audits and 3 aid audits.
3. When running a financial aid audit a normal audit is run but any AWARD blocks are also pulled in based on the AWARD values found in the rad_aid_dtl.
4. All rad_aid_dtl records are read when doing an Aid audit - no UCX-SCR002 entries are needed.

SCRIBE NOTES:

1. AWARD block house aid requirements.
2. AWARD header qualifiers must have Labels as we show their advice to the right of the label.
3. AWARD blocks should always contain the current aid year's requirements, and be saved with open-ended catalog years. As long as your AIDAWARD BAN080 item is up-to-date, that block will only ever be given to students who have that award in the *current* aid year, because only those students will have that AWARD code in rad_aid_dtl.

UCX TABLES:

1. UCX-STU016 –Term Type and Financial Aid Year fields need to be populated; only terms with non-blank aid year fields will appear in the web Aid Term picklist.
2. UCX-AUD033 houses AWARD codes. You need to populate UCX-AUD033 to see awards appear in Scribe picklist
3. UCX-BAN080 – used to provide sql statements to extract financial aid information into the database (Banner); OPS clients use the aid.client.properties file in admin/common.

Suggested entries for UCX-BAN080 AIDAWARD:

```
AIDAWARD:AID                  AWARD
AIDAWARD:COLUMN               RPRAWRD_FUND_CODE
AIDAWARD:ORDERBY              RPRAWRD_FUND_CODE
```

```
AIDAWARD:TABLE                 RPRAWRD
AIDAWARD:WHERE_1               RPRAWRD_AIDY_CODE = '0506'
AIDAWARD:WHERE_2                  AND RPRAWRD_AWST_CODE = 'ACPT'
```

Suggested entries for UCX-BAN080 AIDYEAR:

```
AIDYEAR:AID                   AIDYEAR
AIDYEAR:COLUMN                RPRAWRD_AIDY_CODE
AIDYEAR:ORDERBY               RPRAWRD_AIDY_CODE
AIDYEAR:TABLE                 RPRAWRD
AIDYEAR:WHERE_1               RPRAWRD_AWST_CODE = 'ACPT'
AIDYEAR:WHERE_2                  AND RPRAWRD_AIDY_CODE in
AIDYEAR:WHERE_3                  (SELECT b.ROBINST_AIDY_CODE FROM ROBINST b
AIDYEAR:WHERE_4                   WHERE b.ROBINST_STATUS_IND = 'A')
```

Suggested entries for UCX-BAN080 AIDENRSTATUS

```
AIDENRSTATUS:AID               ENROLLSTATUS
AIDENRSTATUS:COLUMN            SGBSTDN_FULL_PART_IND
AIDENRSTATUS:ORDERBY           SGBSTDN_FULL_PART_IND
AIDENRSTATUS:TABLE             SGBSTDN a
AIDENRSTATUS:WHERE_1           a.SGBSTDN_TERM_CODE_EFF =
AIDENRSTATUS:WHERE_2              (SELECT MAX(b.SGBSTDN_TERM_CODE_EFF)
AIDENRSTATUS:WHERE_3            FROM SGBSTDN b
AIDENRSTATUS:WHERE_4            WHERE b.SGBSTDN_PIDM = a.SGBSTDN_PIDM)
```

Suggested entries for UCX-BAN080 AIDSTATUS

```
AIDSTATUS:AID                  AIDSTATUS
AIDSTATUS:COLUMN               RORSAPR_SAPR_CODE
AIDSTATUS:ORDERBY              RORSAPR_SAPR_CODE
AIDSTATUS:TABLE                RORSAPR
AIDSTATUS:WHERE_1              RORSAPR_SAPR_CODE IS NOT NULL
```

Financial Aid Scribe Words

	Alias	**Data Source**
These are only allowed in an IF-statement in an AID audit - usually in an AWARD block:		
CompletedTermType	TermType	
CompletedTermCount	TermCount	
TotalCreditsEarned	TotalCreditsCompleted	rad_term_dtl.rad_cum_tot_earn
ResidenceCreditsEarned	ResidenceCreditsCompleted	rad_term_dtl.rad_cum_cr_earn
TotalCreditsAttempted	CreditsAttempted	rad_term_dtl.rad_cum_gr_att
CreditsAttemptedThisTerm		look at the classes on the aid-term - count the rad_class_dtl.rad_credits (not earned); this does not include any credits for withdrawn classes – unless they are bridged with non-zero credits.
CreditsEarnedThisTerm		look at the classes on the aid-term - count the rad_class_dtl.rad_gpa_credits
CreditsAttemptedThisAidYear		look at the classes on the terms in the aid-year specified - count the rad_class_dtl.rad_credits (not earned); this does not include any credits for withdrawn classes – unless they are bridged with non-zero credits.
CreditsEarnedThisAidYear		look at the classes on the terms in the aid-year specified - count the rad_class_dtl.rad_gpa_credits
CompletedTermCount		count the terms where at least one class was taken either in residence or as a transfer
ResidenceCompletedTermCount		count the terms where at least one class was taken in residence
LastCompletedTermType		else go backwards in time and find the find the first completed term
DegreeCreditsRequired	CreditsRequired	credits required from starting block - only allowed on right-hand-side of an IF-statement

These are allowed in academic and aid audits - though make the most sense in an aid audit in an AWARD block:		
MinCredits 12 in @ (With DWTerm=**Previous**)		previous term the student took classes - prior to active
MinCredits 12 in @ (With DWTerm=**Current**)		active-term
MinTerm X Credits/Classes		look at credits ATTEMPTED - anywhere in the audit - including failed and OTL only works in the starting block or an AWARD block If the MinTerm is in the starting block or an AWARD block we then look at the entire audit - otherwise we just look at classes in this scope.
Under 30 Credits in @ (With Attribute=DEV)	Below	new header qualifier As long as the student has less than or equal to the number of credits/classes specified = OK All attempted classes/credits are applied here
MinGPA 2.0 in (MAJOR) MinGPA 2.0 in (MAJOR = MATH)		These MinGPA scopes can be used in any block - but normally used in an AWARD block "MinGPA 2.0" in AWARD block looks at the overall GPA

Notes:

1. "Previous" and "Current" are special terms now for DW; uppercase or lowercase - both work
2. **Under** and **Below** are new reserved words
3. **OF** is a new reserved word - but can exist as a discipline - only used in an IF-stmt (75% **of**)
4. **%** is now also being used but it should not be added to the reserved punctuation - it can still be used in codes

HEADER examples

```
if (TotalCreditsEarned > 30) then
  beginif
  MinGPA 3.0
   ProxyAdvice "Your CrEarned > 30 - so you must meet the 3.0 requirement"
  endif
```

```
if (LastCompletedTermType = Spring) then
  beginif
  MinGPA 2.5
    ProxyAdvice "You need a GPA of 2.5 now that you have completed Spring"
  endif
```

```
if (CompletedTermCount > 3) then
  beginif
  MinGPA 3.0
    ProxyAdvice "You need a GPA of 3.0 now that you have 3 terms completed"
  Endif

if (ResidenceCompletedTermCount > 3) then
  beginif
  MinGPA 3.2
    ProxyAdvice "You need a GPA of 3.2 now that you have 3 terms completed at
UW"
  Endif
```

```
MinGPA 2.5 in (MAJOR)
  ProxyAdvice "GPA not good enough for Major block"
```

```
Under 30 Credits in @ (With Attribute=DEV)
  ProxyAdvice "You have more than 30 developmental credits"
  Label "No more than 30 developmental credits"
```

```
if (EnrollStatus=F) then # F=full-time
  beginif
  MinTerm 12 Credits
    ProxyAdvice "You did not take at least 12 credits per term"
  endif
else if (EnrollStatus=P) then # P=part-time
  beginelse
  MinTerm 6 Credits
   ProxyAdvice "You did not take at least 6 credits per term"
  endelse
```

RULE examples

```
if (CreditsAttemptedThisTerm > 12) then
  RuleComplete
  Label "12 credits attempted this term - met"
else
  RuleIncomplete
  ProxyAdvice "You have not yet attempted 12 credits"
  Label "12 credits attempted this term - not met";

if (CreditsEarnedThisTerm >= 75% of CreditsAttemptedThisTerm) then
  RuleComplete
  Label "Term credits earned satisfied"
else
  RuleIncomplete
  ProxyAdvice "You have not yet earned 75% of attempted credits"
  Label "Term credits earned- not met";
```

```
if (CreditsAttemptedThisAidYear > 40) then
  RuleComplete
  Label "Aid Year attempted - met"
else
  RuleIncomplete
  ProxyAdvice "You have not yet attempted 40 credits in the aid year"
  Label "Aid Year attempted - not met";
```

```
if (CreditsEarnedThisAidYear >= 75% of CreditsAttemptedThisAidYear) then
  RuleComplete
  Label "Aid year credits earned satisfied"
else
  RuleIncomplete
  ProxyAdvice "You have not yet earned 75% of attempted credits in the aid
year"
  Label "Aid year credits earned- not met";
```

```
if (TotalCreditsAttempted > 30) then
  RuleComplete
  Label "total-cr-attempted - met"
else
  RuleIncomplete
  ProxyAdvice "You have not yet attempted 30 credits"
  Label "total-cr-attempted - not met";
```

```
if (ResidenceCreditsEarned >= 75% of TotalCreditsAttempted ) then
  RuleComplete
  Label "Credits earned is at least 75% of credits att"
else
  RuleIncomplete
  Label "Credits earned is NOT 75% of attempted";
```

```
if (TotalCreditsEarned < 150% of DegreeCreditsRequired) then
  RuleComplete
  Label "Credits earned is less than 150% of required"
else
  RuleIncomplete
  Label "Credits earned more than than 150% of required";
```

```
12 Credits in @ (With DWTerm = Current)  Label "Current term";
```

```
12 Credits in @ (With DWTerm = Previous) Label "Previous term";
```

Data Structures for the Financial Aid Audit

The data structure for holding data required for the financial aid audit is based on the rad_custom_dtl, with one name value pair stored in each record (custom approach). Based on the current known items required for the financial aid audit, the custom approach allows us to add items without making any structural changes to the database. Since all the items needed for an institution's financial aid audit are unknown, this structure seems most appropriate due to its flexibility. The structure is:

RAD_AID_DTL		
Column Name	**Type**	**Size**
srn_id	Integer	
rad_id	Char	10
rad_aid_code	char	12
rad_aid_value	Char	12
unique_id	Integer	
unique_key	Char	36

This version of the data structure will only deal with a single year's data since there is no aid_year column on the table. The concept of current year will effectively be decided by the customer when they select the data being extracted and bridged. At some point, the decision to begin bridging next year data into these structures will be made, with the understanding that multiple year data cannot be mingled without producing erroneous results.

There are a few items, related to awards, which are repeatable, and require special coding for the rad_aid_code. Any record related to an award should have the fund, or award code, as part of the code. For example, to store the cumulative amount of the Pell award, use the rad_aid_code of PellCumAmt.

You should bridge all the active aid years for each student to the rad_aid_dtl using the code of **AIDYEAR**.

You must bridge all active awards for each student to the rad_aid_dtl using the code **AWARD**.

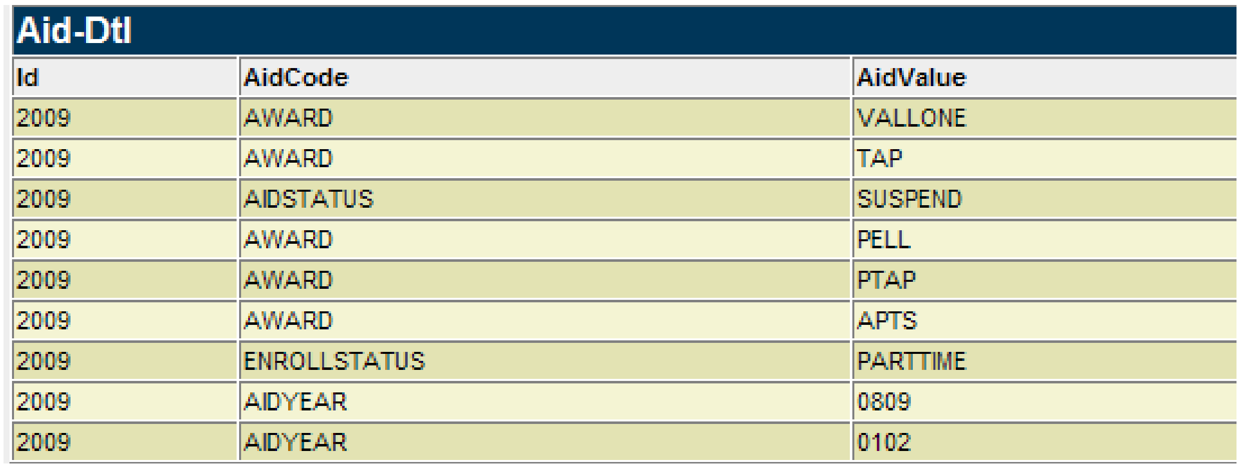
You may also want to bridge an **ENROLLSTATUS** and **AIDSTATUS** for each student for reporting purposes – but neither is required.

Any item that is a number will be stored in ASCII encoded digits, not as binary. Numeric items of the same type (e.g. EstFamContrib) should be zero padded to the same length.

Sample information which might be extracted and bridged from the student system:

	Item	Type	Size	Year Specific	Notes	
01	Year in College	INTEGER		Y	1st, 2nd	
02	Terms Completed	INTEGER		Y	#	
03	Enrollment Status	CHAR	2	Y	Full Time, Part time	
04	Class Level	CHAR	2	Y	FR, SO	
05	Program Type	CHAR	12	Y	5-year, AS, BA, non-degree seeking	
06	SAP Status	CHAR	2	Y	Probation, Suspension	
07	Citizenship	CHAR	2	Y	US, Eligible, Not	
08	TAP Payments	INTEGER		Y	#	
09	TAP Pts Used	INTEGER		Y	#	
10	TAP Waiver Flag	CHAR	1	Y	Y/N	
11	TAP Waiver Term	CHAR	12	Y	Term	
12	TAP Waiver Date	DATE		Y	Date	
13	Award Source	CHAR	12	Y	Pell, ACG, etc	
14	Award Type	CHAR	12	Y	Federal, State, Institution	
15	Cumulative Award Amount	INTEGER		Y	Cumulative, no decimal	
16	Current Award Amount	INTEGER		Y	This year, no decimal	
17	Secondary Program of Study	CHAR	12	N	Rigorous	
18	Previous Undergrad Experience	CHAR	1	Y	Y/N	
19	Pell Recipient	CHAR	1	Y	Y/N	
20	Residence	CHAR	4	Y	State, County, Non	
21	Matriculated	CHAR	1	Y	Y/N	
22	1st time enrollment term	CHAR	12	N	Term	TAP
23	Disability Code	CHAR	4	Y	3C	TAP
24	Estimated Family Contribution	INTEGER		Y	EFC	
25	Education Level	CHAR	4	Y	HS, Some College, BA	
26	Dependency	CHAR	2	Y	Dep / Indep	

The screenshot that follows is of a section from the Student Data Report Worksheet showing the financial aid information extracted and bridged from the student system:



Aid-Dtl

Id	AidCode	AidValue
2009	AWARD	VALLONE
2009	AWARD	TAP
2009	AIDSTATUS	SUSPEND
2009	AWARD	PELL
2009	AWARD	PTAP
2009	AWARD	APTS
2009	ENROLLSTATUS	PARTTIME
2009	AIDYEAR	0809
2009	AIDYEAR	0102

Athletic Eligibility Audit

In addition to Academic and Financial Aid audits, you can run Athletic Eligibility audits. Rules for Athletic Eligibility are built using Scribe. Processing an athlete's athletic data against these requirement blocks will verify if a student is eligible to participate in the sport, in accordance with rules laid out by the NCAA, NJCAA, and other organizations.

This process is very similar to that of the Academic Audit. Some special notes pertinent to the Athletic Eligibility Audit are detailed in the sections that follow.

CreditsAppliedTowardsDegree

The Credits Applied Towards Degree calculation is important for Athletic Eligibility for two reasons:

A) to support the 40/60/80 rule

B) to present an accurate progress bar on the worksheet.

The Credits Applied Towards Degree calculation obviously does not count any credits placed into the Over-The-Limit section (also referred to as Not Counted) but when counting Fall Through (also referred to as Electives) credits, the calculation is more complicated. For most schools, the Gen Ed and Major blocks do not account for all of the credits the student needs to take to graduate. In most cases the student does have to take (non-major) elective credits that appear in the Fall Through section of the audit. However, while some elective credits are required, the student should not take more than that which is required. It is these "excess" elective credits that cannot be counted as Credits Applied Towards Degree.

If credits applied to a block are greater than the credits required, then we use the credits applied. For example, if the major block requires 50 credits but it contains several credit range rules (for example, 6:8 Credits in …) it is possible that the student will actually apply more than 50 credits to the major. These extra credits applied to the major should be subtracted from the elective (fall-through) credits the student must/can take.

If credits are shared between two blocks then those credits are added to the total elective (fall-through) credits the student must/can take. For example, if 7 credits are shared between the major and Gen Ed blocks, the student must then apply 7 additional credits towards the degree – and those additional credits must be taken as elective (fall-through) classes.

In order to correctly calculate how many elective credits are allowed/needed for the degree, each block must have a block header credits qualifier. In some cases, however, you may not want to specify a fixed value – perhaps because of several credit range requirements or because of variable credit classes. In this situation you may use the Pseudo keyword to essentially hide this requirement from the user. For example:

31 Credits **Pseudo** # informational only – not checked by the auditor

The auditor ignores this qualifier when checking to see if the block is complete but will use this credit value when figuring out how many electives are needed to satisfy the overall degree credits. If you have a block that does not represent any credits you can use "0:1 Credits Pseudo".

The credits for a block containing the Optional header qualifier are not included in the calculation

Example 1:

In this situation, the student needs 120 credits to graduate with 98 (50 + 30 + 18) credits coming from required blocks. This means the student must take 22 elective (fall-through) credits in order to meet the 120 required by the degree. However, if the student has more than 22 credits in fall-through the auditor will not count those "excess" credits giving a better measure of how close the student is to reaching the 120 credit goal.

Degree Block:	120 hrs required to graduate
General Education Block:	50 hrs
Major Block:	30 hrs
Conc Block (required by major):	18 hrs

Example 2:

In this situation, the student needs 120 credits to graduate with 100 (50 + 50) credits coming from required blocks; the Minor was added by the student but is not required by the degree or by the major. This means the student must take 20 elective credits in order to meet the 120 required by the degree. However, since the Minor block is not required any classes applying to the Minor are considered as “elective” credits for this calculation – along with any appearing in fall-through. As soon as the student applies more than 20 credits to the Minor and fall-through the extra credits are considered “excess” and are not counted towards degree completion. So if the student has 25 credits applying to the Minor and 10 showing in fall-through only 20 of these 35 credits will be counted.

Degree Block	: 120 hrs required to graduate
General Education Block	: 50 hrs required
Major Block	: 50 hrs required
Minor Block (voluntary)	:30 hrs required

Example 3:

In this situation, the student also needs 120 credits to graduate but 5 credits are being shared between two required blocks.

Degree Block	: 120 hrs required to graduate
General Education Block	: 50 hrs
Major Block	: 30 hrs
Minor Block (required)	: 20 hrs

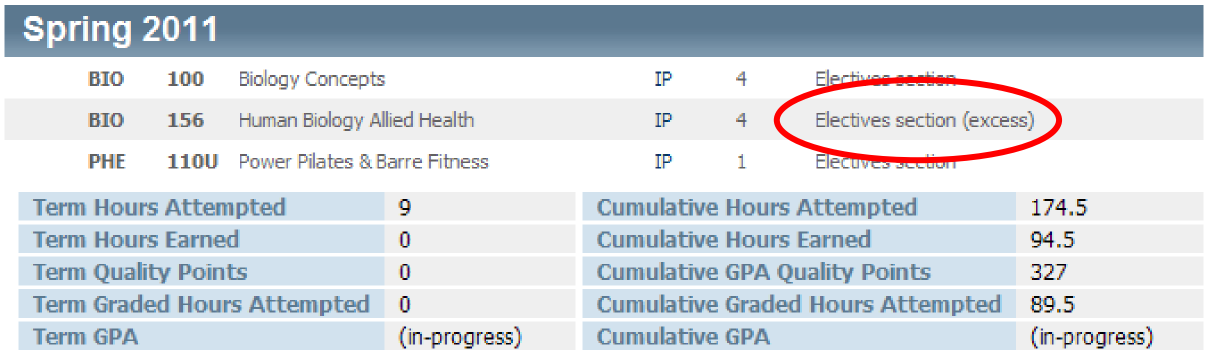
Since 5 credits are shared between major and minor the student must take 25 elective credits instead of only 20.

Example 4:

In this situation, the student needs 120 credits to graduate with 100 (50 + 50) credits coming from required blocks. The Conc is included (called in) by the Major and so the Conc credits are not counted twice. The Major block’s credits required and credits applied values include those credits from the Conc block. For this reason, the Gen Ed and Major block are only included when calculating the credits required. This means that the student must take 20 elective credits in order to meet the 120 required by the degree.

Degree Block	: 120 hrs required to graduate
General Education Block	: 50 hrs required
Major Block	: 50 hrs required
Conc Block (included by major)	: 15 hrs required

In the Class Summary report, any fall-through credits not being counted against degree completion are marked with an “excess” notation.



Spring 2011

BIO	100	Biology Concepts	IP	4	Electives section
BIO	156	Human Biology Allied Health	IP	4	Electives section (excess)
PHE	110U	Power Pilates & Barre Fitness	IP	1	Electives section

Term Hours Attempted	9	Cumulative Hours Attempted	174.5
Term Hours Earned	0	Cumulative Hours Earned	94.5
Term Quality Points	0	Cumulative GPA Quality Points	327
Term Graded Hours Attempted	0	Cumulative Graded Hours Attempted	89.5
Term GPA	(in-progress)	Cumulative GPA	(in-progress)

Completed Term Count

For “completed terms” we only count terms since the first full-time term. So when we are asking if a student is entering the 5th semester, we need to count the completed terms since that first full-time term – but summer terms don’t count. The terms may also be transfer terms or in-resident terms.

Additionally, the first full-time term cannot precede the first official term as bridged to Degree Works on the rad_custom_dtl as AEAFIRSTTERM. For Banner schools this is mapped from the first date of attendance as recorded in SGRATHE. For non-Banner schools this value needs to be the first term you want Degree Works to consider in the term count calculation. The issue here is that high school students who have taken college classes may have transfer work recorded in the system, but the terms for these classes cannot count for athletic eligibility. This AEAFIRSTTERM, or first date of attendance, must represent post-high school terms. For reporting purposes, an AEAFIRSTDATE value must also be bridged to the rad_custom_dtl with a date in the form of ccyymmdd (for example, 20101231 for Dec 31, 2010). This value appears on the worksheet to aid the user in understanding the calculations performed.

Banner schools should ensure you record the student athlete’s first date of attendance at any college/university in the SGRATHE_ATTEND_FROM_DATE field. If you are on Banner 7.x, SGRATHE does not exist. Because of this, Degree Works cannot pull this first date/term of attendance and will have to calculate it using the data it has – starting at the first term it finds which could be data from the student’s high school coursework. However, if you can store the AEAFIRSTTERM in some other place in Banner, you can set up a UCX-BAN080 record to pull it over.

As part of the 6/18/24 hour rule, the full-time term also comes into play – but only with regard to the 6 hour rule. When we look at the previous term attended, we have to go backwards in time until we find the first **full-time** term. For the 18 hour and 24 hour rules, however, the terms may be part-time or full-time terms.

Remedial Credits in First Year

An additional rule is that no more than 6 earned credits can be counted in the first year – and none can be counted after the first year.

Review:

- 6 credits earned in the previous term

- 18 credits earned in the previous two terms – or the previous academic year
- 24 credits earned in the first year

For the 6-credit rule – a check is performed to see if that previous term is in the first year; if it is up to 6 credits remedial are allowed; if it is not in the first year then no remedial credits are allowed.

For the 18-credit rule – a check is performed to see if the previous 2 terms are in the first year; if they are up to 6 credits remedial are allowed; if they are not in the first year then no remedial credits are allowed. Same idea applies when counting the previous year credits.

For the 24-credit rule –up to 6 credits remedial are allowed.

Remedial classes are those with a course number below 100 (eg: MATH 99, ENGL 098) or start with a zero (eg: ART 01224, ANTH 0893).

First Year Earned Credits

A small difference to the calculation for FirstYearEarnedCredits is that the auditor also counts all credits earned prior to the first official term in the first year. This means that transfer credits earned prior to the first year are counted – but the auditor includes credits for all terms (including summer terms) prior to the first term – transfer or not. In addition, summer terms in the first year are also counted. The top of the Diagnostics Report has information that details how this calculation was made for the given student.

Transfer Student-Athletes

The requirements for determining if a transfer student-athlete is eligible to compete at the certifying institution are complex. For example, there are a lot of different rules depending upon whether the student-athlete is a qualifier or non-qualifier through the NCAA and whether they are coming from a 2-year institution or a 4-year institution and these often need to be evaluated on a case by case basis. At this time, the Athletic Eligibility audit does not evaluate whether the transfer student-athlete meets these specific transfer regulations. This verification needs to be done by the institution. However, once the transfer student-athlete has been determined eligible to compete at the certifying institution, an Athletic Eligibility audit can be generated on the student-athlete to certify they meet the general eligibility requirements.

Football Rule

We recommend you place the football rule into its own ATHLETE=FOOTBALL block with an if-statement around your set of rules. If the student is not a football player (that is, does not have FOOTBALL as a sport), the subset will be removed from this block. The auditor recognizes this situation and marks the block as not-needed and the block is hidden from the worksheet display. This means that this FOOTBALL block will be included for all athletes but will only display on the worksheet for true football players.

```
if (SPORT=FOOTBALL) then
  BeginSub
       # Place your requirements here; see sample rules below in this
document
  EndSub
  Label “Football: 9 credits in fall”;
Remark “Football players must earn 9 credits in the prior fall season.";
```

The NCAA states that the student can only regain eligibility one time. Degree Works does not keep track of whether or not the student has already regained eligibility, believing the compliance office on your campus keeps track of this information. Degree Works can tell you if the student has earned 27 credits in the last year, but it is the compliance office which is the final word on whether the student really is allowed to regain eligibility.

Furthermore, the 9 HR/APR E Point status is displayed in the factoids section of the report for students who have SPORT=FOOTBALL on the custom-dtl table. This can be hidden by setting the **vShowFootballStatus** variable in the stylesheet. The status appears as either Passed or Failed. If the student's active term is a fall term then the status is determined based on the completeness of all ATHLETE blocks: if all blocks are complete then the status appears as Passed; otherwise it appears as Failed. If the active term is not a fall term then the status is pulled from the dap_student_mst's dap_aea_status field via the **UCX-SCR002 AEASTATUS** record. You need to create an AEASTATUS record using element number **2407** if you don't have one already. When an audit is run in a fall term the status calculated based on the completeness of the ATHLETE blocks is saved to the dap_student_mst.

In short:

If the audit term is FALL: status is based on completeness of ATHELETE blocks and saved to the database.

If the audit term is not FALL: the status shown is based on what was saved to the database in the previous fall term.

If for some reason no status is found in the database or it is neither PASSED nor FAILED the report will show "(unknown)". This situation will happen for your initial set of audits until you have run audits in a fall term.

ATHLETIC ELIGIBILITY KEYS:

SDATHAUD – Allow Athletic Eligibility tabs
SDATHREV – Allow viewing of Athletic Eligibility audits
SDATHRUN – Allow processing new Athletic Eligibility audits
SDATHHIS – a picklist of the historic audits for the student will appear
SDATHDEL – allow deletion of historic audits
AUDFREEZ – allow freezing of saved audits
AUDDESCR – allow entering/modifying description on a saved audit
SDWEB55 – Athletic Eligibility Worksheet
SDWEB56 – Athletic Eligibility Worksheet combined with Academic Audit Worksheet

These keys assigned to the ATHL user-class (shp group) – you may add or delete keys in SHPCFG

AUDIT NOTES:

1. The In-progress checkbox should always be checked as the current classes are required. The Preregistered checkbox should always be unchecked - the future classes are never wanted for an athletic audit.
2. Athletic Eligibility history follows the same UCX-CFG020DAP14 History Depth setting when saving audits – if the depth is set to 3 you have 3 degree audits and 3 athletic audits. However, you may choose to save an unlimited number of frozen audits.
3. When an Athletic Eligibility audit is run, all ATHLETE blocks are used. The catalog year and other secondary tags are checked.
4. The ATHLETE blocks appear on the worksheet sorted by their block titles. You can alter the block title to get the blocks in the order you want them to appear.
5. For Athletic Audits, the previous term is set to term prior to the active term when the active term is a summer term. When you have chosen to include in-progress classes in the audit the

previous term is set to the active term; otherwise the previous term is the term prior to the active term. Including in-progress classes allows you to forecast for future eligibility.

SCRIBE NOTES:

1. ATHLETE blocks house athletic eligibility requirements.
2. When an Athletic Eligibility audit is run, all ATHLETE blocks are used. You may create a single ATHLETE block or create multiple to separate out the different eligibility rules.
3. The ATHLETE block values are controlled by UCX-AUD031.
4. See the Scribe templates in the *RULE examples* section below to help you quickly and easily create your ATHLETE blocks.

UCX TABLES:

1. **UCX-STU016** – The Term Type field must be set to "SUMMER" for summer terms, “FALL” for fall terms, “SPRING” for spring terms and “WINTER” for winter terms.
2. **UCX-AUD031** houses ATHLETE codes used by Scribe when saving new blocks. You need to populate UCX-AUD031 to see the new codes appear in Scribe.
3. **UCX-STU350:** The Full-time Credits field must be populated for each school
4. **UCX-SCR002**: See the section below title “UCX-SCR002 Setup” on the records needed here.
5. **UCX-BAN080** – Used to provide sql statements to extract athletic information from Banner into Degree Works. If you are a non-Banner and non-Colleague site and use RAD11 to bridge student information, you need to alter your extract to obtain information about your student athletes. You will bridge the data into the rad_custom_dtl. See the “Data Structures” section below for the data that needs to be bridged in.

Suggested entries for UCX-BAN080 ACADSTANDING:

```
BAN080ACADSTANDING:COLUMN           SHRTTRM_ASTD_CODE_END_OF_TERM
BAN080ACADSTANDING:ORDERBY          SHRTTRM_ASTD_CODE_END_OF_TERM
BAN080ACADSTANDING:TABLE            SHRTTRM a
BAN080ACADSTANDING:WHERE_1          a.SHRTTRM_TERM_CODE =
BAN080ACADSTANDING:WHERE_2          (SELECT MAX(b.SHRTTRM_TERM_CODE)
BAN080ACADSTANDING:WHERE_3           FROM SHRTTRM b
BAN080ACADSTANDING:WHERE_4           WHERE b.SHRTTRM_PIDM = a.SHRTTRM_PIDM)
```

Suggested entries for UCX-BAN080 SPORT:

```
BAN080SPORT:COLUMN                  SGRSPRT_ACTC_CODE
BAN080SPORT:ORDERBY                 SGRSPRT_ACTC_CODE
BAN080SPORT:TABLE                   SGRSPRT a
BAN080SPORT:WHERE_1                 a.SGRSPRT_SPST_CODE = 'AC'
```

These ACADSTANDING and SPORT records (along with those built in UCX-SCR002) allow you to Scribe an IF-statement against these codes. To get the Academic Standing description and the Sport description to appear on the report you should create additional UCX-BAN080 entries to pull the corresponding descriptions from the Banner STV tables. The Athletic Eligibility worksheet will look for ACADSTANDESC and SPORTDESC report entries and will display them if found. If they are not found the corresponding ACADSTANDING and SPORT custom entries will be used.

```
BAN080ACADSTANDESC:REPORT           STVASTD
BAN080ACADSTANDESC:COLUMN           SHRTTRM_ASTD_CODE_END_OF_TERM
BAN080ACADSTANDESC:ORDERBY          SHRTTRM_ASTD_CODE_END_OF_TERM
BAN080ACADSTANDESC:TABLE            SHRTTRM a
BAN080ACADSTANDESC:WHERE_1          a.SHRTTRM_TERM_CODE =
BAN080ACADSTANDESC:WHERE_2          (SELECT MAX(b.SHRTTRM_TERM_CODE)
BAN080ACADSTANDESC:WHERE_3           FROM SHRTTRM b
BAN080ACADSTANDESC:WHERE_4           WHERE b.SHRTTRM_PIDM = a.SHRTTRM_PIDM)
```

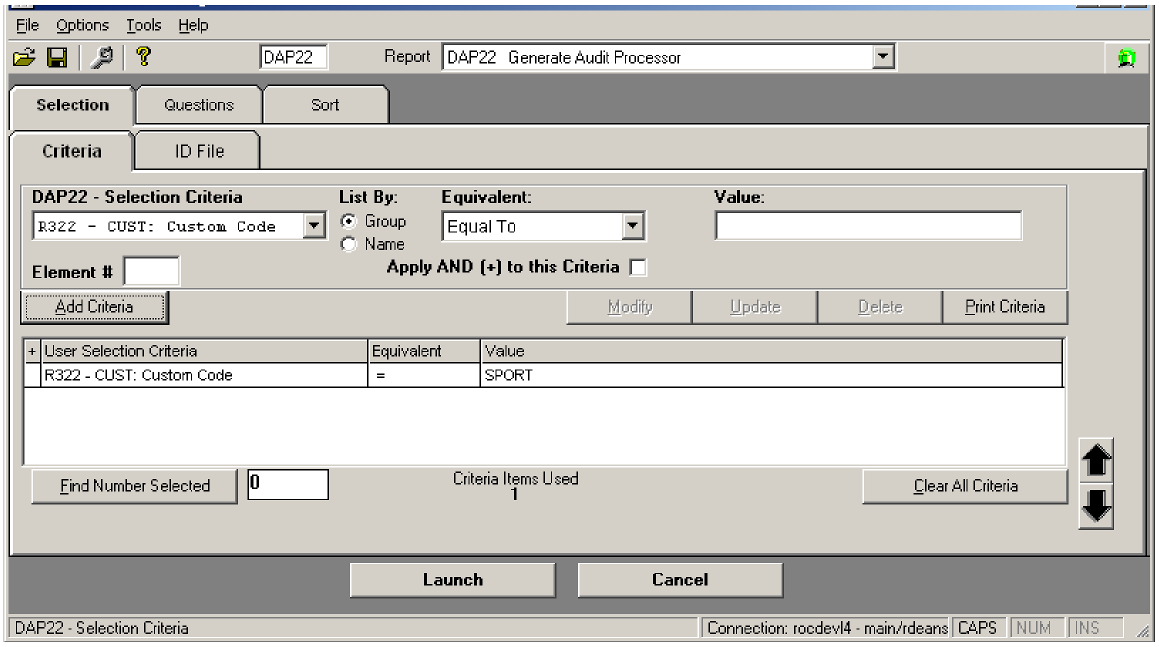
```
BAN080SPORTDESC:REPORT                STVACTC
BAN080SPORTDESC:COLUMN                SGRSPRT_ACTC_CODE
BAN080SPORTDESC:ORDERBY               SGRSPRT_ACTC_CODE
BAN080SPORTDESC:TABLE                 SGRSPRT a
BAN080SPORTDESC:WHERE_1               a.SGRSPRT_SPST_CODE = 'AC'
```

Ensure that you grant access to these tables for the Degree Works Banner-db user.

Batch Audits

You can run batch Athletic Eligibility Audits from DAP22 in Transit like you do with Academic Audits. Choose Athletic Eligibility Audit as the audit type as the first question on the Questions section. You can choose to have them converted to PDF and choose RPT55 – the Athletic Eligibility report. You can also freeze the new audits you run by choosing a freeze type.

On the Selection tab, you can select any student but it makes sense to select your student athletes. This can most easily be done by searching on students with a SPORT record on the rad_custom_dtl – but you may decide to use additional criteria as needed.



DAP22 Report: DAP22 Generate Audit Processor

DAP22 - Selection Criteria: R322 - CUST: Custom Code

List By: [✓ Group] Name

Equivalent: Equal To

Value:

Element #

Apply AND (+) to this Criteria

+	User Selection Criteria	Equivalent	Value
	R322 - CUST: Custom Code	=	SPORT

Find Number Selected: 0

Criteria Items Used 1

DAP22 - Selection Criteria | Connection: rocdevl4 - main/rdeans

Athletic Eligibility Scribe Words

	Alias	Data Source
These are allowed in an IF-statement in an Athletic Eligibility audit - usually in an ATHLETE block:		
CompletedTermCount	TermCount	Includes the current, in-progress term; starting at the first full-time term; either in residence or as a transfer; does not count summer terms
TotalCreditsEarned	TotalCreditsCompleted	rad_term_dtl.rad_cum_tot_earn
ResidenceCreditsEarned	ResidenceCreditsCompleted	rad_term_dtl.rad_cum_cr_earn
TotalCreditsAttempted	CreditsAttempted	rad_term_dtl.rad_cum_gr_att
ResidenceCompletedTermCount		Count the terms in residence – starting at the first full-time term. Does not include summer terms.
DegreeCreditsRequired	CreditsRequired	Credits required from starting block - allowed on right-hand-side of an IF-statement as well as the left-hand side
CreditsAppliedTowardsDegree		Total credits applied towards degree completion. Takes into account the special elective credits allowed/needed discarded the excess fall-through credits.
PreviousTermEarnedCredits		After 4th full-time term these must be degree applicable credits. Previous term must be full-time but can be the in-progress term. Does not include summer terms.
PreviousTermEarnedCredits-Fall		Earned credits for the previous fall term. The Football rule dictates that the student must have earned 9 semester credits in the previous fall term.
Previous2TermsEarnedCredits		After 4th full-time term these must be degree applicable credits. Previous 2 terms can include the current in-progress term – and both can be part-time or full-time terms. Does not include summer terms.
Previous3TermsEarnedCredits		After 4th full-time term these must be degree applicable credits. Previous 3 terms can include the current in-progress term – and all can be part-time or full-time terms. Does not include summer terms.
PreviousAcademicYearEarnedCredits		After 4th full-time term these must be degree applicable credits. Does not include summer terms.
PreviousFullYearEarnedCredits		Includes fall, winter, spring and ending summer term credits. Needed for the Football rule.

FirstYearEarnedCredits		Credits earned in the first full academic year. If first term is summer it is counted. The summer ending term of that year is also counted.

RULE examples

```
#-- 6/18/24 Rule

#-- 6 credits earned in the previous term
if (PreviousTermEarnedCredits >= 6 ) then
  RuleComplete
  Label 6a "Previous Term - at least 6 credits earned required"
else
  RuleIncomplete
  ProxyAdvice "You did not earn 6 credits your previous term"
  Label 6b "Previous Term - at least 6 credits earned required";

#-- 18 credits earned in the previous year

# for Semester schools
if (CompletedTermCount <= 1) then
  RuleComplete
  Label 18a "2 terms have not yet been completed"
else if (Previous2TermsEarnedCredits       >= 18 or
         PreviousAcademicYearEarnedcredits >= 18) then
  RuleComplete
  Label 18b "Previous Year: at least 18 credits earned required"
else
  RuleIncomplete
  ProxyAdvice "You did not earn 18 credits your previous year"
  Label 18c "Previous Year: at least 18 credits earned required";

# for Quarter schools
if (CompletedTermCount <= 2) then
  RuleComplete
  Label 18e "3 terms have not yet been completed"
Else if (Previous3TermsEarnedCredits       >= 27 or
         PreviousAcademicYearEarnedcredits >= 27) then
  RuleComplete
  Label 18f "Previous Year: at least 27 credits earned required"
else
  RuleIncomplete
  ProxyAdvice "You did not earn 27 credits your previous year"
  Label 18g "Previous Year: at least 27 credits earned required";
```

```
#-- 24 credits earned in the first year
if (FirstYearEarnedCredits >= 24 ) then
  RuleComplete
  Label 24a "First Year: at least 24 credits earned required"
else
  RuleIncomplete
  ProxyAdvice "You did not earn at least 18 credits your First Year"
  Label 24b "First Year: at least 24 credits earned required";
```

```
#-- 40/60/80 Rule

#-- 80% complete with course requirements by start of 5th year
if (CompletedTermCount >= 8 And
    CreditsAppliedTowardsDegree >= 80% of DegreeCreditsRequired) then
  RuleComplete
  Label 80a "80% complete by start of 5th year"
else if (CompletedTermCount >= 8) then
  RuleIncomplete
  ProxyAdvice "You did not complete 80% by the start of the 5th year"
  Label 80b "80% complete by start of 5th year"

#-- 60% complete with course requirements by start of 4th year
Else if (CompletedTermCount >= 6 And
    CreditsAppliedTowardsDegree >= 60% of DegreeCreditsRequired) then
  RuleComplete
  Label 60a "60% complete by start of 4th year"
else if (CompletedTermCount >= 6) then
  RuleIncomplete
  ProxyAdvice "You did not complete 60% by the start of the 4th year"
  Label 60b "60% complete by start of 4th year"

#-- 40% complete with course requirements by start of 3rd year
Else if (CompletedTermCount >= 4 And
    CreditsAppliedTowardsDegree >= 40% of DegreeCreditsRequired) then
  RuleComplete
  Label 40a "40% complete by start of 3rd year"
else if (CompletedTermCount >= 4) then
  RuleIncomplete
  ProxyAdvice "You did not complete 40% by the start of the 3rd year"
  Label 40b "40% complete by start of 3rd year"
else # not yet started 3rd year
  RuleIncomplete
  ProxyAdvice "By the start of your 3rd year you need to be 40% complete"
  Label 3rd "3rd year not started ";
```

```
# 90/95/100 GPA Rule
```

```
#-- Entering 2nd year - GPA must be at least 90% of 2.0
If (CompletedTermCount <= 3 and SISGPA >= 1.8) then
  RuleComplete
  Label GPA-1 "Entering 2nd year - GPA must be at least 1.8"
Else If (CompletedTermCount = 3 and SISGPA < 1.8) then
  RuleIncomplete
  ProxyAdvice "Your GPA is less than 1.8 - you are ineligible"
  Label GPA-2 "Entering 2nd year - GPA must be at least 1.8"
Else If (CompletedTermCount < 3 and SISGPA < 1.8) then
  RuleIncomplete
  ProxyAdvice "Your GPA needs to be at least 1.8 by the start "
  ProxyAdvice "of your 3rd term."
  Label GPA-3 "Entering 2nd year - GPA must be at least 1.8"
```

```
#-- Entering 3rd year - GPA must be at least 95% of 2.0
Else If (CompletedTermCount <= 5 and SISGPA >= 1.9) then
  RuleComplete
  Label GPA-4 "Entering 3rd year - GPA must be at least 1.9"
Else If (CompletedTermCount <= 5 and SISGPA < 1.9) then
  RuleIncomplete
  ProxyAdvice "Your GPA is less than 1.9 - you are ineligible"
  Label GPA-5 "Entering 3rd year - GPA must be at least 1.9"

#-- Entering 4th year - GPA must be at least 2.0
#-- And beyond 4th year also
Else If (CompletedTermCount >= 6 and SISGPA >= 2.0) then
  RuleComplete
  Label GPA-6 "4th year and beyond - GPA must be at least 2.0"
Else If (CompletedTermCount >= 6 and SISGPA < 2.0) then
  RuleIncomplete
  ProxyAdvice "Your GPA is less than 2.0 - you are ineligible"
  Label GPA-7 "4th year and beyond - GPA must be at least 2.0";
```

```
#-- Before 3rd year - degree/major must be declared
If (NumMajors >= 1) then  # regardless of how many terms have been completed
  RuleComplete
  Label Major1 "Declare a degree/major by 3rd year"
else if (CompletedTermCount < 5) then
  RuleIncomplete
  ProxyAdvice "You need to declare a degree/major by your Third Year"
  Label Major2 "Declare a degree/major by 3rd year"
else # student has already started her 3rd year
  RuleIncomplete
  ProxyAdvice "You did not declare a degree/major by your Third Year"
  Label Major3 "Declare a degree/major by 3rd year";
```

```
# Student must be in good academic standing
if (AcadStanding = "GOOD") then
  RuleComplete
  Label 1 "You must be in good academic standing"
else # not so good
  RuleIncomplete
  Proxyadvice "You are currently not in good academic standing"
  Label 2 "You must be in good academic standing";
```

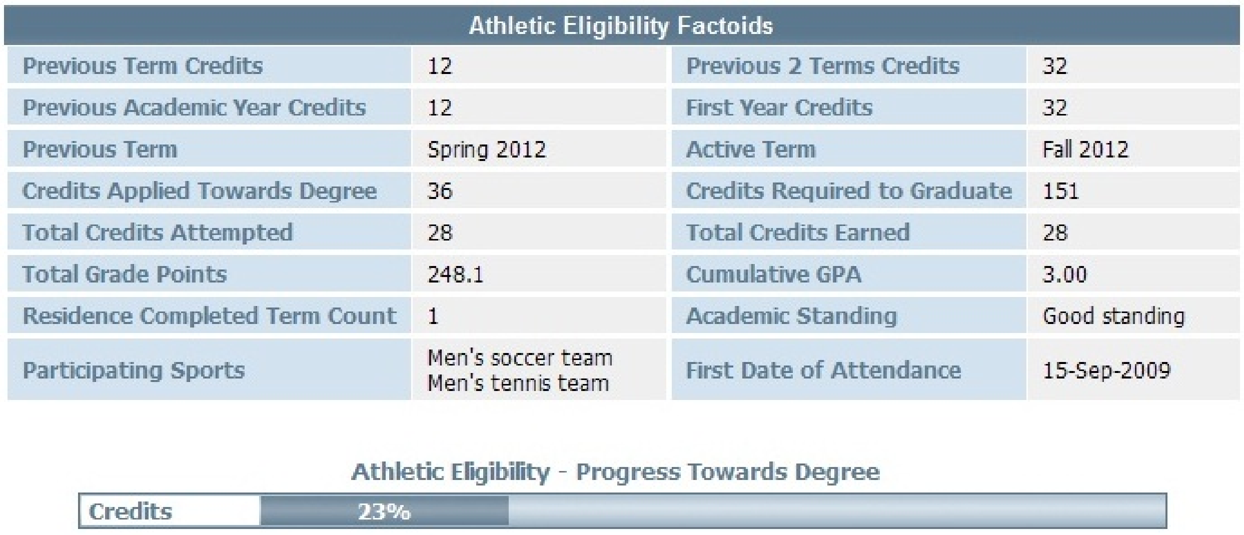
```
############# Football rule - 9 credits in prior fall season
Begin
;
Remark "Football players must earn 9 credits in the prior fall season.";
Remark "You can regain eligibility if you earn 27 credits before the";
Remark "next fall season (includes summer term credits). ";

# If the student does not have FOOTBALL as a SPORT this block will be
# hidden from the worksheet.

if (SPORT=FOOTBALL) then
  BeginSub
   if (PreviousTermEarnedCredits-Fall < 9) then
     RuleIncomplete
     ProxyAdvice "You did not earn at least 9 credits in your last fall term"
     Label "Earned less than 9 credits last fall"

   else # if (PreviousTermEarnedCredits-Fall >= 9) then
     RuleComplete
     Label "Earned at least 9 credits last fall";

  # ResidenceCompletedTermCount does not count summer terms
  # FirstYearEarnedCredits includes prior and ending summer terms
  # PreviousFullYearEarnedCredits includes ending summer
  if (PreviousTermEarnedCredits-Fall < 9) then
    # If a 1st year student and not enough credits
    if (ResidenceCompletedTermCount < 3 and
        FirstYearEarnedCredits < 27) then
      RuleIncomplete
      ProxyAdvice "You earned less than 27 credits and thus did not"
      ProxyAdvice "regain eligibility"
      Label "1st year: less than 27 credits"
    # If a 1st year student and has enough credits
    else If (ResidenceCompletedTermCount < 3 and
             FirstYearEarnedCredits >= 27) then
      RuleComplete
      Label "1st year: at least 27 credits; regained eligibility"
    # If past 1st year and not enough credits
    else If (ResidenceCompletedTermCount >= 3 and
             PreviousFullYearEarnedCredits < 27) then
      RuleIncomplete
      ProxyAdvice "You earned less than 27 credits and thus did not"
      ProxyAdvice "regain eligibility"
      Label "Less than 27 credits this past year"
    # If past 1st year and has enough credits
    else If (ResidenceCompletedTermCount >= 3 and
             PreviousFullYearEarnedCredits >= 27) then
      RuleComplete
      Label "27 credits in the last year; regained eligibility"

    else # just in case
      RuleIncomplete
      ProxyAdvice "Unexpected situation; need to review 27 credit requirement"
      Label "27 credits check; should never see this";

  EndSub
    Label "Football: 9 credits in fall";

# Quarter schools should change  9 to  8 above
# Quarter schools should change 27 to 40 above
# Quarter schools may also want to change the term count

End.

# These may also be useful...
# if (DegreeCreditsRequired >= 120) then
# if (CreditsAppliedTowardsDegree >= 48 ) then
```

Data Structures for the Athletic Eligibility Audit

You need to bridge each student's Academic Standing description to the rad_REPORT_dtl with a code of **ACADSTANDESC and the Academic Standing value to the rad_CUSTOM_dtl with a code of ACADSTANDING**. The description will appear in the header of the worksheet while the value can be used in any IF-statement that you may have scribed. If ACADSTANDESC description is not found, the worksheet will use the ACADSTANDING code on the rad_CUSTOM_dtl.

You also need to bridge each student's participating Sport description to the rad_REPORT_dtl with a code of **SPORTDESC and the Sport value to the rad_CUSTOM_dtl with a code of SPORT**. The description will appear in the header of the worksheet while the value can be used in any IF-statement that you may have scribed. If the student is participating in more than one sport activity then bridge each sport as a different rad_custom_dtl and rad_report_dtl record. If SPORTDESC description is not found, the worksheet will use the SPORT code on the rad_CUSTOM_dtl.

The Athletic Eligibility worksheet has a special section devoted to specific, significant data related to athletic eligibility. Below the special section a progress bar appears. This progress bar is directly related to the calculations performed for the 40-60-80 rule and accounts for elective credits allowed/needed. The data feeding this progress bar is specifically calculated for athletic eligibility and therefore differs from the data feeding the progress bar on the academic audit worksheet.



Athletic Eligibility Factoids			
Previous Term Credits	12	Previous 2 Terms Credits	32
Previous Academic Year Credits	12	First Year Credits	32
Previous Term	Spring 2012	Active Term	Fall 2012
Credits Applied Towards Degree	36	Credits Required to Graduate	151
Total Credits Attempted	28	Total Credits Earned	28
Total Grade Points	248.1	Cumulative GPA	3.00
Residence Completed Term Count	1	Academic Standing	Good standing
Participating Sports	Men's soccer team Men's tennis team	First Date of Attendance	15-Sep-2009

Athletic Eligibility - Progress Towards Degree

Credits 23%

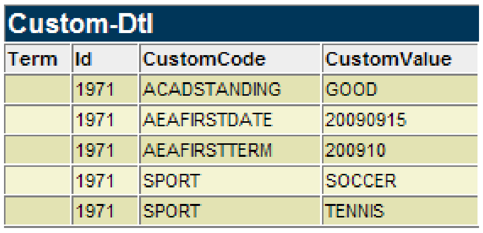
At least four values must be bridged to the rad**_custom**_dtl.

ACADSTANDING – indicates if the student is in good academic standing

AEAFIRSTTERM – first official term that can be counted; mapped from first date of attendance; this date is stored on SGRATHE for Banner schools

AEAFIRSTDATE – first official date of attendance; this date is stored on SGRATHE for Banner schools

SPORT – one or more participating sports; this value is stored on SGRSPRT for Banner schools Multiple SPORT values can be bridged if the student is participating in multiple sports.



Custom-Dtl

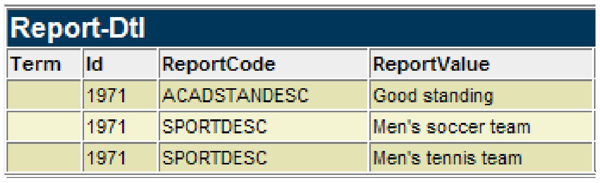
Term	Id	CustomCode	CustomValue
	1971	ACADSTANDING	GOOD
	1971	AEAFIRSTDATE	20090915
	1971	AEAFIRSTTERM	200910
	1971	SPORT	SOCCER
	1971	SPORT	TENNIS

Two values should be bridged to the rad_**report**_dtl. If they are not found the corresponding values from the rad_custom_dtl will be used for the worksheet display.

ACADSTANDESC – description - indicates if the student is in good academic standing

SPORTDESC – one or more participating sport descriptions; this value is stored on SGRSPRT for Banner schools

Multiple SPORTDESC values can be bridged if the student is participating in multiple sports.



Report-Dtl

Term	Id	ReportCode	ReportValue
	1971	ACADSTANDESC	Good standing
	1971	SPORTDESC	Men's soccer team
	1971	SPORTDESC	Men's tennis team

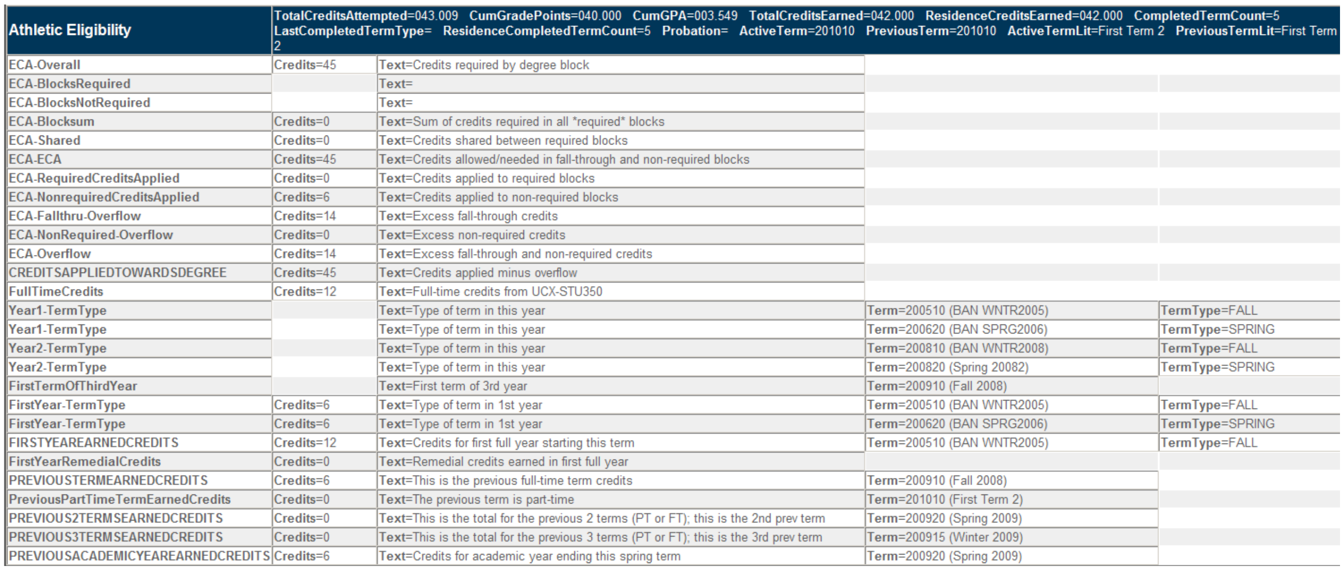
The top of the Diagnostics Report can be very helpful in understanding why rules in your ATHLETE blocks were marked as complete or as unsatisfied. The “type” of data is listed on the far left of this grid. Those types in all UPPERCASE letters indicate those that are directly related to a Scribe reserved word you may use in an IF-statement. Those in MixedCase are used in internal calculations and appear here merely to assist you in understanding the data.

Credits – this field contains the credits value for this type

Text – describes the data for this type

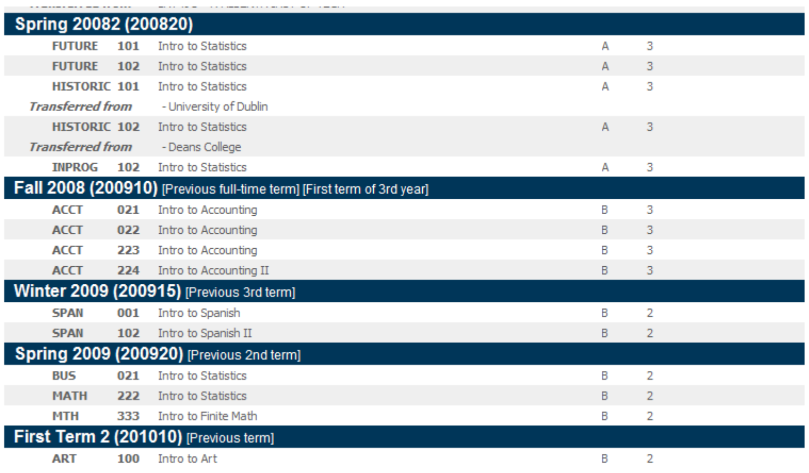
Term – the term value used in this calculation

TermType – the type of term as specified on UCX-STU016



Athletic Eligibility	TotalCreditsAttempted=043.009 CumGradePoints=040.000 CumGPA=003.549 TotalCreditsEarned=042.000 ResidenceCreditsEarned=042.000 CompletedTermCount=5 LastCompletedTermType= ResidenceCompletedTermCount=5 Probation= ActiveTerm=201010 PreviousTerm=201010 ActiveTermLit=First Term 2 PreviousTermLit=First Term 2			
ECA-Overall	Credits=45	Text=Credits required by degree block		
ECA-BlocksRequired		Text=		
ECA-BlocksNotRequired		Text=		
ECA-Blocksum	Credits=0	Text=Sum of credits required in all *required* blocks		
ECA-Shared	Credits=0	Text=Credits shared between required blocks		
ECA-ECA	Credits=45	Text=Credits allowed/needed in fall-through and non-required blocks		
ECA-RequiredCreditsApplied	Credits=0	Text=Credits applied to required blocks		
ECA-NonrequiredCreditsApplied	Credits=6	Text=Credits applied to non-required blocks		
ECA-Fallthru-Overflow	Credits=14	Text=Excess fall-through credits		
ECA-NonRequired-Overflow	Credits=0	Text=Excess non-required credits		
ECA-Overflow	Credits=14	Text=Excess fall-through and non-required credits		
CREDITSAPPLIEDTOWARDSDEGREE	Credits=45	Text=Credits applied minus overflow		
FullTimeCredits	Credits=12	Text=Full-time credits from UCX-STU350		
Year1-TermType		Text=Type of term in this year	Term=200510 (BAN WNTR2005)	TermType=FALL
Year1-TermType		Text=Type of term in this year	Term=200620 (BAN SPRG2006)	TermType=SPRING
Year2-TermType		Text=Type of term in this year	Term=200810 (BAN WNTR2008)	TermType=FALL
Year2-TermType		Text=Type of term in this year	Term=200820 (Spring 20082)	TermType=SPRING
FirstTermOfThirdYear		Text=First term of 3rd year	Term=200910 (Fall 2008)	
FirstYear-TermType	Credits=6	Text=Type of term in 1st year	Term=200510 (BAN WNTR2005)	TermType=FALL
FirstYear-TermType	Credits=6	Text=Type of term in 1st year	Term=200620 (BAN SPRG2006)	TermType=SPRING
FIRSTYEAREARNEDCREDITS	Credits=12	Text=Credits for first full year starting this term	Term=200510 (BAN WNTR2005)	TermType=FALL
FirstYearRemedialCredits	Credits=0	Text=Remedial credits earned in first full year		
PREVIOUSTERMEARNEDCREDITS	Credits=6	Text=This is the previous full-time term credits	Term=200910 (Fall 2008)	
PreviousPartTimeTermEarnedCredits	Credits=0	Text=The previous term is part-time	Term=201010 (First Term 2)	
PREVIOUS2TERMSEARNEDCREDITS	Credits=0	Text=This is the total for the previous 2 terms (PT or FT); this is the 2nd prev term	Term=200920 (Spring 2009)	
PREVIOUS3TERMSEARNEDCREDITS	Credits=0	Text=This is the total for the previous 3 terms (PT or FT); this is the 3rd prev term	Term=200915 (Winter 2009)	
PREVIOUSACADEMICYEAREARNEDCREDITS	Credits=6	Text=Credits for academic year ending this spring term	Term=200920 (Spring 2009)	

In addition to the data above, the Diagnostics Report for athletic audits also contains a summary of classes by term. When applicable, a notation next to each term appears to connect the information above to a specific term.



Spring 20082 (200820)

FUTURE	101	Intro to Statistics	A	3
FUTURE	102	Intro to Statistics	A	3
HISTORIC	101	Intro to Statistics	A	3
Transferred from		- University of Dublin		
HISTORIC	102	Intro to Statistics	A	3
Transferred from		- Deans College		
INPROG	102	Intro to Statistics	A	3

Fall 2008 (200910) [Previous full-time term] [First term of 3rd year]

ACCT	021	Intro to Accounting	B	3
ACCT	022	Intro to Accounting	B	3
ACCT	223	Intro to Accounting	B	3
ACCT	224	Intro to Accounting II	B	3

Winter 2009 (200915) [Previous 3rd term]

SPAN	001	Intro to Spanish	B	2
SPAN	102	Intro to Spanish II	B	2

Spring 2009 (200920) [Previous 2nd term]

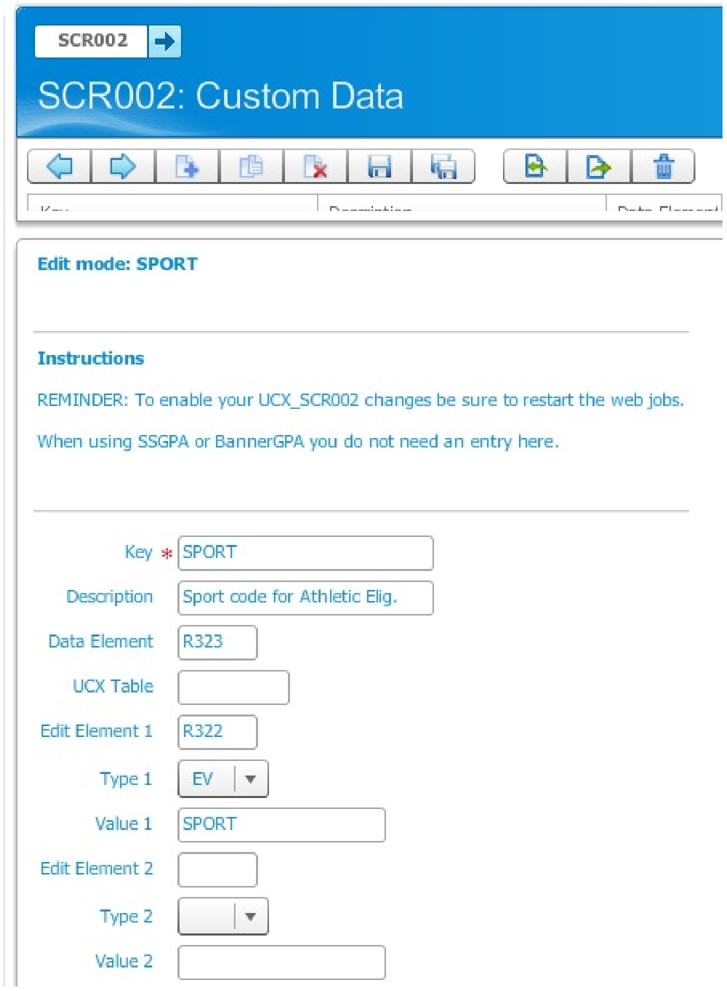
BUS	021	Intro to Statistics	B	2
MATH	222	Intro to Statistics	B	2
MTH	333	Intro to Finite Math	B	2

First Term 2 (201010) [Previous term]

ART	100	Intro to Art	B	2

UCX-SCR002 Setup

Create a **SPORT** record which looks similar to the one shown in the screenshot that follows. The SPORT record is needed if you will be scribing against the sport code and is also needed to gather the data for display on the worksheet.



Edit mode: SPORT

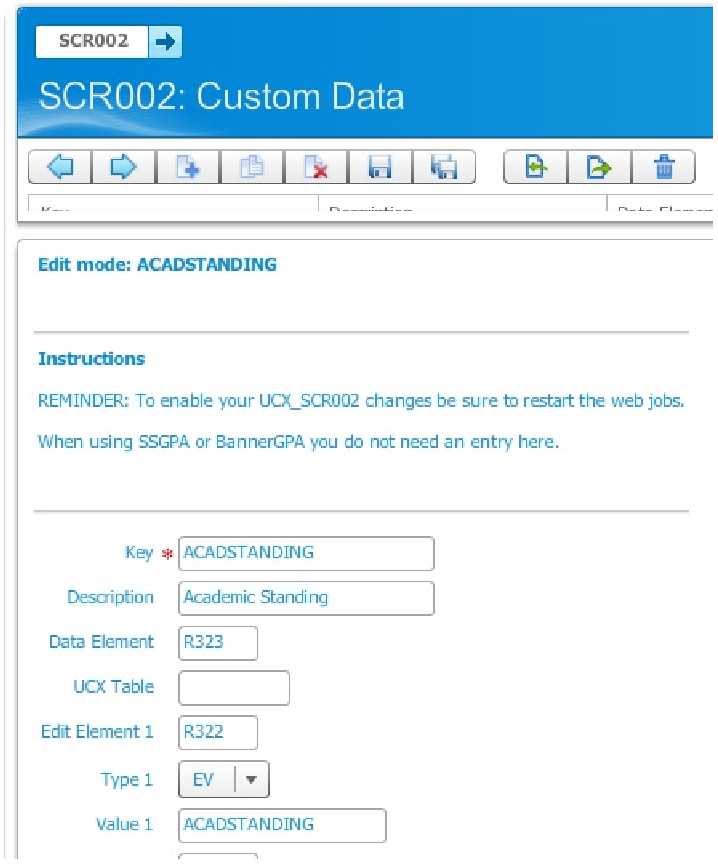
Instructions

REMINDER: To enable your UCX_SCR002 changes be sure to restart the web jobs.

When using SSGPA or BannerGPA you do not need an entry here.

Key *	SPORT
Description	Sport code for Athletic Elig.
Data Element	R323
UCX Table	
Edit Element 1	R322
Type 1	EV
Value 1	SPORT
Edit Element 2	
Type 2	
Value 2	

Create an **ACADSTANDING** record which looks similar to the one shown in the screenshot that follows. You most likely will be scribing against the academic standing and so you will probably need this record. This too is displayed on the worksheet (Ensure the Value field has the ending “G” – which is hidden below.)



Edit mode: ACADSTANDING

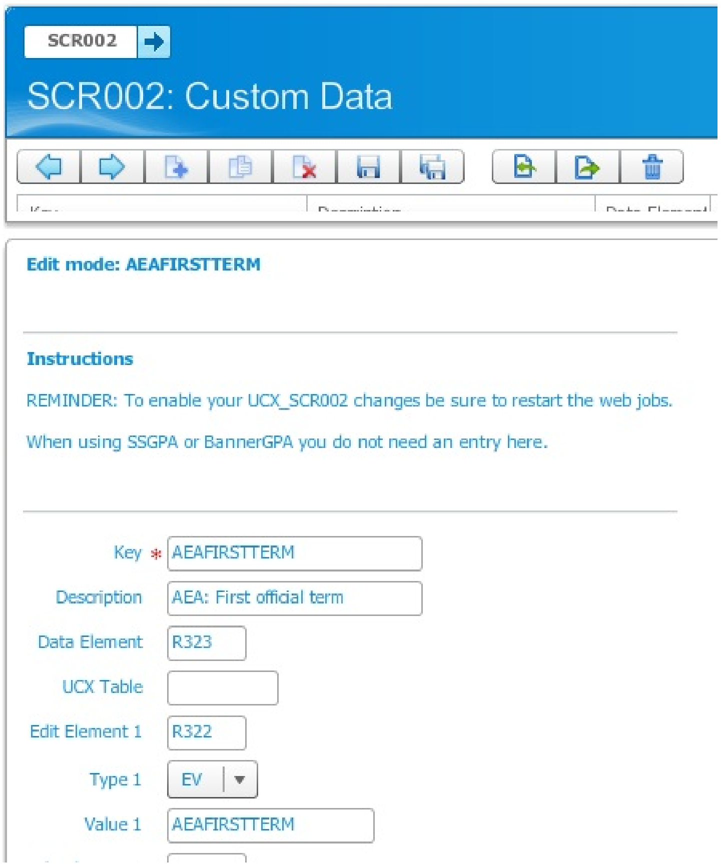
Instructions

REMINDER: To enable your UCX_SCR002 changes be sure to restart the web jobs.

When using SSGPA or BannerGPA you do not need an entry here.

Key *	ACADSTANDING
Description	Academic Standing
Data Element	R323
UCX Table	
Edit Element 1	R322
Type 1	EV
Value 1	ACADSTANDING

Create an **AEAFIRSTTERM** record which looks similar to the one shown in the screenshot that follows. You most likely will not be scribing against this value but this record is still needed so that the value can be passed to the auditor to be uses when calculating the term count. (Ensure the Value field has the ending “M” – which is hidden below.)



Edit mode: AEAFIRSTTERM

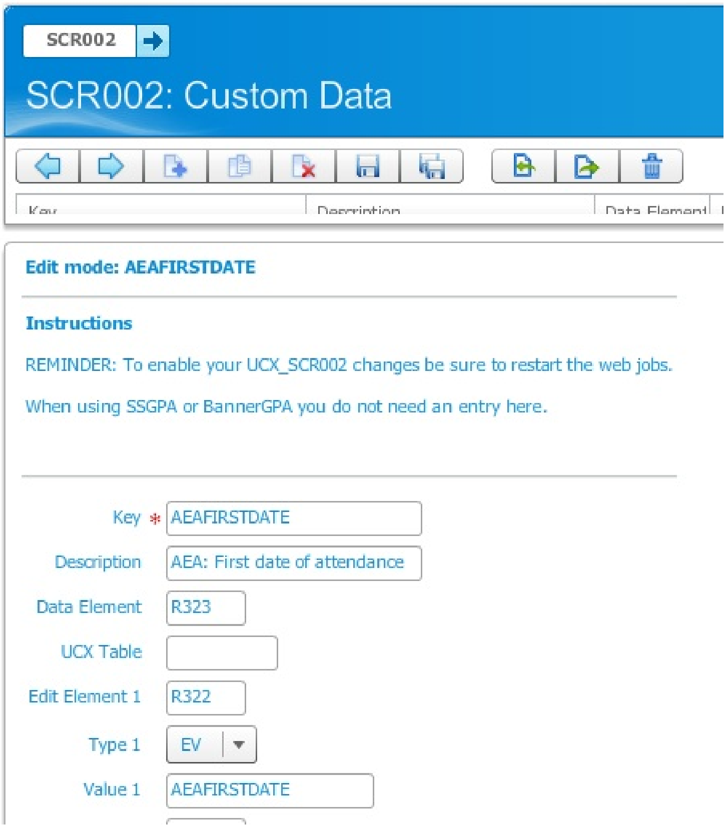
Instructions

REMINDER: To enable your UCX_SCR002 changes be sure to restart the web jobs.

When using SSGPA or BannerGPA you do not need an entry here.

Key *	AEAFIRSTTERM
Description	AEA: First official term
Data Element	R323
UCX Table	
Edit Element 1	R322
Type 1	EV
Value 1	AEAFIRSTTERM

Create an **AEAFIRSTDATE** record which looks similar to the one shown in the screenshot that follows. You most likely will not be scribing against this value but this record is still needed so that the value can be passed to the worksheet for display. (Ensure the Value field has the ending “E” – which is hidden below.)



Edit mode: AEAFIRSTDATE

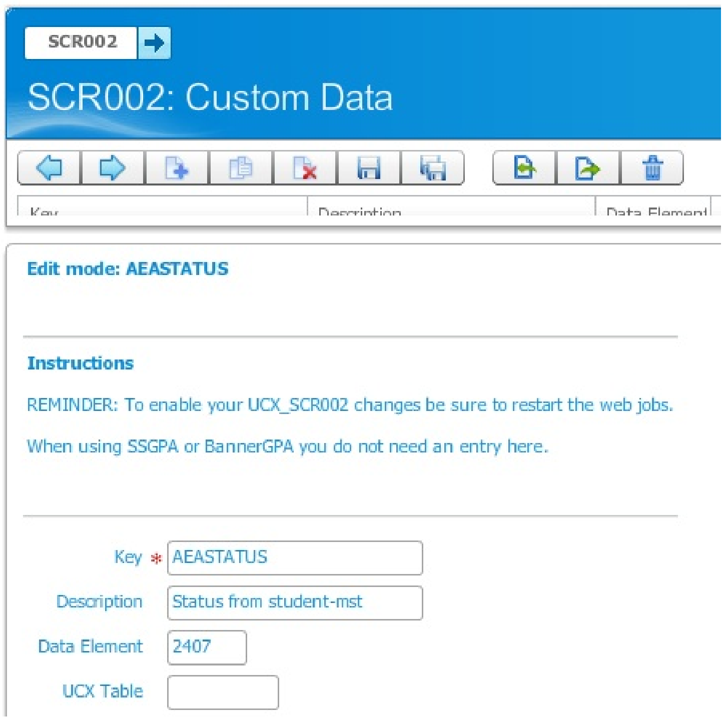
Instructions

REMINDER: To enable your UCX_SCR002 changes be sure to restart the web jobs.

When using SSGPA or BannerGPA you do not need an entry here.

Key *	AEAFIRSTDATE
Description	AEA: First date of attendance
Data Element	R323
UCX Table	
Edit Element 1	R322
Type 1	EV
Value 1	AEAFIRSTDATE

Create an **AEASTATUS** record which looks similar to the one shown in the screenshot that follows. You most likely will not be scribing against this value but this record is still needed so that the value can be passed to the worksheet for display. If you do not want to show the Football 9 HR/APR E Point value on the report you do not need to add this record.



Edit mode: AEASTATUS

Instructions

REMINDER: To enable your UCX_SCR002 changes be sure to restart the web jobs.

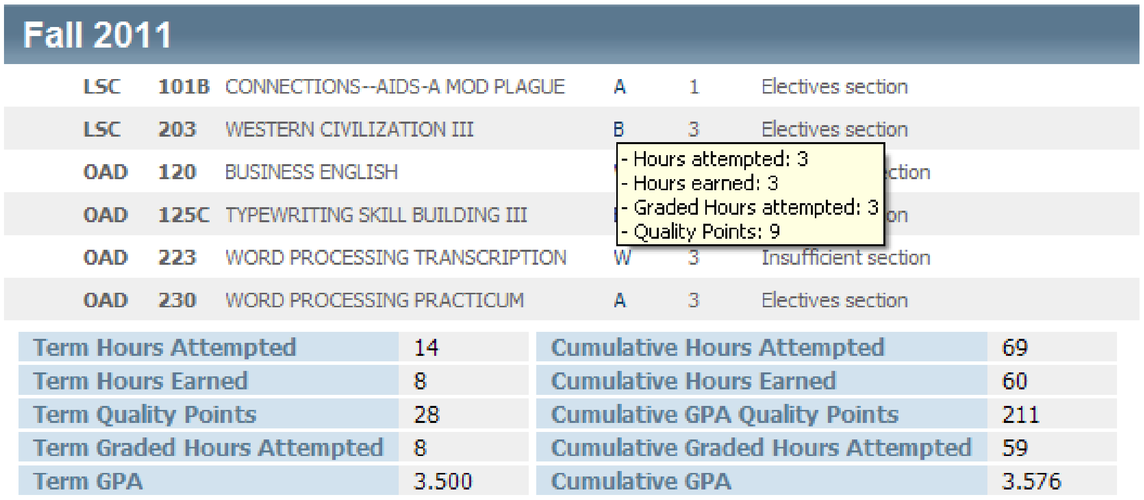
When using SSGPA or BannerGPA you do not need an entry here.

Key *	AEASTATUS
Description	Status from student-mst
Data Element	2407
UCX Table	

Class Summary report

At the top of the Athletic Eligibility page there is a “Class Summary” link. This report is like the Class History report on the worksheets tab but it gives you summary information for each term. The term information appears on the left and the cumulative information appears on the right. In addition, you can place your mouse cursor over the grade for each class to see a hint showing useful details about the class.

This information is the same data which appears on the Diagnostics Report. The credit value appearing for each class is normally the credits earned except for the cases of incomplete/in-progress classes – this value is then the credits attempted.



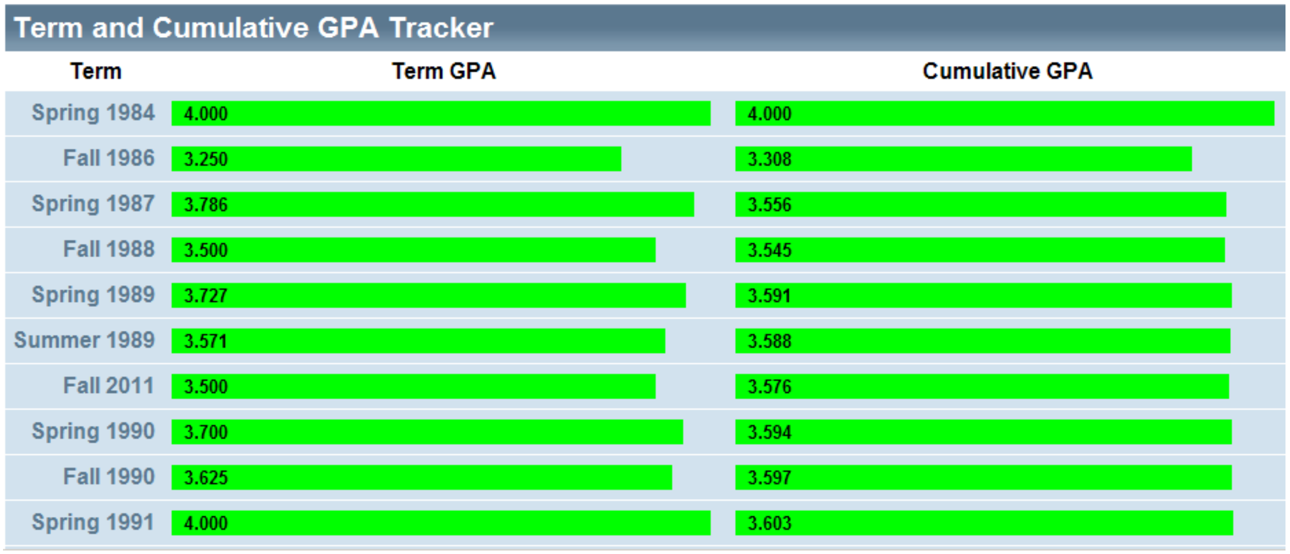
Fall 2011

LSC	101B	CONNECTIONS--AIDS-A MOD PLAGUE	A	1	Electives section
LSC	203	WESTERN CIVILIZATION III	B	3	Electives section
OAD	120	BUSINESS ENGLISH			ction
OAD	125C	TYPEWRITING SKILL BUILDING III			on
OAD	223	WORD PROCESSING TRANSCRIPTION	W	3	Insufficient section
OAD	230	WORD PROCESSING PRACTICUM	A	3	Electives section

- Hours attempted: 3
- Hours earned: 3
- Graded Hours attempted: 3
- Quality Points: 9

Term Hours Attempted	14	Cumulative Hours Attempted	69
Term Hours Earned	8	Cumulative Hours Earned	60
Term Quality Points	28	Cumulative GPA Quality Points	211
Term Graded Hours Attempted	8	Cumulative Graded Hours Attempted	59
Term GPA	3.500	Cumulative GPA	3.576

The bottom of the Class Summary report shows a GPA Tracker. This section gives a great view of how the student’s term and cumulative GPA has changed over time.



Term and Cumulative GPA Tracker

Term	Term GPA	Cumulative GPA
Spring 1984	4.000	4.000
Fall 1986	3.250	3.308
Spring 1987	3.786	3.556
Fall 1988	3.500	3.545
Spring 1989	3.727	3.591
Summer 1989	3.571	3.588
Fall 2011	3.500	3.576
Spring 1990	3.700	3.594
Fall 1990	3.625	3.597
Spring 1991	4.000	3.603

Freezing Audits

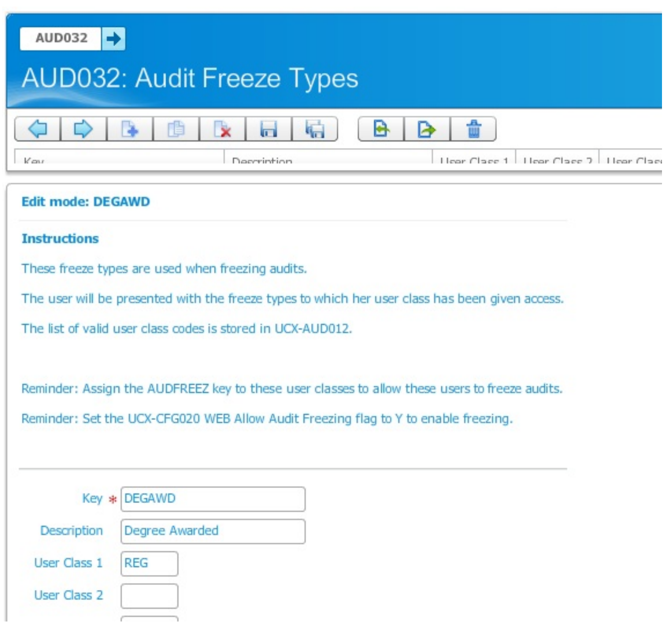
Freezing audits so they don't get deleted
The UCX-CFG020 DAP14 Audit History Count setting controls how many audits to keep per student. When a new audit is processed, Degree Works will delete the oldest audit to ensure this history count is obeyed. However, there are certain times when you want to keep particular audits that do not get deleted you can freeze those audits.

Enabling audit freezing
You can allow users to freeze audits by setting the UCX-CFG020 WEB Allow Audit Freezing flag to Y. Additionally, you can allow users to enter a description when running new audits by setting the Allow Audit Description flag to Y. Users can choose to enter a description on audits but not freeze them, or vice versa. However, normally your users may want to only enter a description when freezing audits –it is recommended that you establish a best practice for your institution.

Only those users who have been given the AUDFREEZ key will be allowed to freeze audits. Additionally, only users with the AUDDESCR key will be able to enter a description on audits. By default the REG, ATHL, and AID user-classes are given these keys. You can use SHPCFG to add or remove these keys on user-classes as needed.

Enabling audit freezing on What-If audits
For what-if audits the keys are WIFFREEZ and WIFDESCR to allow the freezing of audits and to allow users to enter a description. Neither of these keys are assinged to any user-classes by default. Additionally, saving of what-if audits must be enabled by setting the UCX-CFG020 DAP14 What-if History Count value to a number greater than or equal to "01". When this setting is blank or "00" no what-if audits are saved and thus none can be frozen.

Setting up freeze types
When an audit is frozen, the user specifies a freeze type. The freeze type not only specifies that the audit is frozen but also gives an indication as to why is was frozen. The UCX-AUD032 table contains a list of the valid freeze types that your institution is using. This table is initially installed with a set of example freeze types, but you can delete, modify, and add to this list as needed. Each freeze type is associated with up to 10 user-classes. Only users in those user-classes will be allowed to freeze audits with the given freeze type.


AUD032

AUD032: Audit Freeze Types

Edit mode: DEGAWD

Instructions

These freeze types are used when freezing audits.

The user will be presented with the freeze types to which her user class has been given access.

The list of valid user class codes is stored in UCX-AUD012.

Reminder: Assign the AUDFREEZ key to these user classes to allow these users to freeze audits.

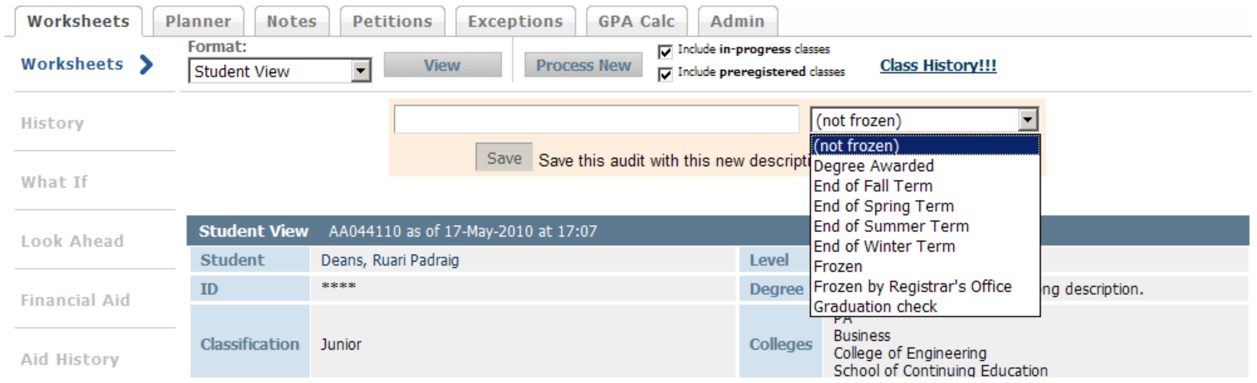
Reminder: Set the UCX-CFG020 WEB Allow Audit Freezing flag to Y to enable freezing.

Key *	DEGAWD
Description	Degree Awarded
User Class 1	REG
User Class 2	

Entering a freeze type and description

When viewing the most recent audit on the Worksheets tab, users with the ability to freeze audits or enter descriptions are given the option to do so. The select box of freeze types is restricted to those set up in UCX-AUD032. However, if the current freeze type on an audit is not associated with the current user's user-class, the freeze type will still appear (selected) in the select box. This allows the user to see the current freeze type and optionally change it to another freeze type.

The user can enter a description without choosing a freeze type, and can also enter a freeze type without entering a description – both fields are optional. The user may un-freeze an audit by choosing the "(not frozen)" option. These un-frozen audits will be deleted from the system when new audits are generated.



Format: Student View

☑ Include in-progress classes

☑ Include preregistered classes

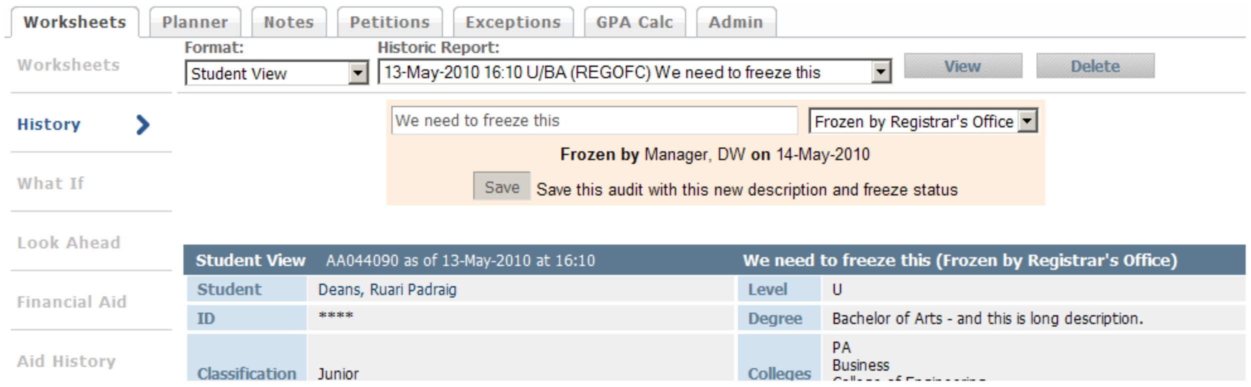
(not frozen)

Save this audit with this new descripti…

Student View	AA044110 as of 17-May-2010 at 17:07		
Student	Deans, Ruari Padraig	Level	
ID	****	Degree	…ng description.
Classification	Junior	Colleges	PA Business College of Engineering School of Continuing Education

On the History tab you can see who froze the audit and when it was frozen. You can also modify the description and freeze type as needed. The description and freeze type also appear in the header of the worksheet itself, so that users who do not have the ability to freeze audits or enter a description can also see this information.

On the History tab you can also Delete any audit – whether it is frozen or not.



Format: Student View

Historic Report: 13-May-2010 16:10 U/BA (REGOFC) We need to freeze this

We need to freeze this

Frozen by Registrar's Office

Frozen by Manager, DW **on** 14-May-2010

Save this audit with this new description and freeze status

Student View	AA044090 as of 13-May-2010 at 16:10	We need to freeze this (Frozen by Registrar's Office)	
Student	Deans, Ruari Padraig	Level	U
ID	****	Degree	Bachelor of Arts - and this is long description.
Classification	Junior	Colleges	PA Business

Freezing audits in a batch

Transit users may freeze audits and enter a description on audits when running DAP22 batch audits. The list of freeze types that appear in Transit are those associated with the REG user-class in UCX_AUD032. If you want all freeze types to appear in Transit, then ensure that all freeze types have REG listed as one of the user-classes in Controller.

Deleting old audits

Frozen audits do not count against the history count value. The Audit History Count may be set to "03", for example, but a student may have several frozen audits and three non-frozen audits. There is no limit on the number of frozen audits that can be kept for each student. For this reason, it is important to limit the number of frozen audits that persist in your database for long periods of time. Audits take up substantial space in your database and freezing many audits that never get deleted will cause you to run out of space eventually. If you do decide to freeze audits, you can consider reducing the history count value to 01 or 02 to help reduce the number of audits saved.

You can have audits frozen with a freeze type of DEGAWD (Degree Awarded), for example, that you do not want to delete but you may have frozen others that you only wanted to keep for a year or two. You can use the **dapdelaudits** script to delete audits older than a specific date and optionally specify a freeze type. If no freeze type is entered, only those non-frozen audits will be deleted.

```
$ dapdelaudits 20091231
Audits older than 20091231 will be deleted.
By default only non-frozen audits will be deleted - but you may choose
 to delete specific audits with a particular freeze type.
Enter the freeze type (or just hit enter for non-frozen audits)? > TRMFAL
Deleting audits older than 20091231 with a freeze type of TRMFAL
Continue with delete? (y/N) >
```

You can supply the freeze type as the second parameter:

```
$ dapdelaudits 20091231 TRMFAL
Deleting audits older than 20091231 with a freeze type of TRMFAL
Continue with delete? (y/N) >
```

You can also supply a confirmation “Y” value as the third parameter to avoid the confirmation message. This is useful when calling dapdelaudits from a script:

```
$ dapdelaudits 20091231 TRMFAL Y
```

You may also use AUD02 in Transit to delete audits in much the same fashion as using the script.

GPA Calculations

All courses applied to each block are used to calculate the BLOCK GPA.
For the Degree Block (or the "starting" block), the GPA is calculated using all the courses applied to the audit (all sections). The Over-the-Limit section can be included or excluded from this calculation (see UCX-CFG020 DAP14 parameter) – but typically the setting is N so that these classes to not count.

The Major and Minor Block GPAs will be calculated using all courses that are or **could be applied** to this block. Courses that end up in "insufficient" or "fail" sections of the audit can be included or excluded from the calculation (see UCX-CFG020 DAP14 parameters for Major and Minor). Classes that could have applied to the Major or Minor block but ended up in the GENED block, for example, will not count in the Major or Minor's GPA calculation – unless the class is being shared between blocks of course.

Redemption Algorithm

After the auditor completes its primary pass through the blocks and rules, it enters one of the last phases of the degree audit process called the Redemption Algorithm. If the class was removed from a rule by mistake, the algorithm works to make the correction. Either the class was removed from a rule and ended up in the fall-through (electives) section, or it was removed from a rule that is allowed to be shared with another rule which the class is also applying. A class is usually removed from a rule because too many credits/classes are applied to a rule and the auditor needs to reduce the number of classes that are applying. Later on, however, one or more of the classes that were kept on the rule may then have to be removed for other reasons (max qualifiers, class should be applied to another block, etc). It is this issue that the Redemption Algorithm attempts to correct.

Fall-through Redemption

When evaluating fall-through classes, the auditor checks all the rules where each class was originally applied at the start of the audit process. For each class, the auditor first examines fits that were on group rules. When a group rule is found that is not complete, the auditor reapplies this class and all other fall-through classes that also fit on this group rule. The auditor then reevaluates the group rules to check if another group(s) option should be kept instead of the one that was previously chosen.

Once the group logic has been performed and if the group rule on which the class was reapplied was not chosen as the group rule to keep, the auditor continues inspecting the other fits for the class. The auditor starts with the best fit for the class and works its way down to the least-best fit – skipping all fits on group rules. Once an incomplete rule has been found, the class is reapplied and no other rules are examined.

Each time a class is placed back on a rule, the Max qualifiers on that rule are reevaluated.

Nonexclusive Redemption

After the fall-through classes have been evaluated, the auditor attempts to find classes that are not doing enough sharing based on the Nonexlusive (also referred to as ShareWith) qualifiers found throughout the blocks. These classes already fit in one or more rules but are possibly allowed to fit in one or more additional rules. For example, the Major rules can share the GenEd block but HIST 1916 was removed from the GenEd for some reason. This step in the redemption algorithm attempts to discover that this sharing is allowed and also attempts to reapply the class to the GenEd block.

For this piece of the algorithm, the auditor steps through each class that is currently applying to at least one rule. The list of the original fits from the start of the audit process is then inspected. This list includes links between fits that indicate if sharing is allowed. The auditor uses the links to check if it can reapply one or more of the fits for the class. When a class is reapplied to a rule, the Max qualifiers on that rule are reevaluated.

After both the Fall-through and Nonexclusive parts of the Redemption algorithm are performed, the auditor performs another check on the header qualifiers in all blocks. In addition, each rule's percent-complete is recalculated.

Too Many Classes on a Rule

At the start of the audit each class is placed on all requirements where the class fits. The auditor then attempts to figure out the best place to keep the class, removing it from the other requirements. In addition, when the auditor has found that too many classes are on a requirement it steps through a set of decision points to determine which classes to keep and which to remove. Each class is compared to each of the other classes on the rule with the auditor making a decision to keep or remove based on the decision points.

For example, let's say the rule has classes A, B, C, D and E but the rule only needs two of the classes. The auditor first compares A and B. If it turns out that B is the better class, based on the decision points, then A is marked as the one to remove. The auditor then compares A with C. If A is now considered to be the better class then C is marked as the one to remove. This continues on down the line until the last class is reached. At this point the auditor has determined which class it the one to remove. If it was determined that class D was the worst out of the list, for example, then D is removed and the cycle starts again comparing the remaining list of A, B, C and E. Since only two classes are needed on the rule the auditor stops when there are two classes remaining. These two remaining classes are the best classes to keep on the rule.

Below are the decision points the auditor uses when comparing two classes on a rule. The auditor steps through each point finding that one of the classes is better than the other or finding that they are equal for the given point. If they are equal only then does the auditor move to the next decision point.

This is the list of decision points, in order. Based on each comparison, one class is kept and the other is marked for removal. The one that is marked for removal is then compared to the next class on the rule and so on. When a decision is made a code is recorded by the auditor and this code shows up in the Diagnostics Report. This information helps you the user figure out why the auditor made the decisions it did.

- Kept class in the including list (SQ)
- Kept class needed by MinAreas (SQ)
- Kept class needed by MinPerDisc (SQ)

- Kept class needed by MinSpread (SQ)
- Kept class that matches rule exactly (1 Class and 3 Credits in) (SQ)
- Kept class with Apply Here exception (li)
- Kept class that was completed over in-progress class (see UCX-CFG020 DAP14 flag) (inpr)
- Kept class that originally only fit this rule and no other (e1f)
- Kept class that currently fits this rule and no other (1f)
- Kept class that is reapplied here from fall-through via Redemption (fara)
- Kept class based on Decide option - if specified (d45)
- Kept class that has more header MIN qualifiers that may have need this class (HMinQ)
- Kept class based on the fit rank (See Fit Rank section) (firk)
- Kept class that is less likely to be removed because of a header qualifier (dect)
- Kept class with fewer exclusive fits on other rules (exfi)
- Kept class with a higher match level (lvl)
- Kept class based on UCX-CFG020 TIEBREAK (tie)
- Kept class based on coin flip

Match Level

The match level is one of the decision points the auditor examines to help determine the best location for the class and also which are the best classes to keep on a rule that has too many classes. On each requirement, the match level is calculated based on how the requirement is scribed.

- Exact match courses get a match level of 6; example: MATH 123
- Courses with a range get a match level of 5; example: MATH 100:199
- Courses with a wildcard number get a match level of 4: example: MATH 1@

However, these changes are then made to the match level:

- If an Apply Here exception is in place – level is set to 99
- If the course is in an Including list – level is set to 16; example: Including MATH 123
- If the course is in a plus-list list – level is set to 16; example: 2 Classes in MATH 123 + 124
- If the course is the only course listed - level is set to 16; example: 1 Class in MATH 123
- For each HighPriority on the block increase the level by 5
- For each HighPriority on the rule increase the level by 5
- For each LowPriority on the block decrease the level by 5
- For each LowPriority on the rule decrease the level by 5
- If the rule’s credit range starts with zero - level is set to 2; example: 0:9 Credits in…
- If the rule has the LowestPriority qualifier - level is set to -9876
- If the rule has a Force Complete exception - level is set to -99

The match level by itself does not determine which classes are removed or kept on rules that have too many credits and it does not determine the requirement on which a class is placed when the class fits on multiple rules, but is one of the decision points used.

Fit Rank

The fit rank is used by the auditor to determine where a class should be placed when it fits on multiple requirements. The auditor uses the decision points below to either increase or decrease the fit rank for a class on a particular requirement. The fits for a class are compared with each other to determine which is the best. A fit with a rank of 1 means it is the best fit; a rank of 2 means it is the 2nd best fit, etc. However, it is possible for two fits to have the same rank - meaning the two fits are equal when it comes to the decision points; neither fit is better than the other. A fit with a higher rank (lower number) will be kept before one with a lower rank (bigger number).

The auditor steps through these decision points in this order. Only when neither fit is determined to better or worse does the auditor proceed to the next decision point.

- Fit is on a rule with a Force Complete exception - worse fit because we can remove the class and the rule will still be complete
- Fit is on a StandAloneBlock - worse fit but only because we know all StandAloneBlock fits will remain so some other fit should be considered a better fit
- Fit has an Apply Here exception - better fit because the class must be applied here no matter what
- Fit is on a plus-list or including-list rule (and not in a group or a wildcard/range w/ multiple fits) - better fit because the rule explicitly says the student must take this class
- Fit is on a rule with another class that only fits here - worse fit because most likely this class will be removed from this rule in favor of the other class
- Fit's rule has a higher priority (per HighPriority, LowPriority, LowestPriority) - better fit because of the priorities specified
- Fit is only class on this rule - and does not have LowPriority - better fit because this rule needs this class; without it, the rule will be 0% complete
- Fit has a higher match level - better fit compared to the levels of other fits (which may be on a range or a wildcard requirement etc)
- Fit is on a group rule that may not be needed - worse fit because by definition a group rule is a list of options and this may not be the best option to take
- Fit is needed on the rule; removal would make rule incomplete - better fit because without this class the rule will not be 100% complete

Group Procesing

When classes are matching multiple rules within a group the auditor must determine which of the options to keep and which to discard. The auditor uses the set of decision points below to make this decision. One important thing to keep in mind here is that the auditor does this group processing early on in the audit and thus it is making decisions before sharing has been resolved and before excess classes have been removed from requirements.

Here is the list of decision points to determine which groups to keep and remove:

- Keep the rule if it has an **exception**
- Keep the rule that has **complete qualifiers** (removing those rules with incomplete qualifiers)
- Keep the rule if it is **more complete** than another rule
- Keep the rule that has at least one **completed class** if multiple rules are incomplete because of in-progress classes
- Keep the rule that has a **higher priority**
- Keep the rule that is needed to satisfy the group's **MinPerDisc** qualifier
- Keep the rule whose classes have a **higher average fit rank**
- Keep the rule with **fewer min/max qualifiers** (since it a less complicated rule)
- Keep the rule whose classes have **fewer fits on other rules** (averaged over the number of classes on the rule)

Removing classes when too many fit on a rule

When too many credits/classes are applied to a rule, the auditor needs to figure out which classes to remove from the rule.

Example:

```
3 credits in MATH 101, ENGL 101, PSYC 101, HIST 101
  Label INTRO "One introduction-level class";
```

Let us assume these are all 3-credit classes.
Let us assume a case where a student has taken all 4 of these classes.
Let us assume that PSYC 101 is found out to be the best fit when comparing these four classes for this student.

Again, we only need 3 credits. We take these 12 credits and ask ourselves, "which is the worst fit"? We start by comparing two classes to one another. The order in which these are applied is not important, as one can see below. For clarity's sake let us assume they are compared in the order Scribed.

MATH 101 is first compared with ENGL 101

We use the "Remove excess classes" decision-process (this is near the bottom of every Diagnostics Report) to compare these two classes. Inevitably, one is considered a "worse fit" than the other. Let's say ENGL 101 is the worse fit. MATH 101 is kept (for now), and ENGL 101 is then compared to PSYC 101. ENGL 101 is still worse than PSYC 101, then class C is kept (for now) and ENGL 101 is compared to HIST 101. Let's say HIST 101 is worse than ENGL 101. Now there are no more classes to compare.

So:
MATH 101 is better than ENGL 101
PSYC 101 is better than ENGL 101
ENGL 101 is better than HIST 101

This means that HIST 101 is the **worst**. We remove it from the rule. We have not compared HIST 101 to PSYC 101, nor to MATH 101, but because of the transitive property, we can rely on the fact that MATH 101 is better than ENGL 101, and ENGL 101 is better than HIST 101. **Therefore** MATH 101 **must be better** than HIST 101.

Getting back to the example, now we have three classes still on the rule since we just removed only HIST 101. That's still too many credits. So we do it again. This time around we determine that:

MATH 101 is better than ENGL 101
PSYC 101 is better than ENGL 101

This means that ENGL 101 is the **worst**. We remove it from the rule.

That leaves us with MATH 101 and PSYC 101, and that's still too many credits. So we do it again. This time, we determine (again, based on the "Remove excess classes" decision-process) that:

PSYC 101 is a better fit than MATH 101.
PSYC 101 is better than MATH 101

So – MATH 101 is removed. Notice that this is the **first time** that MATH 101 was compared to PSYC 101, but because of the work that was done prior to this we can with confidence conclude that:

Based on the classes that originally fit on this rule, when comparing these classes – PSYC 101 is the best fit.

Note that decisions later in the audit process could impact this rule. Perhaps PSYC 101 fits on a different rule in this block or elsewhere in the audit. At that point (later in the audit process) a decision would have to be made: should PSYC 101 stay here on this "One introduction-level class" rule – or should it apply to a different fit? If it turns out that PYSC 101 should be placed on some other rule, the redemption step of the audit process will come into play reapplying one of those other 101 classes to this rule – but only if they themselves are not applying to some other rule.

Evaluating Classes on a Rule

When too many credits/classes are applied to a rule, the auditor needs to figure out which classes to remove from the rule. The auditor goes through the following steps to determine which classes to keep and which to remove:

1. **Kept class in the including list**

Given a rule such as the following:
```
5 Classes in MATH @ Including MATH 201
```
the auditor will keep the "included" class since it is required.

2. **Kept class with Apply Here exception**

If a class has an Apply Here exception, it will be kept on a rule before all others.

3. **Kept class needed by MinAreas**

4. **Kept class needed by MinPerDisc**

5. **Kept class needed by MinSpread**

Given a rule such as the following:
```
2 Classes in MATH @, CHEM @, BIOL @, PHYS @ MinSpread 2
```

Assuming MATH 101, CHEM 101 and CHEM 102 are on this rule, the auditor will keep the MATH 101 class on the rule because it is needed to satisfy the MinSpread qualifier. The same approach is used for MinAreas and MinPerDisc as well.

6. **Kept class that matches rule exactly (1 Class and 3 Credits in)**

Given a rule such as the following:
```
1 Class and 3 Credits in ACCT 2@
```

Assuming ACCT 201 for 1 credit, ACCT 202 for 2, credits and ACCT 203 for 3 credits are applied here, the auditor will keep ACCT 203 because it matches the requirement of 1 class and 3 credits exactly.

7. **Kept completed class and discarded in-progress class**

When the UCX-CFG020 DAP14 In-progress vs Completed flag is set to Y, a completed class is given preference over an in-progress class at this step in the algorithm. This usually works because in-progress classes that end up in fall-through can be reapplied to this rule by the redemption algorithm, if the rule still needs additional classes/credits. If the configuration flag is set to N, this step is skipped. Because setting this flag to Y alters the auditor's best-fit algorithm it is recommended that this flag be left as N so that this step is skipped. When this flag is set to Y you are telling the auditor to make a different choice than it would normally make and thus it is unable to apply classes in the most efficient manner.

8. **Kept class that originally only fit this rule and no other**

If a class can only fit this rule and no other rule, then it is kept over another class that has or had multiple possible fits.

9. **Kept class that currently fits this rule and no other**

If a class did fit on multiple rules at some point, but now only fits this rule, it will be kept over a class that is now still placed on multiple rules.

10. **Kept class that is reapplied here from fall-through via Redemption**

If a class was removed from all rules and placed in fall-through, but was then placed back onto this rule through redemption, it will be kept.

11. **Kept class based on Decide option - if specified**

Given a rule such as the following:
```
2 Classes (Decide = BESTGRADE) in MATH @, CHEM @
```

The auditor will obey the DECIDE operator when deciding which class to keep if none of the above steps has helped make a decision.

12. **Kept class based on the fit rank (See the Fit Rank section in the Diagnostics Report)**

When a class has multiple fits, its fits are ranked from 1-x where 1 is the best place to keep the class and x being the least best place to keep the class. So when comparing two classes that fit this rule, the fit rank of each on this rule is considered. If one of them has a fit rank of 2 on this rule and the other has a fit rank of 3 on this rule, the class with a 2 is kept because for that class this is the better place to be kept.

13. **Kept class that is less likely to be removed because of a header qualifier**

If we have a header qualifier such as the following:
```
MaxClasses 3 in ART 108, 109, 2@
```

And we have a rule such as the following:
```
1 Class in ART 108, 123, 145
```

The auditor sees that ART 108 is part of a MAX qualifier and so has a greater chance of being removed from this rule. The other class (that is not part of any qualifier) is kept on this rule instead.

14. **Kept class with fewer exclusive fits on other rules**

Given a rule such as the following:
```
1 Class in BUS 106, 108;
```

BUS 106 may fit on two other rules – both of which are not shared with this rule.

BUS 108 may fit on two other rules – but one of them is in a block shared with this block.

This means that BUS 106 has two other exclusive (not shared) fits while BUS 108 has only one other exclusive (with the other one being shared) fit. Thus, BUS 108 will be kept on this rule because there is a greater chance that BUS 108 will end up on this rule since it will either go here or on that other exclusive fit – while there is only a 1-in-3 chance that BUS 106 will end up on this rule.

15. **Kept class with a higher match level**

Given a rule such as the following:

```
1 Class in ANTHRO 266, 277 3@;
```

Given the choice between keeping ANTRHO 266 or ANTRHO 304 on this rule, the auditor will keep ANTRHO 269 because it is specifically listed. However, if given the choice between ANTHRO 266 and ANTHRO 277 both are specifically listed – but if one of them is in-progress its match level is actually slightly lower – so in this case the completed class will be kept instead of the in-progress class. In other words, in-progress classes always have a match level lower than they would have if they were completed.

16. **Kept class based on UCX-CFG020 TIEBREAK**

If all other steps cannot help the auditor make a decision about which class to keep, it uses the CFG020 TIEBREAK setting to figure out which class to keep. Usually this step decides which to keep because one is in-progress and one is completed, or one has a higher grade, or one has a higher course number, etc. You can control whether the in-progress vs completed comparison is the first TIEBREAK check the auditor performs – but you need to alter the settings using Controller to configure it according to your requirement.

17. **Kept class based on coin flip**

In very rare situations, for example, when a class is repeated many times (such as PE and ART classes), the TIEBREAK checks do not help the auditor make a decision. At this point the two classes in question are basically identical so the auditor makes a decision based on a randomly generated choice.

In-progress vs Completed

The steps the auditor goes through to evaluate the classes on a rule detailed in the section above. When each step is evaluated and if both classes have “equal values” for that particular step, the auditor continues to the next step to see if it can find a reason to keep one class over another class. The auditor looks at many properties for the classes in question, and based on a UCX-CFG020 DAP14 configuration flag, it also checks to see if one of the classes is in-progress.

It is only at step #7, that the algorithm directly examines whether a class is in-progress when deciding which rule to keep. Step #7 is used or ignored based on a UCX-CFG020 DAP14 flag.

Logging Degree Works Errors

Degree Works uses the UNIX syslog to log conditions that may be of interest to the administrator. Degree Works can be configured so that only serious errors go to the log or so that all error conditions go to the log. Logging within Degree Works can also be turned off as needed. The system log file(s) can be interrogated on a regular basis by the administrator to track for problems with the software.

PSK84 – SBISYSLOG

The PSK84 sbisyslog routine controls the logging of information to the system log. PSK84 reads environment variables to control whether logging is on or off and what kind of information is logged:

SBI_SYSLOG

Set to 0 to turn off logging. Set to 1 to turn on logging. If the variable is not found logging will be on by default.

```
export SBI_SYSLOG=1
```

SBI_SYSLOG_FACILITY

The facility can be used to help differentiate those log entries that pertain to your test and those that pertain to your production environment. The variable should be set to a value from 16 – 23. Doing this will cause the token of “local0” – “local7” to appear in the log. Some systems to not display this information however. If the variable is not found 16 or “local0” will be used as the default. The local facilities must be configured in your **syslog.conf** so that these type of errors are directed at your log file. Log entries tied to a specific facility can be sent to one file while entries with a different log entry can be sent to another file.

```
export SBI_SYSLOG_FACILITY=16
```

SBI_SYSLOG_PRIORITY_MAX

The priority-max environment variable can be used to restrict which types of entries appear in the system log. Setting the max to 0 will direct the operating system to only log entries with a 0 or EMERG priority and ignore all attempts to log any other type of priorities. Setting the max to DEBUG or 7 will direct the operating system to log all entries with priorities 0 – 7. Degree Works makes use of priorities 2, 3, 4, 5 and 6. User logon errors, for example, are logged with a priority of 6 while more serious conditions such as not being able to open the database are logged with a priority of 2.

```
export SBI_SYSLOG_PRIORITY_MAX=7
```

PSK84 uses the **LOG_PID** and **LOG_CONS** options when opening the system log. The former tells the OS to log the PID with each message while the latter tells the OS to log the message on the console if there is a problem writing the system log file.

For more information:

See sys/syslog.h
See man pages for syslog, syslogd, openlog, logger

syslog.conf

Your system’s syslog.conf file controls which priorities and facilities get logged to which log file. Be sure your syslog.conf file is configured to support the facilities and priorities being used by Degree Works. Below are extracts from different syslog.conf files. The syslog.conf file is usually found under /etc.

Aix example:

```
mail.debug                /usr/local/logs/mailog
*.emerg                   /usr/local/logs/syslog
*.alert                   /usr/local/logs/syslog
*.crit                    /usr/local/logs/syslog
*.err                     /usr/local/logs/syslog
auth.notice               /usr/local/logs/syslog
local2.debug              /var/log/sudo.log
daemon.info               /usr/local/logs/infolog
```

Sun example:

```
*.err;kern.notice;auth.notice                     /dev/sysmsg
*.err;kern.debug;daemon.notice;mail.crit          /var/adm/messages
```

```
*.alert;kern.err;daemon.err                     operator
*.alert                                         root
*.emerg                                         *
```

HP-UX example:

```
mail.debug                              /var/adm/syslog/mail.log
*.alert;*.critical;*.info;mail.none     /var/adm/syslog/syslog.log
*.alert                                 /dev/console
*.alert                                 root
*.emerg                                 *
```

You could setup your configuration file to send all local0 critical messages to one file and all local1 critical messages to another file:

```
local0.crit            /var/log/prodenv.log
local1.crit            /var/log/testenv.log
```

The Degree Works sbisyslog routine attempts to log the program and subroutine where the condition was encountered. Errors will be logged differently based on the kind of error that is encountered and the program that encounters the error. There is no fixed format that all messages follow. All Degree Works messages do contain "DW-" followed by a four-byte program name followed by the subroutine or program that encountered the error. The PID of the process encountering the condition is logged. On Sun the actual facility and priority is placed in the system log while other systems may not log this information.

Example log entry from HP-UX:

```
Jan 26 13:32:48 hpdev DW-DP16-DAP16[8749]:
ID/Type/Value=RA000002/DEGREE/BA Status=   -1 Error=    0
```

Example log entry from Sun:

```
Nov 21 16:05:46 Figment DW-DP16-DAP16: [ID 702911 local0.notice]
ID/Type/Value=RA000002/DEGREE/BA Status=   -1 Error=    0
```

You can use the system's **logger** command to test where your system messages end up and be sure the syslog daemon is running.

```
$ logger -p daemon.notice -t FIRSTTEST Now it Works
$ logger -p local0.crit -t MYTEST Test of a critical error
```

Viewing logs

You may want to write your own scripts to examine the contents of your system log files. Any information that seems to contain errors can be sent to the Ellucian help desk and may assist in troubleshooting a problem.

The **dgwlogger** command and **dgwlogshow** commands may also be useful tools to test out adding log entries and viewing them:

```
bin/dgwlogger
bin/dgwlogshow
```

Cleanup

You may want to clean out your system logs regularly so that they contain only recent information.

Multi-entity Processing

Some institutions need to deploy Degree Works across multiple degree-granting campus entities within their institution. Other institutions serve constituents at only one campus and have no need to share certain information beyond a single entity. Degree Works is configurable so that institutions may elect to utilize only one instance of Degree Works at a single entity or campus, or deploy Degree Works services across many campuses or entities while specifying particular types of data that ought to be shared among those entities. Configuration to specific institutional needs allows sites to manage both data and system administration in a manner best suited to their particular needs.

Recognizing that terminology in Higher Education can be ambiguous, understand that in the context of multi-entity processing, we are using "campus" to mean a separate, degree-granting institution which is either "all by itself" (Entity=1) or part of a district or system of sister institutions (Entity=many) which has a central administrative arm and installation of Degree Works. We recognize that "campus" can also mean "location", where the campus is a satellite venue for the larger degree-granting institution. In this discussion of multi-entity processing do not think of a "campus" or "campus entity" as simply a locale or the case study will be confusing.

There are different reasons why an institution would opt for multi-entity processing among multiple campuses. For multi-entity institutions, sharing data enables campuses to use commonly accessed information, thereby reducing the disc space that database tables require. Sharing Oracle databases among entities conserves memory needs on the system and reduces the level of redundant system administration needed for managing the application. Degree Works has the ability to support multiple campuses sharing a variety of academic and biographic data, as well as the sharing of academic requirements among campuses. Degree Works may be configured to share certain tables among some or all campuses, and also allows certain tables to be configured uniquely for individual campus entities.

Institutions supporting only one campus or entity need not configure anything special for their campus installation or updates; they may utilize default values delivered with the product. Institutions supporting multiple campuses or entities have several options when activating multi-entity processing. These options include:

1. Individual Oracle database schemas for each campus entity with unique data for each campus (i.e. nothing shared)
2. Individual Oracle database schemas for each campus entity with some shared data used by some or all campus entities, along with some unshared campus-specific data
3. A commonly shared Oracle database schemas with common access to all data by each campus entity

This article focuses on scenario #2 in which an institution deploys individual databases with some shared data and some unique data. The case study is intended to familiarize institutions with some of the configuration issues to consider before adopting multi-entity processing for multiple campuses.

Case Study as Example

In this case study, the Springfield Community College District is comprised of four campus entities: North College, South College, East College and West College. Students may attend any of the four campuses sequentially or simultaneously, with classes taken at any campus yielding in-residence credit. The four campuses share a common course numbering system with similarly named courses also having common course content. Not every class is offered at every campus entity. It is assumed that a student will generate audits at those District institutions at which he/she is a matriculated, degree-seeking student. In such cases, the student will have one or more rad_goal_dtl ("Degree records") for each campus at which he/she has declared an intended degree.

To reduce database maintenance while giving students and advisors optimum Degree Works services, Springfield plans to configure Degree Works to share as much data as possible between the four colleges or campus entities. Data to be shared includes the course catalog, all student academic data, and UCX validation codes and literals. Each campus entity, however, will maintain its own set of degree requirements and wants the Degree Works audit worksheets to reflect each campus's needs and preferences. To accommodate sharing, Springfield will want to allow common access to most of the RAD database tables. To assure individual maintenance of degree requirements and other audit needs, Springfield will want to use the DAP database tables in the proper configuration for individual campus entities.

RAD tables

The student data imported from Springfield's student system is housed in these tables:

```
rad_Primary_mst
rad_Student_mst
rad_Biog_mst
rad_Noncrse_dtl
rad_Test_dtl
rad_Report_dtl
rad_Attr_dtl
rad_Applicnt_dtl
rad_Previnst_dtl
rad_Class_dtl
rad_Custom_dtl

rad_Transfer_dtl
rad_Term_dtl
rad_Aid_dtl
```

Configuring Degree Works to share these tables guarantees that all campus entities use the same set of classes, test scores, and other academic information for each student, regardless of the campus or campuses at which they matriculate. Springfield does want to assure accurate auditing of academic information against the program requirements for each campus entity, but also wishes to conserve table maintenance and discspace. For that reason, it will share some RAD table information among campuses, but it will not share all institutional data.

To achieve the desired outcome, the District will not share the Degree records. These tables are struck through on the above list to demonstrate that it is not to be shared. The Degree record tables contain a student's degree, major, minor, catalog-year etc information and really need to be linked to the degree-granting campus entity. In other words, this table should be unique to each campus entity or college so that if a student only matriculates at North College, then the other campus entities would not run audits or review data for that student. If the student is matriculated at both the North and South campuses then each of those entities will have their own degree record for that student, and an audit may be generated for that student at each of those campuses.

Each campus will bridge data from the student system that includes any coursework from the District campuses, but each campus will only evaluate data for matriculated students with declared degrees at its campus. North College should be sending to Degree Works all of the students who have matriculated at North regardless of where within the four colleges students have actually taken coursework. A student matriculated at North College may have taken classes at North, South, East and West, but only the student system extract for North should be pulling this student's Degree records data. North will also extract the coursework taken at all four campuses. Extracts generated at South, East and West will not extract this student at all since the student is not a matriculated degree-seeking student at those campuses and the student's Degree records are not shared.

Some students may be matriculated at more than one campus, in which case their student data will be bridged to each matriculated campus and available for auditing. When extracts are run at each campus where a student is matriculated, that student's data will be extracted multiple times but their shared data will only exist once in Degree Works, while their unique campus Degree records will exist for each campus.

DAP tables

As stated above, DAP database tables will be configured to reflect the unique degree requirements for each campus. For example, the humanities requirement at North College is similar but somewhat different from the humanities requirement at East College. Springfield will configure Degree Works so that the requirements tables are not shared among the campuses, and so that each entity has its own table.

```
dap_Req_mst
dap_Req_text_dtl
dap_Req_link_dtl
dap_Req_crs_dtl
```

Likewise, degree audits need to be retained by each campus entity individually, and therefore audit tables are not shared. Additionally, the exceptions and notes associated with those audits are not shared.

```
dap_Student_mst
dap_Except_dtl
dap_Note_dtl
dap_Note_txt_dtl
dap_Audit_dtl
dap_Audtree_dtl
```

When using the Curriculum Planning Assistant tool the audits are extracted to the tables noted below. These tables would also be unique to each campus entity and remain unshared.

```
dap_Result_dtl
dap_Resclass_dtl
dap_Resnoncr_dtl
```

Academic plans for students are linked to their academic requirements. That also dictates that planner tables should not be shared and that each campus entity would maintain unique planner data.

```
sep*
temp*
```

At the end of this section there is a list of Degree Works tables that can be configured for multi-entity processing. Note that the list of UCX_* tables can be reviewed using Controller.

Transit tables

The Transit tables should never be shared and thus should never be placed into the share file.

How table sharing is accomplished

The sharing of tables among multiple campus entities (for example, North, South, East and West) within an institution (Springfield CCC) is accomplished using Oracle synonyms. Degree Works uses synonyms to point to the real table owned by another schema.

For example, schemaNorth's rad_Class_dtl table may really be just a synonym that points to the rad_Class_dtl table owned by schemaAll. When synonyms are created like this for all campuses they end up sharing the data stored in that table:

```
CREATE SYNONYM SCHEMANORTH.RAD_CLASS_DTL FOR SCHEMAALL.RAD_CLASS_DTL;
CREATE SYNONYM SCHEMASOUTH.RAD_CLASS_DTL FOR SCHEMAALL.RAD_CLASS_DTL;
CREATE SYNONYM SCHEMAEAST.RAD_CLASS_DTL  FOR SCHEMAALL.RAD_CLASS_DTL;
CREATE SYNONYM SCHEMAWEST.RAD_CLASS_DTL  FOR SCHEMAALL.RAD_CLASS_DTL;
```

The real table and associated indexes and triggers are created under the schemaAll user.

Degree Works delivers a **dwschema.xml** document defining all of the tables created for use with the application. The sharing rules of tables are defined in a share.xml document. The default share.xml document used for an institution containing a single college where no sharing is needed looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<EntityDatabaseConfig Version="1.0"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:noNamespaceSchemaLocation="dwshare.xsd">
 <ClientName>DegreeWorks</ClientName>
 <DefaultSharing>Unique</DefaultSharing>
  <Entities>
   <Entity Id="DegreeWorks">
    <Name>DegreeWorks</Name>
    <Schema>dwschema</Schema>
    <SchemaOutputFile>dwschema.sql</SchemaOutputFile>
   </Entity>
 </Entities>
</EntityDatabaseConfig>
```

For an institution like Springfield where multiple campus Entities must be defined, the document looks like this:

```
<Entities>
  <Entity Id="NorthCollege">
   <Name>North College</Name>
   <Schema>schemaNorth</Schema>
   <SchemaOutputFile>schemanorth.sql</SchemaOutputFile>
  </Entity>
  <Entity Id="SouthCollege">
   <Name>South College</Name>
   <Schema>schemaSouth</Schema>
   <SchemaOutputFile>schemasouth.sql</SchemaOutputFile>
  </Entity>
  <Entity Id="WestCollege">
   <Name>West College</Name>
   <Schema>schemaWest</Schema>
   <SchemaOutputFile>schemawest.sql</SchemaOutputFile>
  </Entity>
  <Entity Id="EastCollege">
   <Name>East College</Name>
   <Schema>schemaEast</Schema>
   <SchemaOutputFile>schemaeast.sql</SchemaOutputFile>
  </Entity>
 </Entities>
```

Multiple **ShareGroups** may be defined so that subsets of campuses may share certain information while another subset “shares” only their own data. For Springfield a **ShareGroup** is defined allowing all four colleges specified to share the tables listed within:

```
<ShareGroup Id="sgAll" Schema="schemaAll" SchemaOutputFile="schemaall.sql">
  <EntityId>NorthCollege</EntityId>
  <EntityId>SouthCollege</EntityId>
  <EntityId>EastCollege</EntityId>
  <EntityId>WestCollege</EntityId>
  <DatabaseTable>RAD_ETS_MST</DatabaseTable>
  <DatabaseTable>RAD_COURSE_MST</DatabaseTable>
  <DatabaseTable>RAD_NEXT_ID_MST</DatabaseTable>
  <DatabaseTable>RAD_PRIMARY_MST</DatabaseTable>
   … etc etc
</ShareGroup>
```

For a more complete example, review the app/schema/**mepshareexample.xml** document. That document articulates examples where North and South share some tables, East and West share some tables, all campus entities share some tables, and each campus has some unique, unshared tables. This sample document shows that the parent tables must come before the child tables; do not change the order of the tables in this document once they have been setup by the Ellucian Services team.

Creating the database tables

Please see the **dbbuild** script outlined in the *Special Scripts* section of the *Technical Guide*. Please be sure to consult with Ellucian staff to configure your share.xml and before running dbbuild.

Integrated Interface for Banner (Banner Student System sites only)

See the *Banner Database Issues* section of the *Banner Considerations Technical Guide* for Multi-Entity Processing configuration instructions.

Web access

Only students who have a degree record for a particular college can be accessed and processed within the Degree Works web interface.

List of tables used in Degree Works

This is a list of Degree Works tables that may be configured by institutions wishing to utilize multi-entity processing at multiple campuses. Note that the list of UCX_* tables can be reviewed using Controller. Sites considering use of multi-entity capabilities may wish to schedule a technical consulting call with Ellucian to review their plans and particular needs prior to activating these features.

Table name	Description
dap_applicnt_mst	Transfer Equivalency student applicant record
dap_audit_dtl	Stores a degree audit
dap_audtree_dtl	Stores the body of the degree audit – like a BLOB
dap_college_dtl	Transfer Equivalency transfer school record
dap_eqv_crs_mst	Stores a college equivalence records
dap_except_dtl	Stores audit exceptions/waivers
dap_map_cond_dtl	Transfer Equivalency articulation data
dap_mapping_dtl	Transfer Equivalency articulation data
dap_next_id_mst	Sequence numbers for scribe blocks, audits, etc
dap_note_dtl	Stores notes for a student – when created, by whom, etc
dap_note_txt_dtl	Stores the actual note text
dap_plancrs_dtl	Stores a planned class and term value
dap_planner_dtl	Stores planner information for a student
dap_plannote_dtl	Stores planner note information
dap_pt_crs_dtl	Stores planner template information
dap_pt_note_dtl	Stores planner template information
dap_req_crs_dtl	Stores list of courses referenced in a scribe block
dap_req_link_dtl	Stores links from one scribe block to another
dap_req_mst	Stores primary and secondary tags for each scribe block
dap_req_text_dtl	Stores the actual notepad-like text for each scribe block
dap_resclass_dtl	Stores audit CPA data
dap_resnoncr_dtl	Stores audit CPA data
dap_result_dtl	Stores audit CPA data
dap_student_mst	Stores date/time of last audit and locking information
dap_template_mst	Stores planner template information
dap_title_dtl	Transfer Equivalency articulation data
dap_transfer_dtl	Transfer Equivalency articulation data
dap_undecide_dtl	Transfer Equivalency articulation data
rad_aid_dtl	added in DW 4.0.0

rad_applicnt_dtl	Transfer Equivalency application data
rad_attr_dtl	Stores attributes about each class the student has taken – transfer_dtl and class_dtl
rad_class_dtl	Stores in-residence classes taken – historic and in-progress
rad_course_mst	Stores courses offered by the institution: title, credits, etc
rad_crs_attr_dtl	Stores attributes about each class offered by the institution – associated with the course-mst
rad_custom_dtl	Stores other information about the student need by scribe requirements
rad_degree_dtl	Stores degree, major, minor, etc info (deprecated)
rad_ets_mst	Transfer Equivalency list of transfer schools
rad_goal_dtl	Stores school, degree, student level, catalog year information
rad_goaldata_dtl	Stores fields of study, such as major, minor, concentration, as well as advisor information
rad_log_dtl	Log of bridge activity
rad_next_id_mst	Next-id information for courses, students and ETS
rad_noncrse_dtl	Stores student non-course data
rad_previnst_dtl	Stores student's previous degree information
rad_primary_mst	Stores student name
rad_report_dtl	Stores other student data that needs to appear on the worksheet
rad_student_mst	Stores the student's active term
rad_swap_id_dtl	Stores a record of when a student ID was changed from one value to another via the bridge
rad_term_dtl	Stores student cum GPA/credits
rad_test_dtl	Stores student test score information
rad_transfer_dtl	Stores transfer class information
rad_hash_mst	Stores a record of what was loaded for this student; works with all rad_*_hsh tables
rad_aid_hsh	Stores hash value for the aid_dtl data
rad_applicnt_hsh	Stores hash value for the applicnt_dtl data
rad_attr_hsh	Stores hash value for the attr_dtl data
rad_class_hsh	Stores hash for the class_dtl data
rad_custom_hsh	Stores hash for the custom_dtl data
rad_degree_hsh	Stores hash for the degree_dtl data (deprecated)
rad_goal_hsh	Stores hash for the rad_goal_dtl
rad_goaldata_hsh	Stores hash for the rad_goaldata_dtl
rad_noncrse_hsh	Stores hash for the noncrse_dtl data
rad_previnst_hsh	Stores hash for the previnst_dtl data

rad_report_hsh	Stores hash for the report_dtl data
rad_student_hsh	Stores hash value for the primary (name), biog (birthdate and SSN) and student (active-term) data
rad_term_hsh	Stores hash for the term_dtl data
rad_test_hsh	Stores hash for the test_dtl data
rad_transfer_hsh	Stores hash for the transfer_dtl data
shp_group_mst	Loaded from UCX_SHP077; specifies default keys/access for each user-class
shp_log_dtl	Stores web activity information
shp_passport_mst	Stores the Degree Works session ID identifying the user’s access level (key ring)
shp_service_mst	Stores key-ring for tabs, functions, etc – though normally key and service names match
shp_user_mst	Stores user’s ID and password and primary user class

For a list of the UCX tables and their descriptions, see the *Degree Works Configuration Technical Guide*.

Percent Complete Calculation

A percent complete calculation is done for each rule, each block and for the overall audit. It is this calculation which is used to mark a rule, a block or the overall audit as "completed".

Rule Completeness

Course Rule

At the course rule level, the calculation can take on some complex characteristics. Basically, the calculation determines the number of classes or credits that are required for the rule and then calculates the number of classes or credits applied to the rule. The percent complete is calculated from those two factors. If a course is applied to a rule but has not yet been graded (i.e. it is "in progress") and the rule would be completed with that course, the percent complete is reduced to 98%. In the Degree Works reports we show these in-progress rules with a single squiggle signifying that the rule is close to being completed. If the student ends up failing the class it will be placed in Insufficient and the rule will end up with an empty box. There is no guarantee that the in-progress class will actually end up on the rule once it has been completed since other classes the student registers for may cause classes to be shifted around.

A rule qualifier that is not met will make the rule become 99% complete – given the required credits/classes were taken. A MinSpread or MinPerDisc qualifier that has not been met will cause the rule to be marked as 99% complete and will appear with a box with a double-squiggle on the Degree Works worksheet.

Noncourse Rule

The percent complete is calculated based on the total number of noncourses required and the number of noncourses completed.

Subset Rule

All the rules within the subset form the basis of the percent complete calculation. If any subset qualifiers are not satisfied (and all the rules are complete) the percent complete is reduced to 99%. If all rules are complete but one or more contains an in-progress the subset will be considered 98% also – the subset will inherit this property of its child.

Group Rule

The group(s) that is the "most complete" is used as the basis of the percent complete calculation. For example, a group rule states that 1 group is needed from a list of four groups.

```
1 GROUP in
  (8 CREDITS IN BIOL 100:199) or
  (8 CREDITS IN CHEM 100:199) or
  (8 CREDITS IN PHYS 100:199) or
  (9 CREDITS IN MATH 250 + CHEM 200 + CHEM 220)
```

If 6 credits have been applied to the last group and 4 credits to the first group, the last group will be used to do the percent complete calculation.

If the first two rules have 4 credits applied the auditor will simply chose one given everything else equal between the classes in question. The class on the rule not chosen is freed up to be used on another rule or will be placed in fall-through.

Block and Blocktype Rule

The percent complete is based on the details of the rule in the referenced block. If the referenced block is not found in the audit, the percent complete is zero.

If/Then/Else Rule
When the IF condition is FALSE and there is no ELSE rule, the IF statement is not included in the block percent complete.

When the IF condition is true, the calculation is based on the rule type in the THEN portion of the IF rule.

Block Completeness

Each block's percent complete is based on the rules and the block qualifiers. The rules are counted at the highest level for this calculation. If a Subset Rule is included in the block, it is counted as a single rule (it's completeness has already been calculated at the rule level). If all rules within a block are complete but there are block qualifiers that are not satisfied, the completeness is reduced to 99%.

If the block is optional then it is 100% complete. If it is not optional then add up the number of rules at the first level and their total percent complete. Divide the total percent complete by the number of rules counted to get the block's completeness. If all rules have been completed but there is a block header qualifier that is not satisfied, the percent complete is reduced to 99%.

Overall Audit Completeness

The overall audit percent complete is calculated from the rules within each block being used by the Auditor. (If a rule has not been used, e.g. within an IF statement, it is not included in the calculation.)

For each block that does not have an OPTIONAL qualifier do the following: Add up the number of rules at the first level - this means do not count each rule in a subset or group. Add up the percent complete of each of the rules counted. Divide the total percent complete by the number of rules counted to get the overall completeness. If all rules have been completed but there is a block header qualifier that is not satisfied, the percent complete is reduced to 99%.

Output Options

Instead of showing the percent complete for each block, Degree Works maps each percent value to a graphic.

Percent complete	Meaning	Visual display
0 - 97	Not complete	Empty box
98	In-progress incomplete	Single squiggle box
99	Qualifier incomplete	Double squiggle box
100	Complete	Checked box

Repeated Classes

The Auditor needs to know what rules to follow when applying Repeated Classes to the audit. Repeated classes are identified using the Repeat-Ptr and Repeat-Plcy fields on the rad-class-dtl. If the Repeat-Ptr field is non-blank it is assumed to be a repeat. The Repeat-Ptr field contains the discipline and number of the course it is repeating – regardless of which occurrence it is. The Repeat-Plcy tells Degree Works how to handle the repeat set with regard to credit and GPA calculations. Your school's forgiveness policy is important in determining which repeat policy to use here.

Repeated Courses whose course numbers have changed
If a student originally took a course as MATH 150 and retakes it as MATH 153 or STAT 153, the Auditor needs to know that these two classes are associated. The Repeat-Ptr field on the rad-class-dtl can record this.

Degree Works Repeat Policies

Policy 1	The credits and grade points of the last (most recent) occurrence are counted by Degree Works. Earlier occurrences are forced to the "insufficient" section of the audit. Those in insufficient do not count in GPA calculations. **Most recent counts – others do not.**
Policy 2	The credits and grade points of the class with the best grade are counted by Degree Works. The other occurrences go to "insufficient". Those in insufficient do not count in GPA calculations. **Best grade counts – others do not.** However, when two classes have the same grade, Degree Works needs to know which class to keep and which should be placed in insufficient. The Additional Control Flag in UCX-AUD047 is used for this situation. You can set it to O to keep the Oldest class and N to keep the Newest class when two classes have the same grade.
Policy 3	All occurrences are counted by Degree Works. The audit should apply courses wherever they fit or in the "fallthrough" or "insufficient" section of the audit. **All occurrences count and are treated separately.**
Policy 4	All sets of grades and grade points count for GPA calculation. The credits from the last (most recent) occurrence are counted by Degree Works. The last occurrence is applied by Degree Works and all other occurrences go to "insufficient". **Most recent apply to rules – others count in GPA calculation.**
Policy 5	All occurrences of the class should be listed on the Degree Works audit where they could apply (i.e. all the occurrences stay grouped together by Degree Works). All sets of grades and grade points are used in the GPA calculation, but only credits for the occurrence with the best grade are counted by Degree Works. The best grade is used by Degree Works when checking MINGRADE. **All appear together on a rule and all count in the GPA calculation.**
Policy 6	All occurrences with Repeat Policy 6 are applied by the Auditor wherever they fit or in the "fallthrough" section of the audit. Other occurrences of this course that are not tagged with policy 6 (should be tagged with 0) are forced to "insufficient". Those in insufficient do not count in GPA calculations. **All with policy 6 apply to rules – all with policy 0 do not count in GPA calculations.**
Policy 7	If the class is not found in this table the repeat limit is assumed to be 1. The class with the best grade is kept. If multiple classes have the same grade then the most recently taken class is kept – but a completed class is kept over an in-progress class. If a class is found in this table for the term range specified then the repeat limit is obeyed. Classes that are not kept are marked with an "insufficient reason" of "WG" (Worst Grade). These WG classes are

	moved from the insufficient section to the over-the-limit (Not Counted) section by the auditor but only after the auditor has calculated overall the major GPAs based on what was in the insufficient section. The fact that these insufficient classes appear in the over-the-limit section of the audit is a display issue only that is part of this policy.
Policy B	Banner only. If the SHRTCKN_REPEAT_COURSE_IND is equal to an “E” and the Excluded class is NOT skipped the following rule shall apply: the rad_credits_earn, rad_gpa_credits and rad_grade_points will be set to “000000”, the insuff-flag will be set to Y and the repeat-ptr and repeat-plcy will be blanked out. This allows these special classes to be displayed in the insufficient section of the report, but with no impact to the credits earned or GPA. If the SHRTCKN_REPEAT_COURSE_IND is equal to "A" and the Averaged class is NOT skipped the following rule shall apply: the insuff-flag will be set to Y and the repeat-ptr and repeat-plcy will be blanked out. This allows these repeated classes to be displayed in the insufficient section of the report, but they will still impact the GPA. If the SHRTCKN_REPEAT_COURSE_IND is equal to "I" the class will apply to rules as a normal class. The repeat-ptr and repeat-plcy will be blanked out for these classes.

SOC Report Format

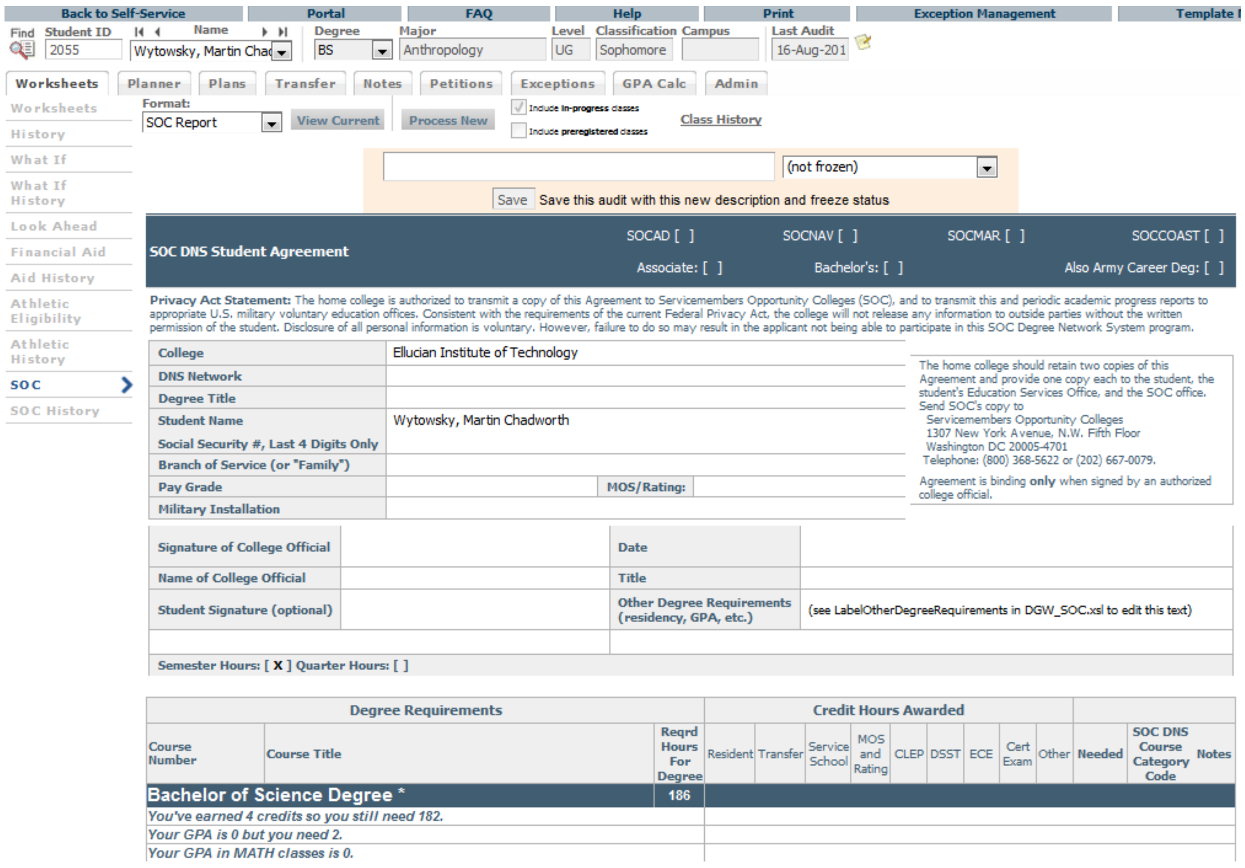
Degree Works has the ability to generate SOC (Servicemembers Opportunity College) DNS Student Agreements. The Degree Works SOC report is an audit worksheet displaying the results in a format accepted by the SOC office.

For more information about this agreement, refer to the official SOC Degree Network System Handbook provided by the Servicemembers Opportunity College office.

Degree Works Dashboard

The Worksheets tab on the Degree Works dashboard has two SOC menu items: SOC and SOC History. Access to these menu items as well as functions under these items is controlled by SHP security keys.

The SOC menu allows a user to display the SOC Report format, and a diagnostics report. New audits can be generated, and saved. This menu only allows the most recent audit to be displayed.



Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit
2055	Wytowsky, Martin Cha	BS	Anthropology	UG	Sophomore		16-Aug-201

Format: SOC Report

✓ Include in-progress classes

Include preregistered classes

(not frozen)

Save this audit with this new description and freeze status

SOC DNS Student Agreement SOCAD [ ] SOCNAV [ ] SOCMAR [ ] SOCCOAST [ ]
Associate: [ ] Bachelor's: [ ] Also Army Career Deg: [ ]

Privacy Act Statement: The home college is authorized to transmit a copy of this Agreement to Servicemembers Opportunity Colleges (SOC), and to transmit this and periodic academic progress reports to appropriate U.S. military voluntary education offices. Consistent with the requirements of the current Federal Privacy Act, the college will not release any information to outside parties without the written permission of the student. Disclosure of all personal information is voluntary. However, failure to do so may result in the applicant not being able to participate in this SOC Degree Network System program.

College	Ellucian Institute of Technology		
DNS Network			
Degree Title			
Student Name	Wytowsky, Martin Chadworth		
Social Security #, Last 4 Digits Only			
Branch of Service (or "Family")			
Pay Grade		**MOS/Rating:**	
Military Installation			

The home college should retain two copies of this Agreement and provide one copy each to the student, the student's Education Services Office, and the SOC office. Send SOC's copy to
Servicemembers Opportunity Colleges
1307 New York Avenue, N.W. Fifth Floor
Washington DC 20005-4701
Telephone: (800) 368-5622 or (202) 667-0079.

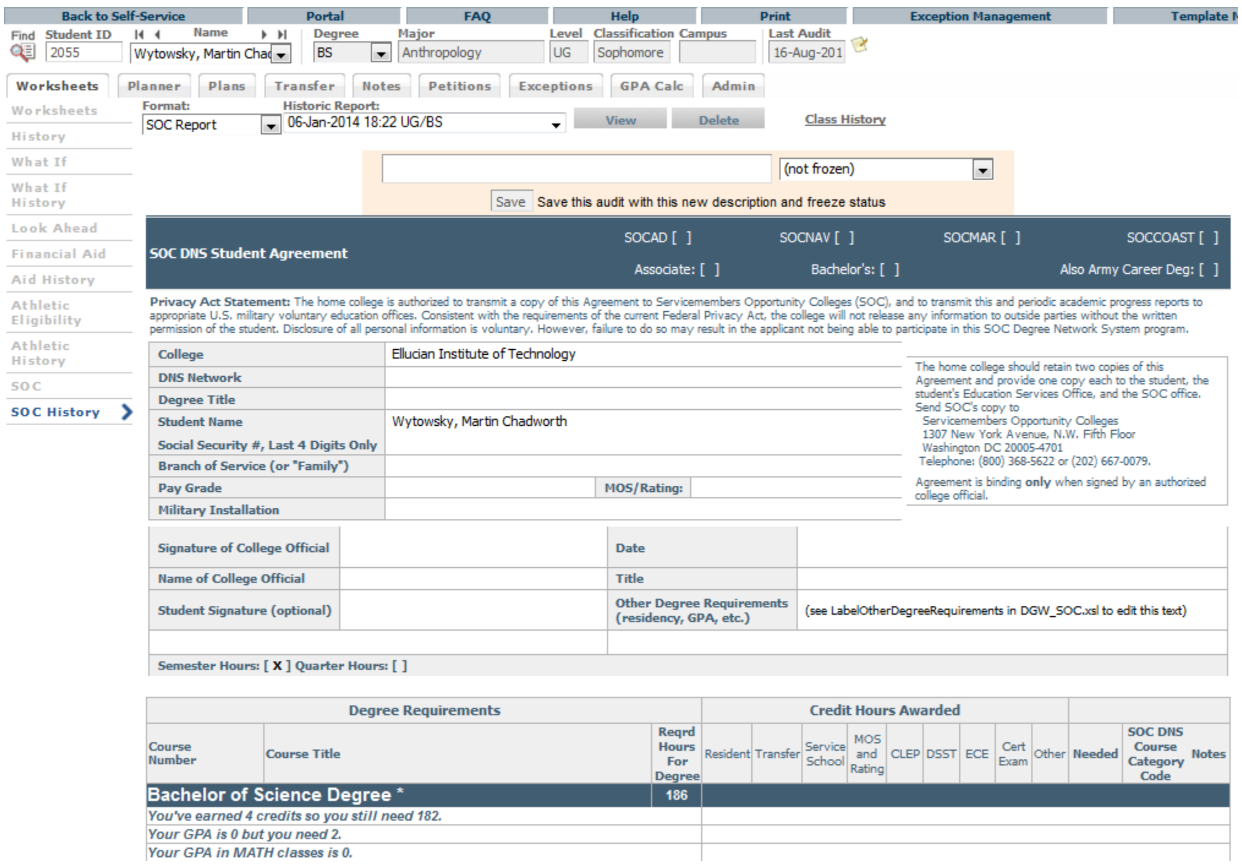
Agreement is binding **only** when signed by an authorized college official.

Signature of College Official		**Date**	
Name of College Official		**Title**	
Student Signature (optional)		**Other Degree Requirements (residency, GPA, etc.)**	(see LabelOtherDegreeRequirements in DGW_SOC.xsl to edit this text)

Semester Hours: [X] Quarter Hours: [ ]

Degree Requirements			Credit Hours Awarded											
Course Number	Course Title	Reqrd Hours For Degree	Resident	Transfer	Service School	MOS and Rating	CLEP	DSST	ECE	Cert Exam	Other	Needed	SOC DNS Course Category Code	Notes
Bachelor of Science Degree *		186												
You've earned 4 credits so you still need 182.														
Your GPA is 0 but you need 2.														
Your GPA in MATH classes is 0.														

The SOC History menu allows the user to view previous versions of the SOC report, the same as the “history menu” option of the other audit types.



Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit
2055	Wytowsky, Martin Chad	BS	Anthropology	UG	Sophomore		16-Aug-201

Format: SOC Report
Historic Report: 06-Jan-2014 18:22 UG/BS

(not frozen)

Save this audit with this new description and freeze status

SOC DNS Student Agreement

SOCAD [ ] SOCNAV [ ] SOCMAR [ ] SOCCOAST [ ]

Associate: [ ] Bachelor's: [ ] Also Army Career Deg: [ ]

Privacy Act Statement: The home college is authorized to transmit a copy of this Agreement to Servicemembers Opportunity Colleges (SOC), and to transmit this and periodic academic progress reports to appropriate U.S. military voluntary education offices. Consistent with the requirements of the current Federal Privacy Act, the college will not release any information to outside parties without the written permission of the student. Disclosure of all personal information is voluntary. However, failure to do so may result in the applicant not being able to participate in this SOC Degree Network System program.

College	Ellucian Institute of Technology		
DNS Network			
Degree Title			
Student Name	Wytowsky, Martin Chadworth		
Social Security #, Last 4 Digits Only			
Branch of Service (or "Family")			
Pay Grade		MOS/Rating:	
Military Installation			

The home college should retain two copies of this Agreement and provide one copy each to the student, the student's Education Services Office, and the SOC office. Send SOC's copy to
Servicemembers Opportunity Colleges
1307 New York Avenue, N.W. Fifth Floor
Washington DC 20005-4701
Telephone: (800) 368-5622 or (202) 667-0079.

Agreement is binding **only** when signed by an authorized college official.

Signature of College Official		Date	
Name of College Official		Title	
Student Signature (optional)		Other Degree Requirements (residency, GPA, etc.)	(see LabelOtherDegreeRequirements in DGW_SOC.xsl to edit this text)

Semester Hours: [X] Quarter Hours: [ ]

Degree Requirements			Credit Hours Awarded											
Course Number	Course Title	Reqrd Hours For Degree	Resident	Transfer	Service School	MOS and Rating	CLEP	DSST	ECE	Cert Exam	Other	Needed	SOC DNS Course Category Code	Notes
Bachelor of Science Degree *		186												
You've earned 4 credits so you still need 182.														
Your GPA is 0 but you need 2.														
Your GPA in MATH classes is 0.														

Sources of Data

To display and print a DNS Student Agreement (Degree Works SOC report) that is usable for the institution and also acceptable to submit to the SOC office, a review of the current Degree Works environment may be needed. Specifically:

1) Location and extraction of SIS data needs to occur so that header information can be completed
2) Attributes need to be assigned to courses so that credits are assigned to the appropriate location in the tabular section of the report
3) Scribed requirements may need additional modification so that the audit section display is appropriate
4) Total credits required for the degree needs to be defined

Report Header Data

In order to fully utilize SOC reports, this informational data must be stored in Degree Works. This is data that is strictly for display-purposes only. In Degree Works, the standard location to store this type of data is the rad_report_dtl. For SOC reports, these are the rad_report_dtl records that are used:

Report-dtl code	Description
SOC**AGREEMENT**	SOCAD, SOCNAV, SOCMAR, SOCCOAST
SOC**DEGTYPE**	Type of degree (associates, bachelors, also army career)
SOC**DNSNETWRK**	Appropriate Degree Network System network
SOC**DEGRTITLE**	Degree Title, as listed in catalog
SOC**SSN**	Social Security Number, last 4 digits only
SOC**BRANCH**	Branch of Service (Navy, Army, etc.)
SOC**MILRANK**	Pay Grade (Military Rank)
SOC**MOS**	Military Occupational Specialties
SOC**MILINSTAL**	Military Installation where servicemember is assigned
SOC**DATE**	Date of signatory approval
SOC**OFFNAME**	Name of college official
SOC**OFFTITLE**	Title of college official

If you are currently storing this data in your SIS, you will need to setup the extraction process to pull the data into Degree Works:

If you are a Banner site you will setup UCX-BAN080 records
If you are a Colleague site you will setup report.client.properties

If you are not currently storing this data in your SIS and want it to appear on the Degree Works worksheet, you will need to start doing so to allow the extraction of these elements.

Each of these report header items is further explained in the screenshot illustrations sections that follow.

Credit Hour Awarded Data

An important aspect of the SOC report is the “SOC Type”, also known as the credit source. This information is used to identify the category in which credits are counted in the “Credit Hours Awarded” section. This is the military training, college-level examinations, and other non-traditional credit sources that apply toward degree requirements. It is stored on the rad_attr_dtl record tied to the rad_transfer_dtl. Banner schools need to record an attribute for each transfer class for the appropriate SOC type. Here are the known SOC Types and the attribute values you need to add to SHRTATT:

Attribute	Description
SVSC	“Service School,” also known as Military Training Courses
MOS	“Military Occupational Specialties”
CLEP	“College-Level Examination Program,” also known as General and Subject Exams
DSST	“DANTES Standardized Subject Tests”
ECE	“Excelsior College Examinations”
CRTX	Certification Examination Credit, such as FAA certification
SOCO	Other earned credits, such as internships, portfolio assignments, other non-military experience

Scribe changes needed – “SOC” RuleTags

Effort has been made to make scribing changes minimal to produce the SOC report format. Scribe changes have been limited to the addition of RuleTags to help organize the output appropriately, and are needed only if it is determined that the output needs modification. There are 4 new RuleTags that have been added to assist with this.

SOC_CATALOG - SOC Course Catalog Number

Each Degree Works requirement may contain a course that has a SOC-standard course associated with it. For example, MATH 101 at your institution maps to the SOC-standard MH001B. In order to populate the "SOC Course Cat #" column in the SOC report, you must update your Scribe blocks with the information. This is done using the "RuleTag" syntax using "**SOC_CATALOG**" as the RuleTag description. For example:

```
3 Credits in MATH 101
     RuleTag SOC_CATALOG=MH001B
     Label "College Algebra";
```

SOC_LABEL - SOC Course Title

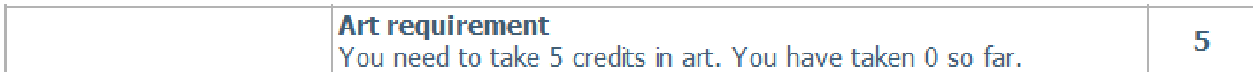
The label on each requirement is used for the Course Title column. However, you may specify alternative text to be used in the SOC report by using the **SOC_LABEL** rule tag. In most every case you will surely find that the label you have in place is perfectly fine to use on the SOC report but you always have the option of changing it without affecting the academic audits.

```
3 Credits in EVA 101
     RuleTag SOC_LABEL="Environmental Foundations"
     Label "This label will not appear on the SOC report";
```

SOC_ADVICE - SOC advice

If ProxyAdvice is found on the rule then it will be placed underneath the label in the Course Title field. When ProxyAdvice is found the Course Number field is left empty regardless of what courses are listed on the requirement.

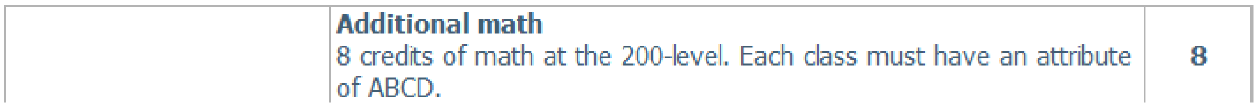
```
5 Credits in ART 2@
     ProxyAdvice "You need to take 5 credits in art. "
     ProxyAdvice "You have taken <APPLIED> so far."
      Label "Art requirement";
```



	Art requirement You need to take 5 credits in art. You have taken 0 so far.	5

But you can override the ProxyAdvice by using **SOC_ADVICE**. Actually, even if the rule does not have ProxyAdvice you may still add SOC_ADVICE to be displayed below the label in the Course Title field. If the SOC_ADVICE is found it will be used and any ProxyAdvice will be ignored. As with ProxyAdvice, when SOC_ADVICE is found the Course Number field is left empty regardless of what courses are listed on the requirement.

```
18 Credits in MATH 200:299 (With Attribute=ABCD)
     RuleTag SOC_CREDITS=8
     RuleTag SOC_ADVICE="8 credits of math at the 200-level. "
     RuleTag SOC_ADVICE="Each class must have an attribute of ABCD."
     ProxyAdvice "You still need <NEEDED> credits"
      Label "Additional math";
```



	Additional math 8 credits of math at the 200-level. Each class must have an attribute of ABCD.	8

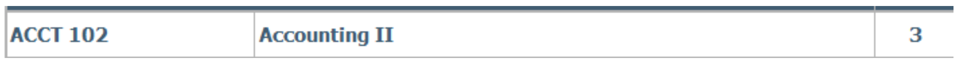
SOC_CREDITS - SOC Requirement Credits

The Hours Required field is a critical piece of information for the SOC report. However, it may not always be obvious how many credits each requirement is worth. In these cases you may need to add the **SOC_CREDITS** rule tag to your requirements.

The example that follows details how the report determines the Hours Required value and when you need to use **SOC_CREDITS**.

Example 1
The credits on the rule are specified so they are placed in the Required Hours column.

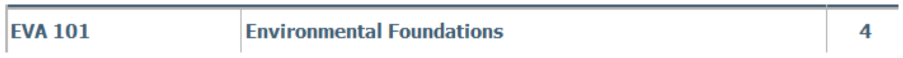
```
3 Credits in ACCT 102
     Label "Accounting II";
```



ACCT 102	Accounting II	3

Example 2
The credits were not specified but since it is a single course listed we take the credits from the course (looked up on the rad_course_mst). Here EVA 101 is a 4 credit course.

```
1 Class in EVA 101
     Label "Environmental Foundations ";
```

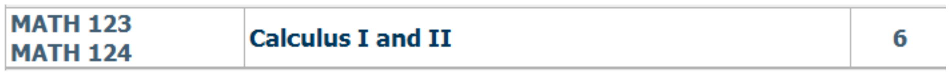


EVA 101	Environmental Foundations	4

Example 3

Here again the credits were not specified. However, since the rule requires two classes and because two classes were listed we simply add up the credits for both courses – again taken from the rad_course_mst.

```
2 Classes in MATH 123 + 124
     Label "Calculus I and II";
```

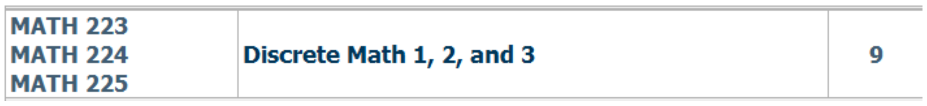


MATH 123 MATH 124	Calculus I and II	6

Example 4

This same determination of credits is made as with the previous example. Here it is a comma-list instead of a plus-list of courses but since three classes are required and since we have three classes listed we can still add up the credits for each.

```
3 Classes in MATH 223, 224, 225
     Label "Discrete Math 1, 2, and 3";
```

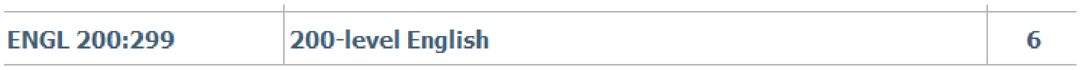


MATH 223 MATH 224 MATH 225	Discrete Math 1, 2, and 3	9

Example 5

Here the credits were not specified and we have a range of courses listed. We cannot determine the credits needed by examining the list of courses and thus you must use SOC_CREDITS to indicate the number of credits required. This would also be the case if this rule was scribed using a wildcard.

```
2 Classes in ENGL 200:299
     RuleTag SOC_CREDITS=6
      Label "200-level English";
```



ENGL 200:299	200-level English	6

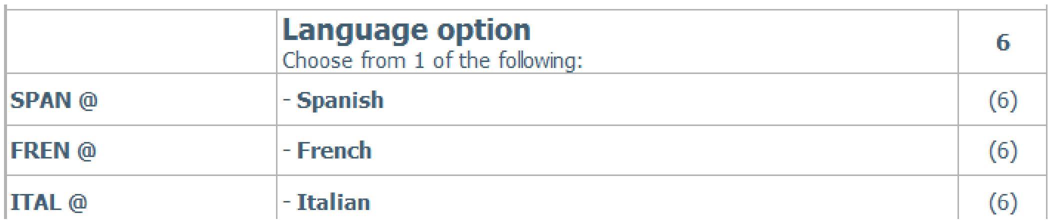
Here are other cases where the SOC_CREDITS is needed:

```
3 Classes in MATH 104, 109, 118, CHEM 2@, 314, 418, BIOL 198, 203

1 Class in ART 3@
```

Example 6

On very simple group rules the number of credits for each requirement is the same – as shown below. However, on many group rules the number of credits on each option is different and thus it is not so easy to determine how many credits actually are required by the group. For this reason you must add SOC_CREDITS to each group rule. The credits for each of the rules within the group are shown in the report in parentheses as an FYI of sorts.

```
1 Group in
  (6 Credits in SPAN @ Label "Spanish") or
  (6 Credits in FREN @ Label "French") or
  (6 Credits in ITAL @ Label "Italian")
  RuleTag SOC_CREDITS=6
 Label "Language option";
```

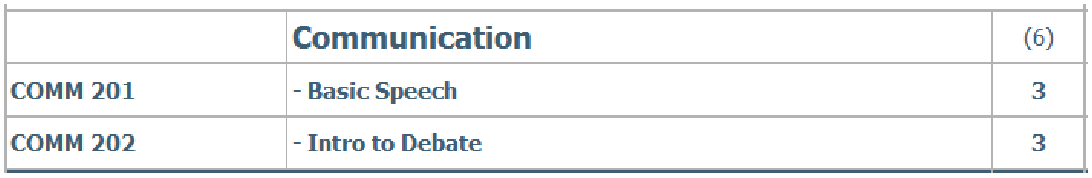


	Language option Choose from 1 of the following:	6
SPAN @	- Spanish	(6)
FREN @	- French	(6)
ITAL @	- Italian	(6)

Example 7

The credits for the entire subset are actually not required but if you so choose you may add SOC_CREDITS to each of your subsets. This number will be shown as an FYI with the number displayed in parentheses. To limit the number of number that appear on the report it might be best for you not to add SOC_CREDITS to subsets. You should at least test the report with it on your subsets to see what you think and then add them only if you think it is helpful.

```
beginsub
  3 Credits in COMM 201
     Label "Basic Speech";
  3 Credits in COMM 202
     Label "Intro to Debate";
endsub
  RuleTag SOC_CREDITS=6
  Label "Communication";
```



	Communication	(6)
COMM 201	**- Basic Speech**	**3**
COMM 202	**- Intro to Debate**	**3**

Total Credits Required

The total credits required in a SOC report is identical to the total credits required in a normal Degree Works report. It uses the starting block's "Credits Required", Scribed as a header qualifier. If the starting block does not contain a "Credits" qualifier it will result in an inaccurate SOC report.

You can scribe using Pseudo if needed. This will give the report the information that is needed but will not need to be met for the student to satisfy the degree requirements. For example:

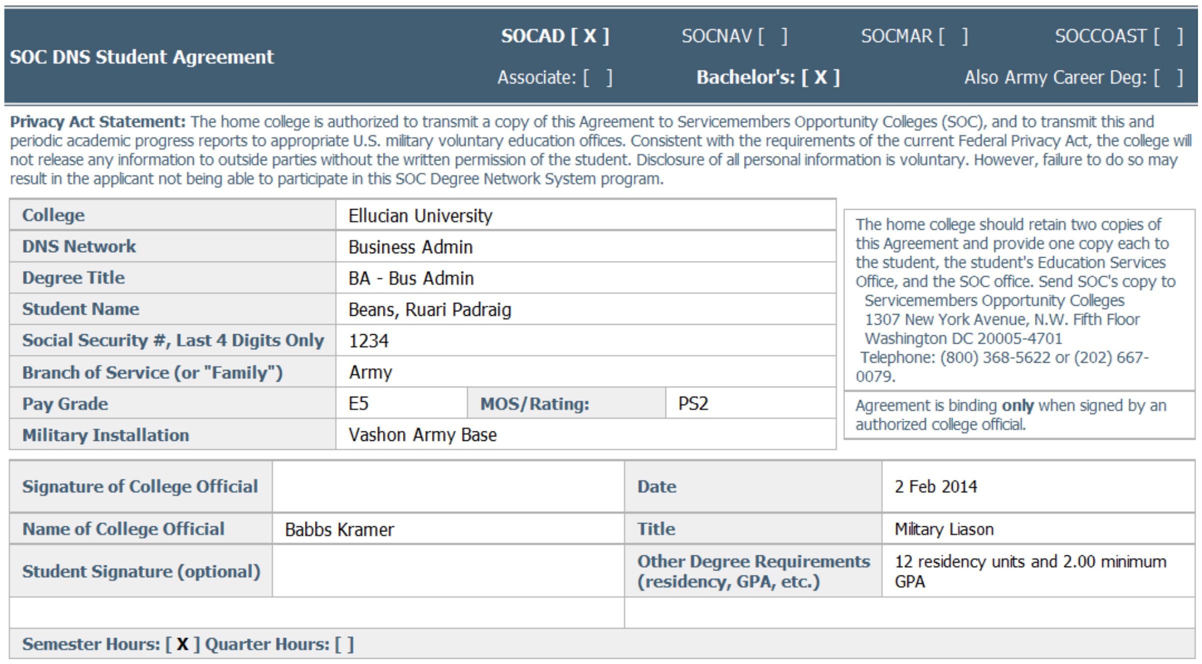
```
90 Credits Pseudo
```

Screenshot Illustrations – SOC DNS Student Agreement

This section will show by example how a Degree Works SOC report is constructed and where the source of the data or configuration that influences the output is maintained. For this purpose, the report is separated organizationally into the header, body, and footer.

Header

Here is an example of a complete SOC report header (Form edition December 2010)



SOC DNS Student Agreement **SOCAD [X]** SOCNAV [ ] SOCMAR [ ] SOCCOAST [ ]
Associate: [ ] **Bachelor's: [X]** Also Army Career Deg: [ ]

Privacy Act Statement: The home college is authorized to transmit a copy of this Agreement to Servicemembers Opportunity Colleges (SOC), and to transmit this and periodic academic progress reports to appropriate U.S. military voluntary education offices. Consistent with the requirements of the current Federal Privacy Act, the college will not release any information to outside parties without the written permission of the student. Disclosure of all personal information is voluntary. However, failure to do so may result in the applicant not being able to participate in this SOC Degree Network System program.

College	Ellucian University		
DNS Network	Business Admin		
Degree Title	BA - Bus Admin		
Student Name	Beans, Ruari Padraig		
Social Security #, Last 4 Digits Only	1234		
Branch of Service (or "Family")	Army		
Pay Grade	E5	**MOS/Rating:**	PS2
Military Installation	Vashon Army Base		

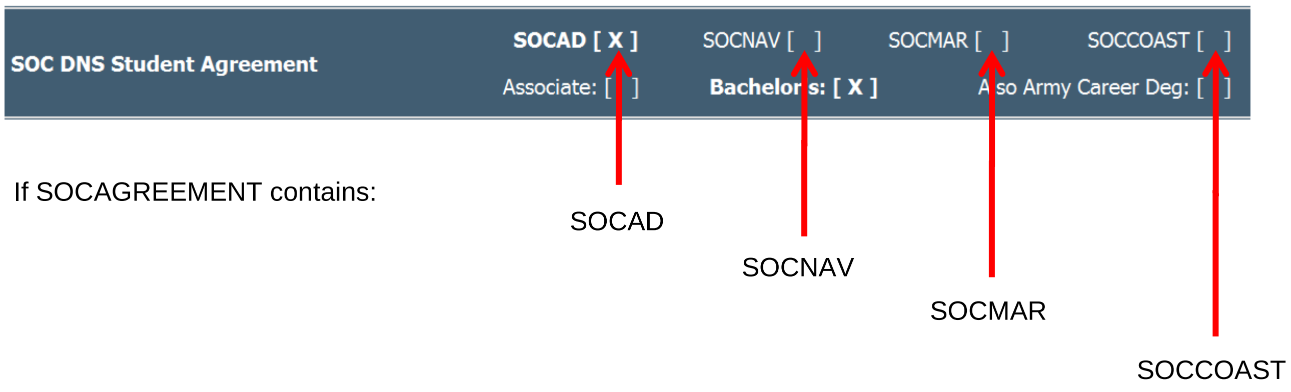
The home college should retain two copies of this Agreement and provide one copy each to the student, the student's Education Services Office, and the SOC office. Send SOC's copy to
Servicemembers Opportunity Colleges
1307 New York Avenue, N.W. Fifth Floor
Washington DC 20005-4701
Telephone: (800) 368-5622 or (202) 667-0079.

Agreement is binding **only** when signed by an authorized college official.

Signature of College Official		**Date**	2 Feb 2014
Name of College Official	Babbs Kramer	**Title**	Military Liason
Student Signature (optional)		**Other Degree Requirements (residency, GPA, etc.)**	12 residency units and 2.00 minimum GPA
Semester Hours: [X] Quarter Hours: [ ]			

SOC Degree Network System Program – SOCAGREEMENT

Only one of the boxes will be marked with an "X". This is driven by the text contained in the value of the rad_report_dtl SOCAGREEMENT record.



For example, if the SOCAGREEMENT contains SOCAD-2, SOCAD, or SOCAD4 (anything with "SOCAD") then an X is placed next to SOCAD.

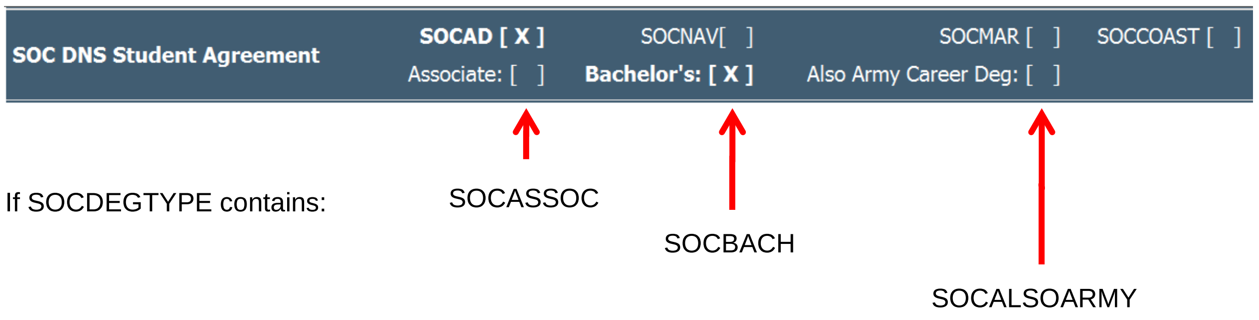
SOCAGREEEMENT is optional. If it is bridged it will be used as the doc states.

If no SOCAGREEMENT record is found, then SOCBRANCH will be used to determine the agreement:

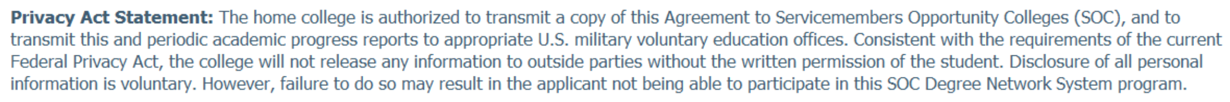
If "Army" is in the branch then an X is placed next to SOCAD.
If "Coast" is in the branch then an X is placed next to SOCCOAST.
If "Navy" is in the branch then an X is placed next to SOCNAV.
If "Marine" is in the branch then an X is placed next to SOCMAR.

Degree Type - SOCDEGTYPE

Only one of the boxes will be marked with an "X". This is driven by the text contained in the value of the rad_report_dtl SOCDEGTYPE record.

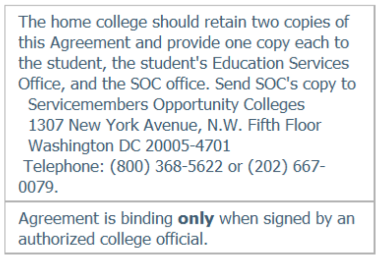


Privacy Statement


Privacy Act Statement: The home college is authorized to transmit a copy of this Agreement to Servicemembers Opportunity Colleges (SOC), and to transmit this and periodic academic progress reports to appropriate U.S. military voluntary education offices. Consistent with the requirements of the current Federal Privacy Act, the college will not release any information to outside parties without the written permission of the student. Disclosure of all personal information is voluntary. However, failure to do so may result in the applicant not being able to participate in this SOC Degree Network System program.

This text can be changed by altering the LabelPrivacyStatement variable in DGW_SOC.xsl – but you should contact your SOC office before making changes to this text as it currently represents the approved format.

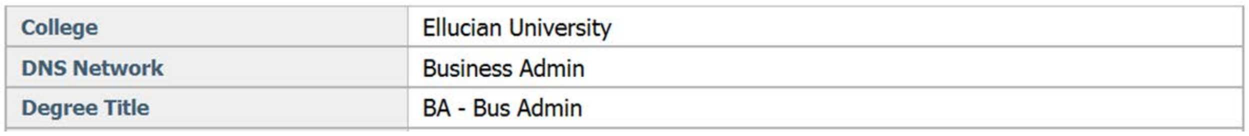
Form Text


The home college should retain two copies of this Agreement and provide one copy each to the student, the student's Education Services Office, and the SOC office. Send SOC's copy to
Servicemembers Opportunity Colleges
1307 New York Avenue, N.W. Fifth Floor
Washington DC 20005-4701
Telephone: (800) 368-5622 or (202) 667-0079.

Agreement is binding **only** when signed by an authorized college official.

This text can be changed by altering the LabelCopies and LabelAgreement variables in DGW_SOC.xsl – but you should contact your SOC office before making changes to this text as it currently represents the approved format.

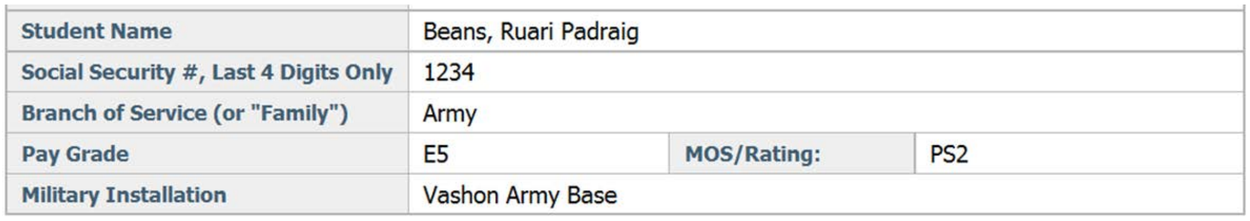
Home College Information



College	Ellucian University
DNS Network	Business Admin
Degree Title	BA - Bus Admin

College displays your school's name (SRNDWAUDITTITLE variable)
DNS Network displays the **SOCDNSNETWRK** value from the rad_report_dtl
Degree Title displays the **SOCDEGRTITLE** value from the rad_report_dtl

Pertinent Demographic Information



Student Name	Beans, Ruari Padraig		
Social Security #, Last 4 Digits Only	1234		
Branch of Service (or "Family")	Army		
Pay Grade	E5	**MOS/Rating:**	PS2
Military Installation	Vashon Army Base		

Student Name displays the student's name
Social Security # displays the **SOCSSN** value from the rad_report_dtl – whatever is bridged. Please be sure to bridge only the last four characters
Branch of Service displays the **SOCBRANCH** value from the rad_report_dtl
Pay Grade displays the **SOCMILRANK** value from the rad_report_dtl
MOS/Rating displays the **SOCMOS** value from the rad_report_dtl

Military Installation displays the **SOCMILINSTAL** value from the rad_report_dtl

Approving Authority Information

Date displays the **SOCDATE** value from the rad_report_dtl. If this item is not bridged, then the date the audit was generated will be displayed.
Name of College Official displays the **SOCOFFNAME** value from the rad_report_dtl
Title displays the **SOCOFFTITLE** value from the rad_report_dtl
Other Degree Requirements displays the text in **LabelOtherDegreeRequirements** in the stylesheet. This is changed by each school as needed during implementation.

The two **Signature** fields are left blank.

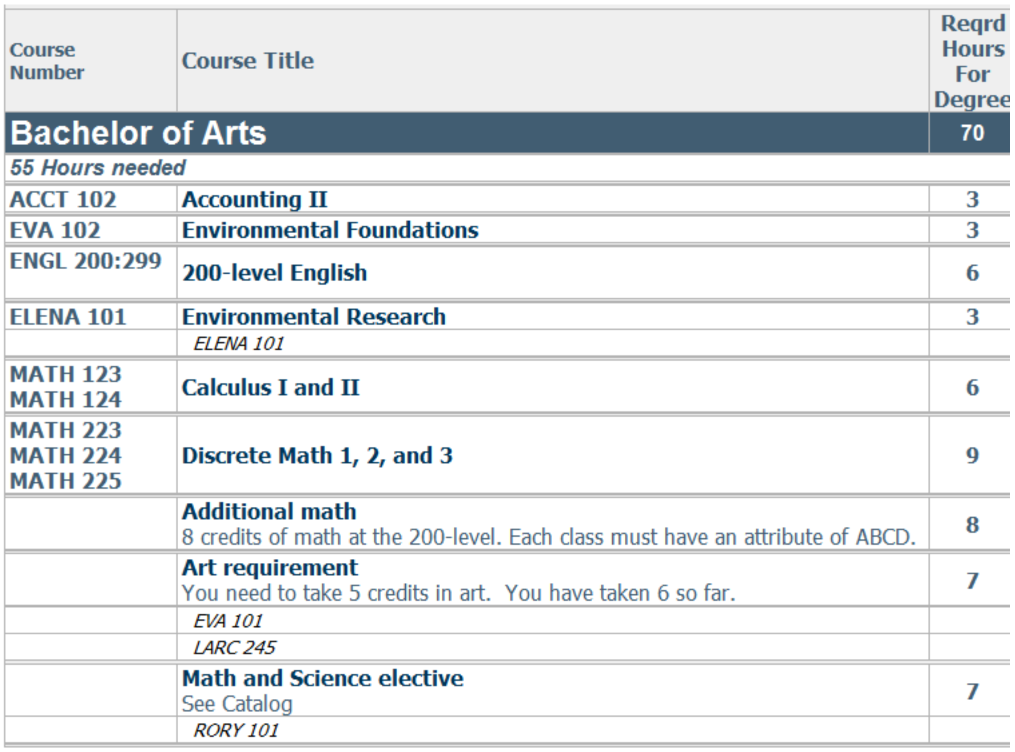
Miscellaneous

This is controlled by the **LabelCalendar** variable in the DGW_SOC.xsl stylesheet. This is changed by each school as needed during implementation.

Body

Degree Requirements

Here are some examples showing how the **Course Number**, **Course Title** and **Reqrd Hours for Degree** fields are populated based on what is scribed.



Course Number	Course Title	Reqrd Hours For Degree
Bachelor of Arts		**70**
55 Hours needed		
ACCT 102	Accounting II	3
EVA 102	Environmental Foundations	3
ENGL 200:299	200-level English	6
ELENA 101	Environmental Research	3
	ELENA 101	
MATH 123 MATH 124	Calculus I and II	6
MATH 223 MATH 224 MATH 225	Discrete Math 1, 2, and 3	9
	Additional math 8 credits of math at the 200-level. Each class must have an attribute of ABCD.	8
	Art requirement You need to take 5 credits in art. You have taken 6 so far.	7
	EVA 101	
	LARC 245	
	Math and Science elective See Catalog	7
	RORY 101	

Accounting II

Only one course is listed and it appears in the Course Number column. The rule’s label is placed in the Course Title column.

The rule is scribed with credits and its value is placed in the Reqrd Hours column.

```
3 Credits in ACCT 102
  Label "Accounting II";
```

Environmental Foundations

Only one course is listed and it appears in the Course Number column. The SOC_LABEL is placed in the Course Title column.

The rule is scribed as 1 Class so the course’s credits are placed in the Reqrd Hours column.

```
1 Class in EVA 102
  RuleTag SOC_LABEL="Environmental Foundations"
  Label "This should not appear - see SOC_LABEL";
```

200-level English

A course range is listed and it appears in the Course Number column. The rule’s label is placed in the Course Title column.

The rule is scribed as 2 Classes and 2 classes were not listed so SOC_CREDITS is needed and its value is placed in the Reqrd Hours column.

```
2 Classes in ENGL 200:299
  RuleTag SOC_CREDITS=6
  Label "200-level English";
```

Environmental Research

Only one course is listed and it appears in the Course Number column. The SOC_LABEL is placed in the Course Title column.

The rule is scribed with a credit range and its low value is placed in the Reqrd Hours column.
In addition, the student took ELENA 101 and this class applied to the rule so it appears in black beneath the requirement.

```
3:5 Credits in ELENA 101
  RuleTag SOC_CATALOG="RD987B"
  RuleTag SOC_LABEL="Environmental Research"
  Label "This should not appear - see SOC_LABEL";
```

Calculus I and II

Two courses are listed and both appear in the Course Number column. The rule’s label is placed in the Course Title column.

The rule is scribed with 2 Classes and 2 classes are listed so the total credits for both courses are placed in the Reqrd Hours column.

```
2 Classes in MATH 123 + 124
```

```
  Label "Calculus I and II";
```

Discrete Math 1, 2, and 3

Three courses are listed and all appear in the Course Number column. The rule’s label is placed in the Course Title column.

The rule is scribed with 3 Classes and 3 classes are listed so the total credits for all courses are placed in the Reqrd Hours column.

```
3 Classes in MATH 223, 224, 225
  Label "Discrete Math 1, 2, and 3";
```

Additional Math

SOC_ADVICE was found so no courses are listed in the Course Number column. The rule’s label is placed in the Course Title column.

The rule’s is scribed with 18 Credits but the SOC_CREDITS are placed in the Reqrd Hours column.

```
18 Credits in MATH 200:299 (With Attribute=ABCD)
  RuleTag SOC_CREDITS=8
  RuleTag SOC_ADVICE="8 credits of math at the 200-level. "
  RuleTag SOC_ADVICE="Each class must have an attribute of ABCD."
  ProxyAdvice "You still need <NEEDED> credits"
  Label "Additional math";
```

Art Requirement

ProxyAdvice was found so no courses are listed in the Course Number column. The rule’s label is placed in the Course Title column.

The rule’s 7 credits are placed in the Reqrd Hours column. Additionally, the student took EVA 101 and LARC 245 and both apply to this rule and appear beneath the requirement.

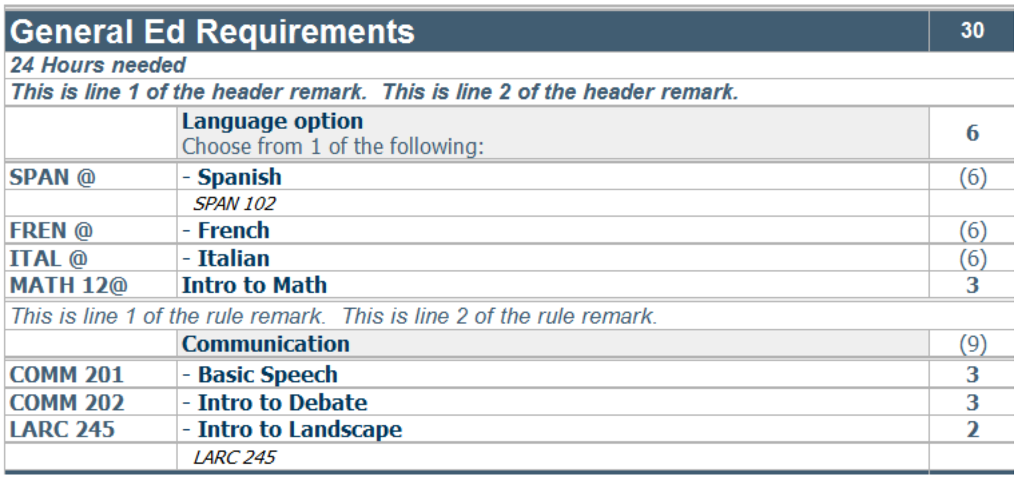
```
7 Credits in ART 2@, EVA 101, LARC 245
  ProxyAdvice "You need to take 5 credits in art. "
  ProxyAdvice "You have taken <APPLIED> so far."
  Label "Art requirement";
```

Math and Science Elective

More than 5 courses (not hidden) were found so no courses are listed in the Course Number column. This maximum limit is controlled by the **vMaxCoursesToShow** variable in the stylesheet. The rule’s label is placed in the Course Title column.

The rule’s 7 credits are placed in the Reqrd Hours column. Additionally, the student took RORY 101 and it applies to this rule and appears beneath the requirement.

```
7 Credits in MATH 101, 102, 103, 104, 105, 106, 107, RORY 101
  RuleTag SOC_CATALOG=AB123X
  Label "Math and Science elective";
```



General Ed Requirements		30
24 Hours needed		
This is line 1 of the header remark. This is line 2 of the header remark.		
	Language option Choose from 1 of the following:	6
SPAN @	- Spanish	(6)
	SPAN 102	
FREN @	- French	(6)
ITAL @	- Italian	(6)
MATH 12@	Intro to Math	3
This is line 1 of the rule remark. This is line 2 of the rule remark.		
	Communication	(9)
COMM 201	- Basic Speech	3
COMM 202	- Intro to Debate	3
LARC 245	- Intro to Landscape	2
	LARC 245	

24 Hours needed

The **header advice** appear at the top of the block if the **vShowHeaderAdvice** setting in the stylesheet is set to Y.

This is line 1 of the header remark. This is line 2 of the header remark.

The **header remarks** appear beneath the header advice the top of the block if the **vShowHeaderRemarks** setting in the stylesheet is set to Y.

Language Option

This is a group rule so the group's label appears with a grey background and "Choose from 1 of the following" is placed beneath the label.

It is important to know how many credits the group is worth so the **SOC_CREDITS** rule tag should be placed on the rule. In this example each rule is worth the same amount of credits but in many groups that is not the case.

Each of the rules in the group is processed normally with the following exceptions:

- A hyphen is placed before each title to show it belongs to the group.
- The credits for each rule are placed within parentheses as an FYI; the credits on the group is what is used in the report.

```
1 Group in
  (6 Credits in SPAN @ Label "Spanish") or
  (6 Credits in FREN @ Label "French") or
  (6 Credits in ITAL @ Label "Italian")
  RuleTag SOC_CREDITS=6
    Label "Language option";
```

Intro to Math

Only one course is listed and it appears in the Course Number column.

The rule's label is placed in the Course Title column. The rule is scribed with credits and its value is placed in the Reqrd Hours column.

```
3 Credits in MATH 12@
```

```
  Label "Intro to Math ";
```

This is line 1 of the rule remark. This is line 2 of the rule remark.

The **rule remarks** appear above the rule to which they are attached if the **vShowRuleRemarks** setting in the stylesheet is set to Y.

Communication

This is a subset rule so the subset's label appears with a grey background. The SOC_CREDITS on the subset are optional but if found the value is placed in parentheses as an FYI.

Each of the rules in the subset is processed normally but each shows with a hyphen to show it belongs to the subset.

```
beginsub
  3 Credits in COMM 201 Label "Basic Speech";
  3 Credits in COMM 202 Label "Intro to Debate";
  2 Credits in LARC 245 Label "Intro to Landscape";
endsub
  RuleTag SOC_CREDITS=9
  Label "Communication";
Remark "This is line 1 of the rule remark. "
Remark "This is line 2 of the rule remark. "
```

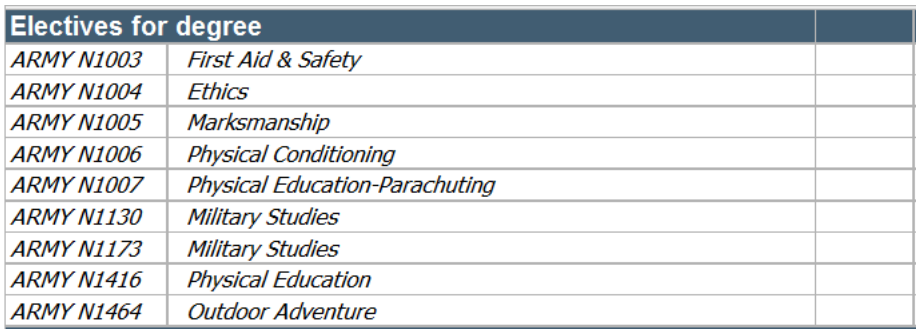
The report does show the **Course**, **Group**, and **Subset** rules from the audit.

The **RuleComplete** and **RuleIncomplete** rules appear based on the **vShowRULECOMPLETERules** and **vShowRULEINCOMPLETERules** flags in the stylesheet.

The **Noncourse**, **Block** and **Blocktype** rules are never shown.

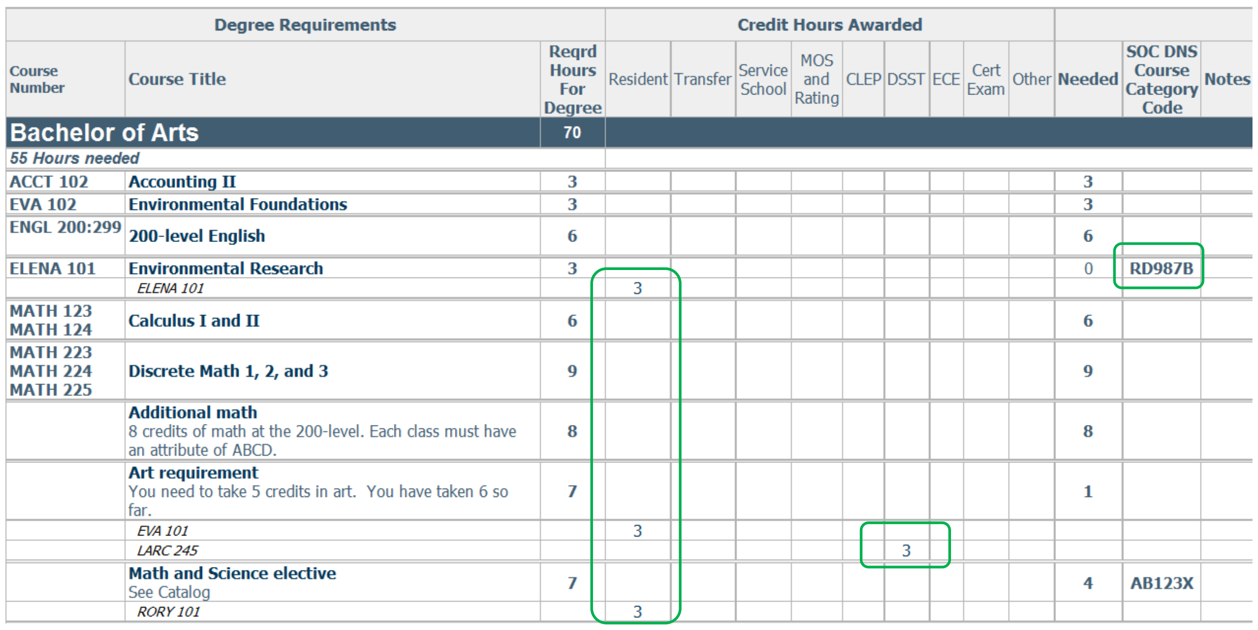
Electives for degree

Classes appearing in the fall-through section of the audit appear in the **Electives for degree** section. Since these are not associated with any specific requirement the class's course number and discipline are placed in the Course Number column and the class's title is placed in the Course Title column. The Reqd Hours columns is always left blank since these classes are not requirements.



Electives for degree		
ARMY N1003	*First Aid & Safety*	
ARMY N1004	*Ethics*	
ARMY N1005	*Marksmanship*	
ARMY N1006	*Physical Conditioning*	
ARMY N1007	*Physical Education-Parachuting*	
ARMY N1130	*Military Studies*	
ARMY N1173	*Military Studies*	
ARMY N1416	*Physical Education*	
ARMY N1464	*Outdoor Adventure*	

Credit Hours Awarded, Needed and Course Category



Degree Requirements			Credit Hours Awarded											
Course Number	Course Title	Reqrd Hours For Degree	Resident	Transfer	Service School	MOS and Rating	CLEP	DSST	ECE	Cert Exam	Other	Needed	SOC DNS Course Category Code	Notes
Bachelor of Arts		70												
55 Hours needed														
ACCT 102	**Accounting II**	3										3		
EVA 102	**Environmental Foundations**	3										3		
ENGL 200:299	**200-level English**	6										6		
ELENA 101	**Environmental Research**	3										0	RD987B	
	ELENA 101		3											
MATH 123 MATH 124	**Calculus I and II**	6										6		
MATH 223 MATH 224 MATH 225	**Discrete Math 1, 2, and 3**	9										9		
	Additional math 8 credits of math at the 200-level. Each class must have an attribute of ABCD.	8										8		
	Art requirement You need to take 5 credits in art. You have taken 6 so far.	7										1		
	EVA 101		3											
	LARC 245							3						
	Math and Science elective See Catalog	7										4	AB123X	
	RORY 101		3											

SOC DNS Course Category Code

RD987B appears in this column because the requirement was scribed with it as the SOC_CATALOG. It so happens that ELENA 101 is applying to this requirement but the value would have appeared in this column even if the class did not yet apply.

```
3:5 Credits in ELENA 101
  RuleTag SOC_CATALOG="RD987B"
  RuleTag SOC_LABEL="Environmental Research"
  Label "This should not appear - see SOC_LABEL";
```

Credit Hours Awarded

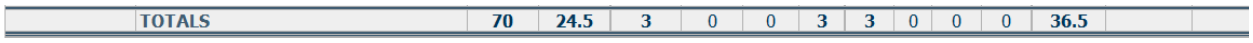
3 credits appear in the DSST column because LARC 245 has an Attribute value of DSST.

Resident

3 credits appear in the Resident column for these three classes because they were taken at the home institution.

Note: If a class from the rad_class_dtl happens to have one of the SOC attributes associated with it then the credits for the class will show up in both the Resident column and one of the other columns. It is assumed that the SOC attributes will only be associated with transfer records on the rad_transfer_dtl. For transfer records, the class's credits will only show up in the Transfer column if the class does not have any of the SOC attributes associated with it.

TOTALS



TOTALS	70	24.5	3	0	0	3	3	0	0	0	36.5	

The **70** appearing here is the total credits scribed in the degree block. This is not a total of the Reqd Hours summed together and thus will most likely not match the summation of the credits listed in the fields above.

The **24.5** credits is the total number of in-residence credits applied to rules added to those in the fall-through section.

The **3** credits is the total number of transfer credits applied to rules added to those in the fall-through section. These transfer credits do not include any of the transfer credits that have any of the SOC attributes; these credits are those transfer records that do not have any of the SOC attributes.

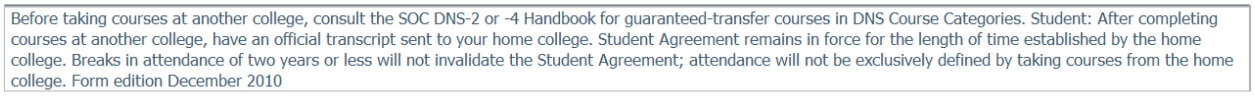
The next seven columns contain the sum of credits for each of the SOC attributes. The two 3 values you see here are the total number of CLEP and DSST credits.

The **36.5** credits is the number of credits still needed by the degree. This is not a summation of the fields in the Needed column above it. Instead this is simply the number of credits still needed to meet the total required by the degree.

Note: In this audit one of the DSST classes applies to two different requirements in two different blocks. The credits for the class are only counted once in this totals value however. This means that these totals fields for the SOC attributes, and also the Resident and Transfer values, may not be the summation of all of the credits in the fields above them; each class's credits is only counted once toward the total number.

Note: All of the credits in the TOTALS line except for the last number (Needed) will include the credits in the fall-through section even if the UCX-CFG020 DAP14 Fallthu Counts in Overall flag is set to N.

Footer



Before taking courses at another college, consult the SOC DNS-2 or -4 Handbook for guaranteed-transfer courses in DNS Course Categories. Student: After completing courses at another college, have an official transcript sent to your home college. Student Agreement remains in force for the length of time established by the home college. Breaks in attendance of two years or less will not invalidate the Student Agreement; attendance will not be exclusively defined by taking courses from the home college. Form edition December 2010

This text can be changed by altering the LabelFooterStatement variable in DGW_SOC.xsl – but you should contact your SOC office before making changes to this text as it currently represents the approved format.

Setup Summary

Shepherd security keys for SOC

SDSOCMIL	show SOC Tab
SDSOCRUN	allow processing new SOC audits
SDSOCHIS	show SOC History tab

SDSOCDEL	allow deletion of historic SOC audits
SOCFREEZ	allow freezing of SOC audits
SOCDESCR	allow entering a description when saving a SOC audit

These keys are not assigned to any user class (shp group) – you need to assign these keys in SHPCFG or Controller.

Audit notes

1. SOC history obeys the same UCX-CFG020DAP14 History Depth setting when saving audits – if the depth is set to 3 you will end up with 3 academic audits and 3 SOC audits.

Scribe notes

1. No special blocks are needed.
2. No special requirements are needed.
3. Update appropriate requirements using the SOC RuleTags
    a. SOC_CATALOG
    b. SOC_LABEL
    c. SOC_CREDITS
    d. SOC_ADVICE
4. Starting block needs Credits qualifier

Student Data

- Informational Data (rad_report_dtl)
  using UCX-BAN080 (or report.client.properties for Colleague schools)
- SOC Type attributes (rad_transfer_dtl)

UCX Tables

1. UCX-AUD032 – setup at least one SOC freeze type – "SOCFRZ"
2. UCX-RPT036 WEB60 SOC Report – make sure it exists and has DGW_SOC.xsl as the stylesheet.
3. UCX-BAN080 – used to provide sql statements to extract SOC information into the DW database from Banner. All SOC data needs to be bridged to the rad_report_dtl. (Colleague will use the report.client.properties file in admin/common instead of UCX-BAN080)

Stylesheet settings

You can show/hide the block titles and credits using this flag:
```
<xsl:variable name="vShowBlockHeaders">Y</xsl:variable>
```

You can show/hide the header and rule remarks and the header advice using these flags:
```
<xsl:variable name="vShowHeaderRemarks">Y</xsl:variable>
<xsl:variable name="vShowRuleRemarks">Y</xsl:variable>
<xsl:variable name="vShowHeaderAdvice">Y</xsl:variable>
```

You can show/hide the RuleComplete and RuleIncomplete requirements using these flags:
```
<xsl:variable name="vShowRULECOMPLETERules">Y</xsl:variable>
<xsl:variable name="vShowRULEINCOMPLETERules">Y</xsl:variable>
```

You can display the name of the transfer school in the Notes field using this flag. However, for the CLEP, DSST, etc classes, whatever is in the transfer school field will also display.
```
<xsl:variable name="vShowTransferSchoolInNotes">N</xsl:variable>
```

You can display the class's grade next to it using this flag. Example, "MATH 123 (B)"
```
<xsl:variable name="vShowClassesAppliedGrade">N</xsl:variable>
```

You can control the maximum number of courses from the rule to display in the Course Number field using this setting:
```
<xsl:variable name="vMaxCoursesToShow">5</xsl:variable>
```

This setting controls the text above the fall-through section:
```
<xsl:variable name="vFallthroughHeading">Electives for degree</xsl:variable>
```

When more than the vMaxCoursesToShow courses are found on a rule and the rule does not have ProxyAdvice or SOC_ADVICE this text will display in the Course Title column.
```
<xsl:variable name="vSeeCatalogText">See Catalog</xsl:variable>
```

These are the attributes associated with each type of credit awarded. If you need to use an attribute value that differs from the standard you can specify the attribute here.
```
<xsl:variable name="vSVSC-Attribute">SVSC</xsl:variable>
<xsl:variable name="vMOS-Attribute">MOS</xsl:variable>
<xsl:variable name="vCLEP-Attribute">CLEP</xsl:variable>
<xsl:variable name="vDSST-Attribute">DSST</xsl:variable>
<xsl:variable name="vECE-Attribute">ECE</xsl:variable>
<xsl:variable name="vCRTX-Attribute">CRTX</xsl:variable>
<xsl:variable name="vSOCO-Attribute">SOCO</xsl:variable>
```

If your school is on the semester system then use this setting. If your school is on the quarter system then move the <span> text inside the square brackets on the second line.
```
<xsl:variable name="LabelCalendar">
Semester Hours: [ <span style="color:black">X</span> ]
Quarter Hours: [  ]
</xsl:variable>
```

These four settings let you control the other text that appears on the report. You should contact the SOC office before making modifications to this text.
LabelPrivacyStatement
LabelCopies
LabelAgreement
LabelFooterStatement

Split Credits

Split credits occur in situations where a course is valued at more credits than is required. In that situation, what should happen to the "excess" credits? The default behavior of Degree Works is to move the course to another block or to the fall-through section of the audit if MAXTERM or MAXCREDITS is exceeded. The Scribe reserved words SPMAXTERM and SPMAXCREDIT can be used to force the excess credits to be applied by the Auditor to other places in the audit. Use the SPMAXTERM and/or SPMAXCREDIT block qualifiers when the Auditor Engine should split the credits automatically.

Syntax

The Parser Engine allows two block qualifiers for split credits: **SPMAXTERM** and **SPMAXCREDIT**. These two reserved words are extensions of the MAXTERM and MAXCREDIT qualifiers. The "SP" prefacing MAXTERM and MAXCREDIT signals to the Auditor Engine that the credits are to be split if the maximum is reached.

The format for MAXCREDIT is:

MAXCREDIT[S] real [IN | FROM] course_list

The format for SPMAXCREDIT follows:

SPMAXCREDIT[S] real [IN | FROM] course_list

The format for MAXTERM is:

MAXTERM {real CREDIT[S] | int CLASS[ES]} [IN | FROM] course_list

The format for SPMAXTERM follows:

SPMAXTERM {real CREDIT[S] [IN | FROM] course_list

where:

- course_list is a list of courses, e.g., MATH 101, 102, CHEM 300:400, PHYS @
- int is a numeric value from 1 to 3 digits long
- real is a decimal value with up to 3 digits on each side of the decimal

SPMAXTERM and SPMAXCREDIT are only allowed as block qualifiers. SPMAXTERM and SPMAXCREDIT are NOT allowed as rule qualifiers. SPMAXTERM is only allowed with CREDITS.

Scribe Guided Edit Mode is not setup to add these qualifiers. While in guided edit mode, you may construct your qualifiers as if they were MAXCREDITS and MAXTERM qualifiers and manually enter the necessary "SP" prefix.

Auditor Engine

The Auditor Engine is able to determine where the excess should be applied by examining the block in which the SPMAXTERM or SPMAXCREDIT is found. If this qualifier is in the starting block then the Auditor puts all excess credits and classes into the over-the-limit list. If the split credit qualifier is found in any block except the starting block, then the Auditor tries to apply the excess credits to rules in next or previous blocks. If the excess cannot be applied to other blocks then the excess goes to fall-through.

Class Count

If a course is split across two blocks, BLOCK1 and BLOCK2, then the course is counted as a class in the class totals for BLOCK1 and is also counted as a class in the totals for BLOCK2. It does, however, only count once in the overall class totals for all blocks.

```
BEGIN  # BLOCK1
  SPMAXCREDITS 3 IN CHEM @
  MAXCLASSES 2 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.

BEGIN  # BLOCK2
  MAXCLASSES 2 IN CHEM @
;
7 CREDITS IN CHEM @, GEOG @ LABEL "More science";
END.
```

If CHEM 105 was taken for 5 credits then the Auditor Engine may apply 3 credits to BLOCK1 and split the course so that the other 2 credits can be applied to BLOCK2. The CHEM course would be counted as a course in the MAXCLASSES qualifier in both blocks.

Exclusivity

Splitting a class between two blocks does not mean that the class is considered NONEXCLUSIVE. It will not be counted in a block's NONEXCLUSIVE count because the split course will be treated as if it is really two different courses.

```
BEGIN  # BLOCK1    # not the starting block
  SPMAXCREDITS 3 IN CHEM @
  NONEXCLUSIVE 1 CLASSES (ALLBLOCKS)
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.

BEGIN  # BLOCK2
;
7 CREDITS IN CHEM @, GEOG @ LABEL "More science";
1 class in math 1@ label "low level math";
END.
```

If CHEM 120 is applied to BLOCK1 for 5 credits the Auditor Engine may keep three of the credits in BLOCK1 and apply the other 2 credits to BLOCK2. If MATH 156 is applied to BLOCK1, the Auditor Engine would see that it can also apply this class to BLOCK2 because of the NONEXCLUSIVE qualifier. CHEM 120 is not considered as a NONEXCLUSIVE class while the placing of MATH 156 into two blocks is. CHEM 120 could, however, be applied nonexclusively to another block for 3 credits instead of MATH 156. Only the portion of the course that is applied to BLOCK1 can be applied elsewhere as a nonexclusive course.

Note: It is recommended that you do not use NONEXCLUSIVE with either SPMAXCREDITS or SPMAXTERM in the same Scribe block. The example shown above is for the purpose of illustrating the exclusivity of classes that have been split between blocks and is not an endorsement of the use of these two header qualifiers concurrently. The use of this combination of qualifiers may give unpredictable audit results.

Multiple Splits

A class can be split as many times as needed. The course may have been taken for a non-integer credit amount or the SPMAXCREDITS could have specified a non-integer value number of credits.

```
BEGIN  # BLOCK1  - not the starting block
  SPMAXCREDITS 3 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.

BEGIN  # BLOCK2  - not the starting block
  SPMAXCREDITS 1.5 IN CHEM @
;
7 CREDITS IN CHEM @, GEOG @ LABEL "More science";
END.

BEGIN  # BLOCK3
;
4 CREDITS IN CHEM @, GEOG @, ANTH @ LABEL "Even More science";
END.
```

If CHEM 120 was taken for 6 credits then the Auditor Engine could apply 3 credits to BLOCK1, 1.5 credits to BLOCK2, and the remaining 1.5 credits to BLOCK3. If CHEM 120 was taken for 6.7 credits then the Auditor could apply 3 credits to BLOCK1, 1.5 credits to BLOCK2, and the remaining 2.2 credits to BLOCK3.

Output

On output, Degree Works on the Web shows the courses that were split with their modified number of credits on the rule to which they were applied. Any part of a split credit course that ended up in over-the-limit or fall-through is also reported with its modified number of credits.

GPA

To be accurate, the GPA credits and the GPA grade-points should be reduced by the fraction of the course that was applied. This has the biggest implication when part of a course gets put into over-the-limit. If a 5 credit class that has 20 grade-points gets split into two rules where one gets 3 credits and the other 2 credits, then the Auditor Engine also splits the grade-points and credits used in the GPA calculation.

Split Power

One block has the power to tell the Auditor Engine that a course is to be split across blocks. The block to which the remaining credits may be applied does not have to indicate that it is OK to split credits.

```
BEGIN  # BLOCK1   - not the starting block
  SPMAXCREDITS 3 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.

BEGIN  # BLOCK2
```

```
;
7 CREDITS IN CHEM @, GEOG @ LABEL "More science";
END.
```

If CHEM 140 (5 credits) is applied to BLOCK1 then the Auditor Engine can split this course into BLOCK1 and BLOCK2 even though BLOCK2 does not explicitly give permission. As long as the Auditor Engine has permission from one of the blocks it can split the course as necessary.

Other Qualifiers

The split credit qualifiers cannot change the behavior of other block or rule qualifiers. The maximum number of credits specified by another qualifier cannot be exceeded even if a split credit qualifier exists in the same block header. Another block qualifier may remove the whole course from a block without allowing the split credits qualifier to split a course.

```
BEGIN
  MAXPERDISC 2 CREDITS IN (CHEM, PHYS)
  SPMAXCREDITS 3 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.
```

If a 4 credit chemistry course fits on this block then the Auditor Engine will end up removing the whole course because of the MAXPERDISC qualifier. This qualifier does not have the capability of splitting courses and therefore does not do so. Blocks like the above should be avoided. The qualifiers that you insert into blocks should not conflict with each other.

The Auditor Engine processes the split credit qualifiers first, so courses will be split, if needed, before other qualifiers are encountered.

```
BEGIN
  MAXPERDISC 3 CREDITS IN (CHEM)
  SPMAXCREDITS 3 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.
```

If a 4 credit chemistry course fits on this block then the Auditor Engine first splits the course. It then continues to process the MAXPERDISC qualifier and sees that the maximum has not been exceeded.

If there are multiple split credit qualifiers in the block then the Auditor Engine must obey all qualifiers.

```
BEGIN
  SPMAXTERM 2 CREDITS IN CHEM @
  SPMAXCREDITS 3 IN CHEM @
;
9 CREDITS IN CHEM @, PHYS @, MATH @ LABEL "Science stuff";
END.
```

If a 4 credit chemistry course fits on this block then the Auditor Engine first splits the course leaving 3 credits in this block because of SPMAXCREDITS. It then continues to process the SPMAXTERM qualifier and sees that the maximum has been exceeded and splits the course leaving only 2 credits in this block.

Best Fit

Due to the best fit algorithm used by the Auditor Engine, a course may be placed on a rule without splitting it across multiple rules. The split credit qualifier is only used by the Auditor Engine after it places the course in the current block as the best fit.

```
BEGIN   # BLOCK1
;
10 CREDITS IN CHEM 102, 103, PHYS 113:120;
END.

BEGIN   # BLOCK2    # not the starting block
  SPMAXCREDITS 2 IN CHEM 102
;
10 CREDITS IN CHEM @, PHYS @;
END.
```

If a student takes CHEM 102 for 5 credits, then there are two possible places that the course could fit. The Auditor Engine will see that the best fit for the course is in BLOCK1 (all else being equal, e.g., no PHYS courses were taken). The Auditor Engine will NOT try to apply 2 credits of CHEM 102 to BLOCK2 and the rest to BLOCK1. If the SPMAXCREDITS was on BLOCK1 then the Auditor Engine would indeed split the credits across the blocks.

Fall-Through

If a split course only fits or is only needed in one location then the remaining number of credits goes to fall-through if the split qualifier is not in the starting block.

```
BEGIN      # not the starting block
  SPMAXCREDITS 2 IN CHEM @
;
10 CREDITS IN CHEM @, PHYS @;
END.
```

If this is the only place where CHEM 102 can be applied then the Auditor Engine will split this 4 credit course leaving 2 here and putting the remaining 2 credits in fall-through.

Over-The-Limit

The remaining number of credits goes to over-the-limit if the split qualifier is in the starting block.

```
BEGIN      # IS the starting block
  SPMAXCREDITS 2 IN CHEM @
;
1 block (major);
END.

BEGIN      # some other block
;
10 CREDITS IN CHEM @, PHYS @;
END.
```

The Auditor Engine will split this 4 credit course leaving 2 credits in the block where it is applied and putting the remaining 2 credits in over-the-limit.

Fall-Through Split

A split qualifier on the starting block counts those courses applied to all blocks and those in the fall-through list. When processing a split qualifier the Auditor Engine may split a course that is in fall-through by putting part of it into over-the-limit in order to satisfy the maximum.

Multiple Split Qualifiers

In a situation like that below the smallest number of the two will be kept on rules. If a 5 credit CHEM course is taken then 3 credits of CHEM will end up in Over-The-Limit.

```
BEGIN   # # Starting block
  SPMAXCREDITS 3 IN CHEM @
  SPMAXTERM 2 CREDITS IN CHEM @
;
...
...
...
END.
```

Transfer Courses

Transfer Courses and GPA Calculations

The UCX-CFG020 DAP14 parameter has values to control the application of transfer courses to the MINGPA and MINGRADE Scribe Reserved Words and their use in GPA calculations.

Web Interface - Tool and Audits

Overview

The most important function of Degree Works is to perform a degree audit. There are two kinds of audits: real and what-if. A "real" audit processes a student's coursework against the requirements associated with the student's degree data from the student system. The results of a real audit are stored in a BLOB-like structure in the database. A "what-if" audit processes a student's coursework against the requirements associated with pretend degree data and/or pretend course work. The what-if feature is valuable when a student want to see "what" the audit results would be "if" the major, minor, concentration, specialization, liberal learning, or catalog year were changed or if the student complete future planned courses. The results of a "what-if" audit are only saved if the UCX-CFG020 DAP14 What-if History Count setting is “01” or greater.

The auditor performs one audit at a time for a particular student/school/degree combination. The auditor takes input from several locations:

- Run-time options: student ID, school, degree, include in-progress, and cutoff term
- Student curriculum bridged from the student system
- Student classwork history bridged from the student system
- List of requirements blocks based on the student’s curriculum
- Parsed requirement blocks – stored in the daptrees directory on the classic server
- Exceptions entered by advisors or the registrar’s office
- Notes entered by advisors
- Configuration settings (UCX-CFG020)

The audit results are stored in the database in a binary fashion. The results are then returned the web page as an XML document or are converted into the CPA tables – comprised mostly of the dap_result-dtl.

Degree Works Web Localizations

Introduction

In order to localize Degree Works reports at your institution, it is necessary to modify certain HTML, JavaScript, XML, XSL and CSS files. Degree Works utilizes XML and XSL technology in combination with HTML to display web reports. Degree Works generates an XML representation of the degree audit, which is rendered in a web browser using XSL to transform the document into HTML. Most of these localizations are managed using the Composer application. See the *Degree Works Composer Administrative Guide* for additional information.

Most of the details in this section apply to the Dashboard and not the Responsive Dashboard. Please also review the *Responsive Dashboard Administrative Guide* for details on setting up the Responsive Dashboard.

Section Organization

This Section of the Degree Works Technical Guide has been divided into the following sections:

Terms and Definitions. Defining terms used in this document.

Dashboard File List. A listing of each Degree Works file in the DashboardServlet war file.

Web Interface. The look and feel of the interface.

Web Interface: Header. Special configurations for the header section.

Miscellaneous Setup Files. A handful of files that are intended to be localized.

Localizing Worksheets. Reports viewed through the Worksheets, History, What-If, and Look-Ahead tabs.

Localizing the Student Educational Planner. Interface viewed through the Planner tab.

Special Topic: Reintegrating Localizations. How to reintegrate your localizations when processing a Degree Works software update.

Special Topic: Shepherd Scripts. Not all localizations you want to make can be made to files in the DashboardServlet war file. Localizations to the Shepherd Scripts may be required.

Note: If you make localizations to the Web worksheets you will most likely want to make the same modifications to the FOP files used for PDF and printed worksheets. These are located in your *$LOCAL_HOME/xsl* directory.

Terms and Definitions

Here are terms that will be used throughout this document. It is assumed you know and understand all these terms if you are to use this document to localize Degree Works.

Term	Definition
HTML	Hypertext Markup Language, the standard language for web pages. These files are named with an ".html" extension.
CSS	Cascading Style Sheets, a language used to describe the presentation of HTML documents. Degree Works uses CSS to define the colors and images that are presented to the web page. These files are named with a ".css" extension.
JavaScript	Web scripting language used by Degree Works. JavaScript files are named with a ".js" extension.
Frames	A specific type of HTML programming used heavily by Degree Works
JSON	JavaScript Object Notation. Data is returned from the classic server to the Responsive Dashboard in this format.
React	(aka React.js or ReactJS) A JavaScipt library used to create the Responsive Dashboard interface.
XML	Extensible Markup Language, a general-purpose markup language that allows users to define their own tags. Some Degree Works data are returned to the web browser as XML, whereupon XSL or JavaScript parses through them to extract information to display on the web as HTML. XML is used mostly by the Dashboard with limited use in the Responsive Dashboard.
XSL	Extensible Stylesheet Language, a language which allows the user to describe how to transform XML data. Degree Works uses XSLT to transform XML to HTML. These files are named with an ".xsl" extension. XSL files are used mostly by the Dashboard with limited use in the Responsive Dashboard.
XML Data Island	XML data that is embedded within an HTML page. Degree Works uses JavaScript to navigate this data island to extract specific data for different purposes, such as in the search results.

DashboardServlet File List

These files are used by the Dashboard and not by the Responsive Dashboard.

AuditBlocks.xsl	Included in the main worksheet stylesheets; this is where you make your localizations for the blocks and the requirements within them
AuditBlocksTF.xsl	Stylesheet for the displaying the contents of blocks in the Transfer Finder worksheet
AuditDisclaimer.xsl	Disclaimer text stored as XSL
AuditDisclaimer_Aid.xsl	Disclaimer text stored as XSL for the Financial Aid Audit
AuditDisclaimer_Ath.xsl	Disclaimer text stored as XSL for the Athletic Eligibility Audit
AuditExceptions.xsl	Exceptions Tab – Audit Report XSL Stylesheet
AuditHD.xsl	Worksheets Tab – Diagnostics Audit XSL Stylesheet
AuditLegend.xsl	Included in the worksheet stylesheets; this is where you make your localizations for the legend

AuditLocalizationsTF.xsl	Included in the Transfer Finder worksheet; this is where you make your localizations for special features unique to your Transfer Finder audit
AuditSections.xsl	Included in the main worksheet stylesheets; this is where you make your localizations for the sections like Fall-Through and Over-The-Limit.
AuditSEP.xsl	Planner Tab (Student Educational Planner) – Planner Audit Report XSL Stylesheet
AuditStudentHeader.xsl	Included in the worksheet stylesheets; this is where you make your localizations for the student header
AuditStudentHeaderSelfService.xsl	Student Header for the Transfer Equivalency Self-Service worksheet
AuditTranscript.xsl	Worksheets Tab – Class History Report XSL Stylesheet sorted by term
AuditTranscript2.xsl	Worksheets Tab – Class History Report XSL Stylesheet sorted by discipline
BrowserSniffer.js	JavaScript that checks for valid web browsers for compatibility
ClassTranscript.xsl	Athletic Eligibility – Class Summary report stylesheet
CommonTemplates.xsl	Common templates used by several stylesheets
CourseInfo.xsl	Course Link styleheet
CurrRules.xsl	What-if Curriculum Rules stylesheet
default.jsp	
DegreeInfo.jsp	Installed GPA Calculator “Total Credits Required” link – intended for localization
DGW_AdvisorDataIsland.js	JavaScript that parses an XML Data Island to manage a list of advisors for emailing.
DGW_Aid_Report.xsl	Financial Aid Report XSL Stylesheet
DGW_Ath_Report.xsl	Athletic Eligibility XSL Stylesheet
DGW_AuditViewReport.xsl	Stylesheet used by the planner audit worksheet
DGW_Charts.js	JavaScript that is used by the GPA calculators
DGW_Control.js	JavaScript that contains miscellaneous shared functions.
DGW_DragAndDrop.js	JavaScript that contains functions for dragging and dropping courses in the Planner (Student Educational Planner)
DGW_EMExceptionsDataIsland.js	JavaScript that parses an XML Data Island to manage Exception Management “Exceptions Report”.
DGW_EMPetitionsDataIsland.js	JavaScript that parses an XML Data Island to manage Exception Management “Manage Petitions Waiting Approval,” “Apply Approved Petitions,” “View Petitions Applied as Exceptions,” “View Rejected Petitions,” “Fix Petition Status”
DGW_Exceptions.js	JavaScript that contains functions for saving and deleting exceptions (Exceptions Tab)
DGW_Functions.js	JavaScript that contains miscellaneous shared functions
DGW_GPADataIsland.js	JavaScript that parses an XML Data Island to manage the GPA Term calculator in-progress classes.
DGW_HistoryDataIsland.js	JavaScript that parses an XML Data Island to generate a list of historic audits in the History Tab
DGW_LookAhead.js	JavaScript that contains functions for the Look Ahead Tab
DGW_objKeyEventGlobal.js	JavaScript that contains miscellaneous shared functions.

DGW_objKeyEventHandler.js	JavaScript that contains miscellaneous shared functions.
DGW_Petitions.js	JavaScript that contains functions for the Petitions Tab
DGW_Refresh.xsl	Reads the response from the refresh student data request
DGW_Registration.xsl	Worksheets Tab “Registration Checklist” XSL Stylesheet
DGW_ReloadMain.js	JavaScript that contains miscellaneous shared functions
DGW_ReloadSearch.js	JavaScript that contains miscellaneous shared functions
DGW_Report.xsl	Worksheets Tab “Student View,” “Registrar Report,” “Graduation Checklist” XSL Stylesheet
DGW_SearchDataIsland.js	JavaScript that parses an XML Data Island to manage the Search screen
DGW_SEPCompareView.js	JavaScript that contains functions for the Planned vs. Taken report in the Planner (Student Educational Planner) Tab
DGW_SEPDataIsland.js	JavaScript that parses an XML Data Island to manage the Planner (Student Educational Planner) Tab (both Notes and Calendar “Edit” mode)
DGW_SEPEdit.js	JavaScript that contains functions for the Planner (Student Educational Planner) Tab (drawing the form and saving the plan)
DGW_SEPView.js	JavaScript that contains functions for the Planner (Student Educational Planner) Tab (View mode)
DGW_SimpleSearchDataIsland.js	JavaScript that parses an XML Data Island for a simple student ID search
DGW_SkinnyReport.xsl	Stylesheet used for the worksheet in the classic planner
DGW_SOC.xsl	Worksheets Tab “SOC Report” XSL Stylesheet
DGW_StudentFunctions.js	JavaScript that contains miscellaneous shared functions
DGW_TermDataIsland.js	JavaScript that parses an XML Data Island for terms
DashboardStyles.css	All Degree Works styles are defined here. (except for the SEP3, Transfer Finder, Scribe, Responsive Dashboard (including SEP4), Transit and Controller)
DGW_Tabs.js	JavaScript that contains functions for the tabs in Degree Works
DGW_TMPDataIsland.js	JavaScript that parses an XML Data Island to manage the Templates Tab
DGW_TMPEdit.js	JavaScript that contains functions for Templates (drawing the form and saving the template)
DGW_Window.js	JavaScript that contains functions for creating pop-up confirmation windows
error.jsp	Handles the display of an error returned from the classic server
FormatDate.xsl	Formats the date display based on UCX-CFG020 WEB setting for the worksheets.
ImageSwap.js	JavaScript that contains functions for manipulating the Tab images
jquery.min.js	Jquery javascript utility functions
jquery-ui.min.js	Jquery javascript ui functions
login.jsp	
logout.jsp	

Notes.xsl	Display the notes on the Notes tab
Petitions.xsl	Display the petitions on the Petitions tab
PlanSEP.xsl	Planner (Student Educational Planner) Tab View Mode (Both Notes Mode and Calendar Mode)
PlanSEPCompareView.xsl	Planner (Student Educational Planner) Tab “Planned vs. Taken” Report
ProgressBar.xsl	The worksheet progress bar logic
RADData.xsl	Worksheets Tab “Student Data Report”
SD_AdviseeWait.jsp	Web page that displays when an automatic advisee search is performed
SD_BodyFrameset.html	Creates the frames within the body of the main window
SD_DepartWait.jsp	Web page that displays when an automatic department search is performed
SD_Exceptions_Introduction.jsp	Web page that instructs the user to select an exception type. It displays when the Exceptions Tab is clicked
SD_Exceptions_LoadContextFrame.jsp	Web page for loading Exceptions Tab information
SD_Exceptions_LoadFrame.jsp	Web page for loading Exceptions Tab information
SD_Exceptions_NoSave.jsp	Web page for loading Exceptions Tab information
SD_Exceptions_Save.jsp	Web page for loading Exceptions Tab information
SD_ExceptionsSort.html	Web page for Exception Management “Exceptions Report”
SD_ExceptionsSortCount.html	Web page for Exception Management “Exceptions Report”
SD_Exp_Mgt_Introduction.jsp	Web page for Exception Management introduction
SD_ExpHeader.jsp	Web page for the Exception Management header. It is intended to be localized similarly to SD_HeaderFrame.jsp
SD_ExpMgt_NoSave.jsp	Web page for Exception Management “Apply Approved Petitions”
SD_ExpMgt_Save.jsp	Web page for Exception Management “Apply Approved Petitions”
SD_ExpMgtLogon.jsp	Web page for logging on to Exception Management
SD_GeneralIntroduction.jsp	Web page that displays when the user first logs on. It is intended to be localized.
SD_GPALoadFrameForm.jsp	Web page for GPA Calc tab for describing the different calculators
SD_HeaderFrame.jsp	Web page for the overall header. It is intended to be localized similarly to SD_ExpHeader.jsp
SD_HelpAdmin.jsp	Web page for context-based help: Admin Tab. It is intended to be localized.
SD_HelpAid.jsp	Web page for context-based help: Financial Aid Tab. It is intended to be localized.
SD_HelpAidHistory.jsp	Web page for context-based help: Financial Aid History Tab. It is intended to be localized.
SD_HelpAth.jsp	Web page for context-based help: Athletic Eligibility Tab. It is intended to be localized.
SD_HelpAthHistory.jsp	Web page for context-based help: Athletic Eligibility History Tab. It is intended to be localized.
SD_HelpAuditHistory.jsp	Web page for context-based help: History Tab. It is intended to be localized.
SD_HelpAuditRun.jsp	Web page for context-based help: Worksheets Tab: Process New. It is intended to be localized.
SD_HelpAuditView.jsp	Web page for context-based help: Worksheets Tab: View. It is intended to be localized.

SD_HelpExceptions.jsp	Web page for context-based help: Exceptions Tab. It is intended to be localized.
SD_HelpExpMgt.jsp	Web page for context-based help: Exception Management. It is intended to be localized.
SD_HelpFind.jsp	Web page for context-based help: Student Search. It is intended to be localized.
SD_HelpGPACalculator.jsp	Web page for context-based help: Student Search. It is intended to be localized.
SD_HelpLookAhead.jsp	Web page for context-based help: Look Ahead Tab. It is intended to be localized.
SD_HelpNotes.jsp	Web page for context-based help: Notes Tab. It is intended to be localized.
SD_HelpPetitions.jsp	Web page for context-based help: Petitions Tab. It is intended to be localized.
SD_HelpSEP.jsp	Web page for context-based help: Planner (Student Educational Planner) Tab. It is intended to be localized.
SD_HelpSoc.jsp	Web page for context-based help: SOC Tab. It is intended to be localized.
SD_HelpSocHistory.jsp	Web page for context-based help: SOC History Tab. It is intended to be localized.
SD_HelpTMP.jsp	Web page for context-based help: Planner Template Tab. It is intended to be localized
SD_HelpWhatIfAudit.jsp	Web page for context-based help: What-If Tab. It is intended to be localized.
SD_HelpWhatIfHistory.jsp	Web page for context-based help: What-If History Tab. It is intended to be localized.
SD_Historic_Introduction.jsp	Web page for History Tab telling the user how to use the service.
SD_LoadFrameForm.html	Web page for loading into many different frames.
SD_LookLoadFrameForm.jsp	Web page for loading into a Look Ahead frame
SD_MainBackground.html	Web page for loading into the main background
SD_MainBorderLeft.html	Web page for loading into the left border around the interface
SD_MainBorderRight.html	Web page for loading into the right border around the interface
SD_MainFooter.jsp	Web page for loading into the bottom border around the interface
SD_PlainBorder.html	Web page for loading into the search page
SD_SearchWait.jsp	Web page for displaying a message while a search is performed
SD_SEPAuditFrameForm.html	Web page for loading into the Planner (Student Educational Planner) Audit frame
SD_SEPPlanFrameForm.html	Web page for loading into the Planner (Student Educational Planner) Plan frame
SD_StudentBody.jsp	Web page for loading into the search page
SD_StudentFooter.jsp	Web page for loading into the search page bottom border
SD_StudentSort.html	Web page for loading the search results into a sortable form
SD_TMPPlanFrameForm.html	Web page for loading into the Template Tab.
SD_TMPTop.jsp	Web page in the context area of the Templates Tab. The default reads, “Help your students Plan for Success”
SD_Waiting.jsp	Web page for issuing a “Please wait while your request is processed” message

SD_WhatIf_LoadFrame.jsp	Web page for loading into the What-If Tab.
SD_WhatIfCurrRules_LoadFrame.jsp	Web page for loading into the What-If Tab when curriculum rules is enabled.
SD_WhiteFrame.html	Web page containing a white background for general use
SelfServiceAudit.xsl	Transfer Equivalency Self-Service audit
SEP_Approval_Context.xsl	Classic Planner approval context frame stylesheet
SEP_Context.xsl	Classic Planner context frame stylesheet
SEP_Save.xsl	Classic Planner save action stylesheet
SEP_SaveApprStatus.xsl	Classic Planner save approval action stylesheet
SEP_TemplateSearch.xsl	Classic Planner template search stylesheet
SEP_TemplateSearchResults.xsl	Classic Planner template search results stylesheet
SupportedBrowserCheck.js	JavaScript that checks for valid web browsers for compatibility
TMP_Buttons.xsl	Template Tab Buttons frame XSL Stylesheet
TMP_Context.xsl	Template Tab context frame XSL Stylesheet
TMP_Save.xsl	Template Tab Save action XSL Stylesheet
TMP_Search.xsl	Template Tab Search XSL Stylesheet
ToolTip.js	JavaScript for displaying hints
TransferAudit.xsl	Stylesheet for audits in Transfer Finder
TransferAuditSummary.xsl	Stylesheet for audit summary in Transfer Finder
TreqDisclaimer.xsl	Disclaimer text for Transfer Equivalency Admin audit worksheet
TreqReportArticulate.xsl	Stylesheet for Transfer Equivalency Admin articulation and audit worksheet
TreqReportAudit.xsl	Stylesheet for Transfer Equivalency Admin audit worksheet
WhatIf3.xsl	What-if with specializations stylesheet

Note: You may have more web files installed than the ones listed here. Those extra files represent deprecated code or images that are no longer used.

Web Interface

These comments apply to the Dashboard and not by the Responsive Dashboard.

TIP: Utilizing the "View Source" option in your web browser to see the actual HTML source is the recommended way to determine which style is applied to the object (i.e., background color, font color, image, etc.) you wish to localize. You may also want to try using Firebug in Firefox, or the Google Developer Tools in Chrome as a useful tool when modifying web pages.

Every color and image you see in Degree Works is defined and can be modified either in the HTML or in the CSS.

Web Interface – Header

These comments apply to the Dashboard and not by the Responsive Dashboard.

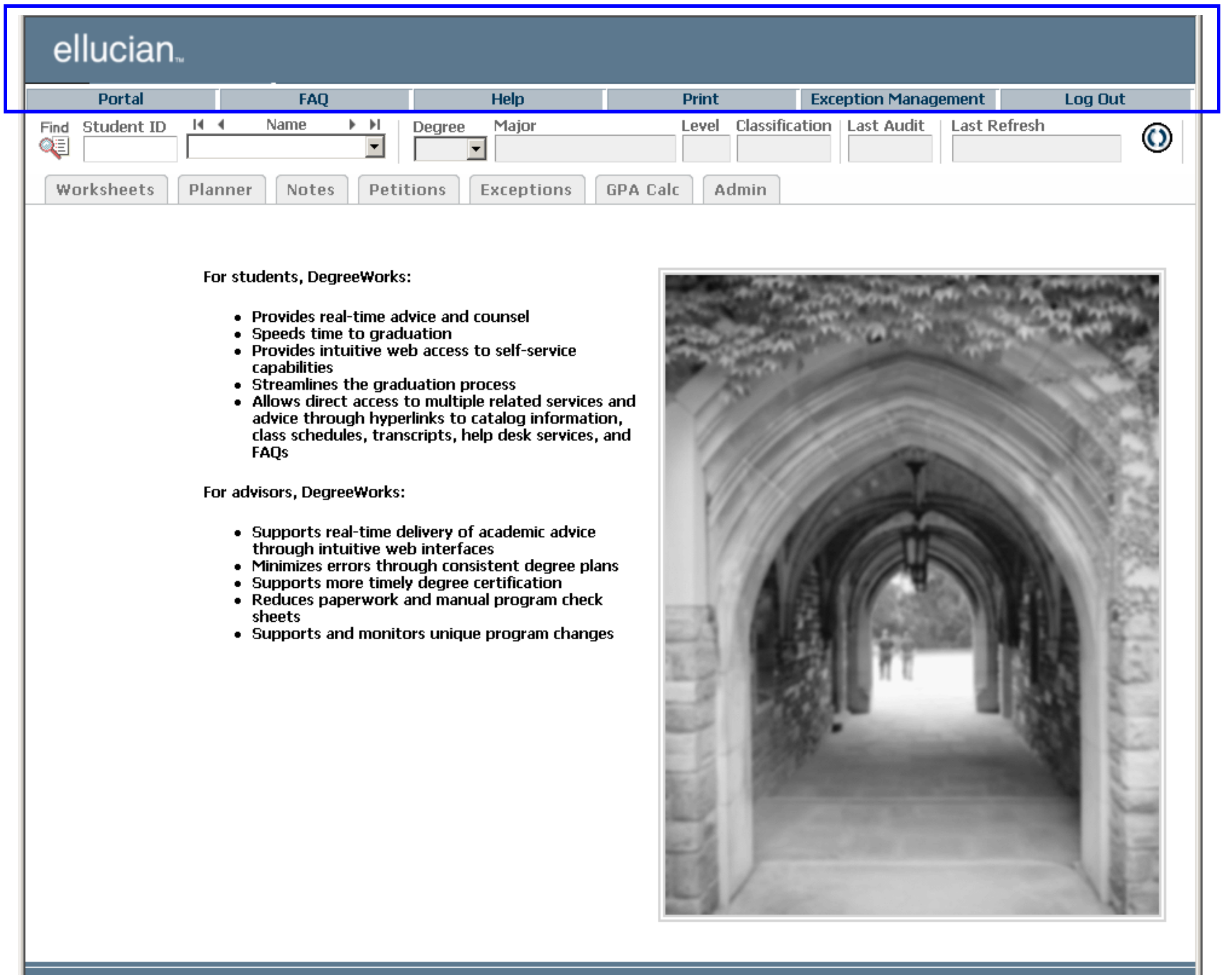
The header section of the interface is specially designed for localization. There are two HTML files that are used for generating this header section:

SD_HeaderFrame.jsp (Standard header)

SD_ExpHeader.jsp (Exception Management header)

To localize the text of the links in these files, create a localized version of DashboardServletMessages.properties with Composer and change the **dw.dashboard.header.link**.* and/or **dw.dashboard.exceptionManagement.header.link**.* properties values. Some links such as "Help", "FAQ" and "Portal" will only display if they have been enabled. This is managed with the **localization.dashboard.header.show*** and **localization.exceptionManagement.show*** Shepherd settings.

The Portal and FAQ links are managed with the **localization.dashboard.header.url*** and **localization.exceptionManagement.url*** Shepherd settings.
There is a UCX-CFG020 WEB setting that controls the height of this frame. This way, you can make this header section as short or tall as you see fit. Typically, this is determined by height of the image (a school logo, for example) you may choose to utilize. This setting is called "Header Frame Height". It is a two-byte entry. The unit of measure is pixels. When you modify this entry, be sure to issue a "webrestart" so that your changes will take effect.



For students, DegreeWorks:

- Provides real-time advice and counsel
- Speeds time to graduation
- Provides intuitive web access to self-service capabilities
- Streamlines the graduation process
- Allows direct access to multiple related services and advice through hyperlinks to catalog information, class schedules, transcripts, help desk services, and FAQs

For advisors, DegreeWorks:

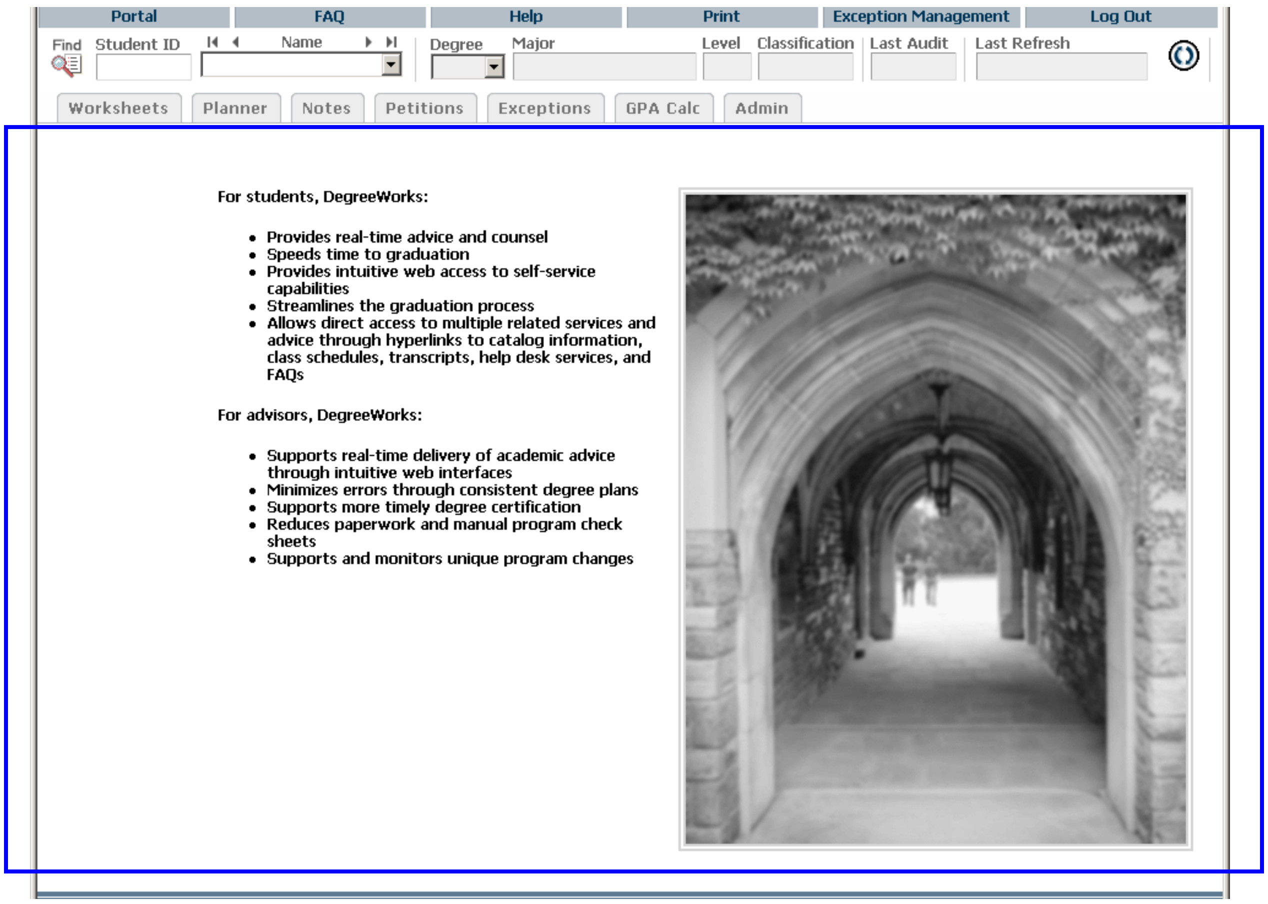
- Supports real-time delivery of academic advice through intuitive web interfaces
- Minimizes errors through consistent degree plans
- Supports more timely degree certification
- Reduces paperwork and manual program check sheets
- Supports and monitors unique program changes

Miscellaneous Setup Files

These files apply to the Dashboard and not by the Responsive Dashboard.

File Name	Description
DegreeInfo.jsp	Static web page containing degree information for use in the GPA Calculator's Graduation Calculator. The intent is to give users a source of information for "total credits required" for their specific degree.
default.jsp	Login page
AuditDisclaimer.xsl	Disclaimer text for the worksheet
SD_GeneralIntroduction.jsp	Introduction page that displays when a user first logs on to Degree Works. See the screenshot below. It is intended that the text and image on this page be localized via Composer – the text is defined in the dw.dashboard.introduction property in DashboardServletMessages.properties, and the image is front_background.gif.

SD_GeneralIntroduction.jsp



For students, DegreeWorks:

- Provides real-time advice and counsel
- Speeds time to graduation
- Provides intuitive web access to self-service capabilities
- Streamlines the graduation process
- Allows direct access to multiple related services and advice through hyperlinks to catalog information, class schedules, transcripts, help desk services, and FAQs

For advisors, DegreeWorks:

- Supports real-time delivery of academic advice through intuitive web interfaces
- Minimizes errors through consistent degree plans
- Supports more timely degree certification
- Reduces paperwork and manual program check sheets
- Supports and monitors unique program changes

Miscellaneous Configurations

There are a number of configurations you can make to certain UCX entries to further localize your Degree Works dasboard. Here are the UCX entries and their brief descriptions:

These tables apply the Dashboard and to the Responsive Dashboard. Be sure to review the *Degree Works Configuration Technical Guide* documentation for additional setttings that apply to both dashboards.

UCX Table (Code)	**Description**
UCX-CFG020 (WEB)	Controls various settings throughout the web interface
UCX-CFG020 (WHATIF)	Controls content in the What-If tab
UCX-CFG020 (SEARCH)	Controls the search page. Include or exclude various search items such as Degree, Major, etc.
UCX-SCR001	Defines labels for terms such as Degree, Major, Credits, etc.
UCX-STU016	Term values need to be flagged for use in the Planner (Student Educational Planner)
UCX-AUD027	What-If major picklist filters
UCX-STU035	Catalog year values need to be flagged for use in the Planner (Student Educational Planner)
UCX-RPT036	Report types (See below for more details)
UCX-CFG071	Predefined Note text

Localizing Worksheets

To generate the worksheets in Degree Works, an XML representation of the degree audit is generated and returned to the browser. One of the XSL files below (based on context) transforms that XML into a formatted HTML document.

XSL	**Where it is used**
AuditExceptions.xsl	Exceptions Tab, exceptions audit report
AuditHD.xsl	Worksheets Tab, typically referred to as “Diagnostics Audit”. Do not modify content. It is used by the support team.
AuditSEP.xsl	Student Educational Planner Audit report
AuditTranscript.xsl	Worksheets Tab, “Class History” – sorted by term
AuditTranscript2.xsl	Worksheets Tab, “Class History” – sorted by discipline
DGW_Registration.xsl	Worksheets Tab, typically referred to as “Registration Report”
DGW_Aid_Report.xsl	Aid Tab, used in the standard financial aid audit reports
DGW_Ath_Report.xsl	Athletic Eligibility tab, used in the standard athlete audit reports.
DGW_SOC.xsl	Worksheets Tab, typically referred to as “SOC Military Report”
DGW_Report.xsl	Worksheets Tab, used in the standard audit reports (except the ones listed above)
RADData.xsl	Worksheets Tab, “Student Data Report”

Typically, the only XSL that is localized is the DGW_Report.xsl file. That is the XSL that controls the Student View, Registrar Report, and Graduation Checklist reports. This documentation will only describe the features available in DGW_Report.xsl.

To change the Class History report to use AuditTranscript2.xsl change the XML31 entry in UCX-RPT036.

DGW_Report.xsl, DGW_Aid_Report.xsl and DGW_Athl_Report.xsl each include and share these three stylesheets:
AuditBlocks.xsl

AuditLegend.xsl

AuditStudentHeader.xsl

Most of your localizations will actually be made to these three files and not to the parent Report files. You can make changes to the student header in one file allowing the changes to be seen by all three worksheets, for example. This helps you make sure your changes show consistently throughout the three different reports.

There are four key pieces that comprise the final web output. They are

1) Audit XML – standard XML that Degree Works generates (not available to be localized. See next section for a brief description of the XML.)

2) UCX-RPT036 – report settings that can be changed using Controller
   a. Title: This is the report name that will display in the report picklist as well as the top of the web report.
   b. XSL Stylesheet: Each report type can utilize its own specific XSL. For example, WEB31 “Student View” is delivered to use DGW_Report.xsl. If, however, you would rather create a new XSL (“myReport.xsl”) you can simply change this value to myReport.xsl and Degree Works will use the new one instead.
   c. Show Block Remarks
   d. Show Block Qualifiers
   e. Show Block Exceptions
   f. Show Block Include list
   g. Show Block Advice
   h. Show Rule Remarks
   i. Show Rule Qualifiers
   j. Show Rule Exceptions
   k. Show Rule Advice
   l. Show Rule Requirement Text
   m. Show Courses Applied
   n. Show Fallthrough (Electives) section
   o. Show Insufficient (Failed) section
   p. Show Over-the-Limit section
   q. Show In-Progress section
   r. Show Notes section
   s. Show Exceptions section
   t. Show Errors (not implemented)
   u. Show Legend
   v. Show Disclaimer
   w. Show Progress Bar
   x. And/Or Advice
   y. Show Prerequisite Indicator
   z. Create Course Link
   aa. Show Course Keys ONLY
   bb. Show Student Header
   cc. Show Student Alerts and Reminders section
   dd. Show Student System GPA in header

3) DGW_Report.xsl – XSL that can be localized based on your specific needs. Any configurations that cannot be accomplished using the UCX-RPT036 settings can be modified here. There are, however, a few key components that will be explained here:

   a. Block headers: Requirement block header sections can contain between 0 and 4 data elements in the standard DGW_Report.xsl. They are: Block Catalog Year, Block GPA, Block Credits/Classes Applied, Block Credits/Classes Required.

      i. See the “tBlockHeaderChoose” template. Here are the different attributes of requirement blocks that you can use to identify blocks (@ = the attribute name):

         1. Block ID (@Req_id)
         2. Block Title (@Title)
         3. Block Type (@Req_type)
         4. Block Percent Complete (@Per_complete)
         5. Catalog Year Literal (@Cat_yrLit)
         6. Block Value (@Req_value)
         7. Catalog Year Start (@Cat_yr_start)
         8. Catalog Year Stop (@Cat_yr_stop)
         9. Block GPA (@GPA)
         10. Block GPA Credits applied (@Gpa_credits)
         11. Block Classes applied (@Classes_applied)
         12. Block Credits applied (@Credits_applied)
         13. Block GPA Grade Points applied (@Gpa_grade_pts)

      ii. The standard DGW_Report.xsl has a few examples of how to identify specific requirement blocks and then use specific block header templates for those blocks.

      Here is one coding example:

         1.
```
<xsl:when test="@Req_type = 'OTHER'">
        <xsl:choose>
                <xsl:when test="@Req_value = 'GENED'">
                <!-- OTHER GENED blocks -->
                <xsl:call-template name="tBlockHeader_1"/>
                </xsl:when>
                <xsl:otherwise>
                <xsl:call-template name="tBlockHeader_2"/>
                </xsl:otherwise>
        </xsl:choose>
</xsl:when>
```

            This means: For blocks with a dap_block_type of “OTHER”, if the dap_block_value is “GENED” then call the “tBlockHeader_1” template. For blocks with a dap_block_type of “OTHER”, if the dap_block_value is NOT “GENED” then call the “tBlockHeader_2” template.

         2. There are 16 predefined “tBlockHeader_X” templates that encompass the different permutations of the four standard block header data elements.

iii. There are a few XSL variables that are defined at the top of DGW_Report.xsl. These are labels on the Degree Works web report that can be customized by simply localizing the value of these variables.

1. LabelProgressBar: Text just above the progress bar
2. LabelStillNeeded: Installed as “Still Needed”
3. LabelAlertsReminders: Title of the Alerts and Reminders section
4. LabelFallthrough: Title of the Fallthrough section
5. LabelInprogress: Title of the In progress section
6. LabelOTL: Title of the Over-The-Limit section
7. LabelInsufficient: Title of the Insufficient section
8. LabelSplitCredits: The title of the Split Credits section
9. LabelIncludedBlocks: Installed as "Blocks included in this block"
10. vShowTitleCreditsInHint: If Y, title and credits of courses in advice will display
11. vLabelSchool: Label for "School"
12. vLabelDegree: Label for "Degree"
13. vLabelMajor: Label for "Major"
14. vLabelCollege: Label for "College"
15. vLabelLevel: Label for "Level"
16. vLabelAdvisor: Label for "Advisor"
17. vLabelStudentID: Label for " ID"
18. vLabelStudentName: Label for "Student"
19. vLabelOverallGPA: Label for "Overall GPA"
20. vGPADecimals: Specify how many decimals to display in GPA.
21. vCreditDecimals: Specify how many decimals to display for credit values
22. vProgressBarPercent: If Y, show the progress bar for percent complete (rules)
23. vProgressBarCredits: If Y, show the progress bar for percent complete (credits)
24. vProgressBarRulesText: Label for the progress bar (rules)
25. vProgressBarCreditsText: Label for the progress bar (credits)

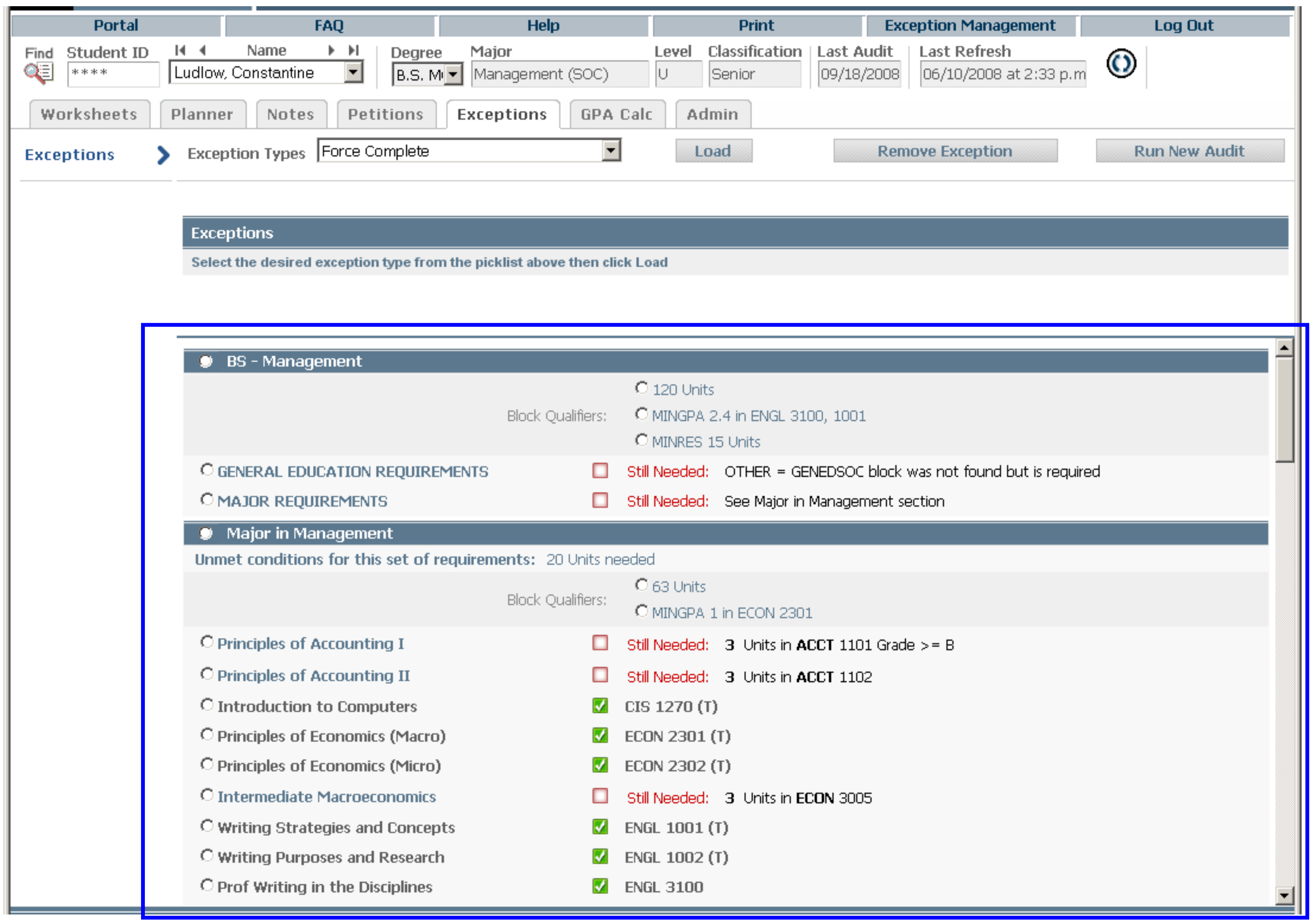
4) DashboardStyles.css – this CSS will be localized using your specific colors and styles. Here is the general structure of the audit XML. See an actual XML audit for all possible element types.

```
<Report>
      <Audit>
            <AuditHeader />
            <Block>
                  <Header>
                        <Qualifier> </Qualifier>
                        <Qualifier> <Text> </Text> </Qualifier>
                        <Qualifier> <SubText> </SubText> </Qualifier>
                        <Qualifier> </Qualifier>
                  </Header>
                  <Rule>
                        <Classes_applied> </Classes_applied>
                        <Credits_applied> </Credits_applied>
                        <Requirement>
                              <Course />
                              <Qualifier />
                        </Requirement>
                        <Advice>
                        </Advice>
                  </Rule>
            </Block>
```

```
<Clsinfo>
       <Class>
              <Loc />
              <Loc />
       </Class>
</Clsinfo>
<Fallthrough>
       <Class />
       <Class />
       <Class />
</Fallthrough>
<OTL>
       <Class />
       <Class />
       <Class />
</OTL>
<Insufficient>
       <Class />
       <Class />
       <Class />
</Insufficient>
<In_progress>
       <Class />
       <Class />
       <Class />
</In_progress>
<FitList>
       <Class />
       <Class />
       <Class />
</FitList>
<Deginfo>
       <DegreeData />
       <Custom />
       <Custom />
       <Custom />
       <Report />
       <Report />
       <Report />
       <Goal />
       <Goal />
       <Goal />
       <Goal />
       <Goal />
       <Goal />
</Deginfo>
<ExceptionList>
       <Exception />
       <Exception />
       <Exception />
</ExceptionList>
<Notes>
       <Note>
              <Text>
              <Text>
              <Text>
       </Note>
       <Note>
              <Text>
              <Text>
              <Text>
       </Note>
       <Note>
```

```
                            <Text>
                            <Text>
                            <Text>
                     </Note>
              </Notes>
       </Audit>
</Report>
```

AuditExceptions.xsl



Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

Exceptions > Exception Types: Force Complete

Exceptions

Select the desired exception type from the picklist above then click Load

BS - Management

Block Qualifiers: 120 Units; MINGPA 2.4 in ENGL 3100, 1001; MINRES 15 Units

GENERAL EDUCATION REQUIREMENTS	☐	Still Needed: OTHER = GENEDSOC block was not found but is required
MAJOR REQUIREMENTS	☐	Still Needed: See Major in Management section

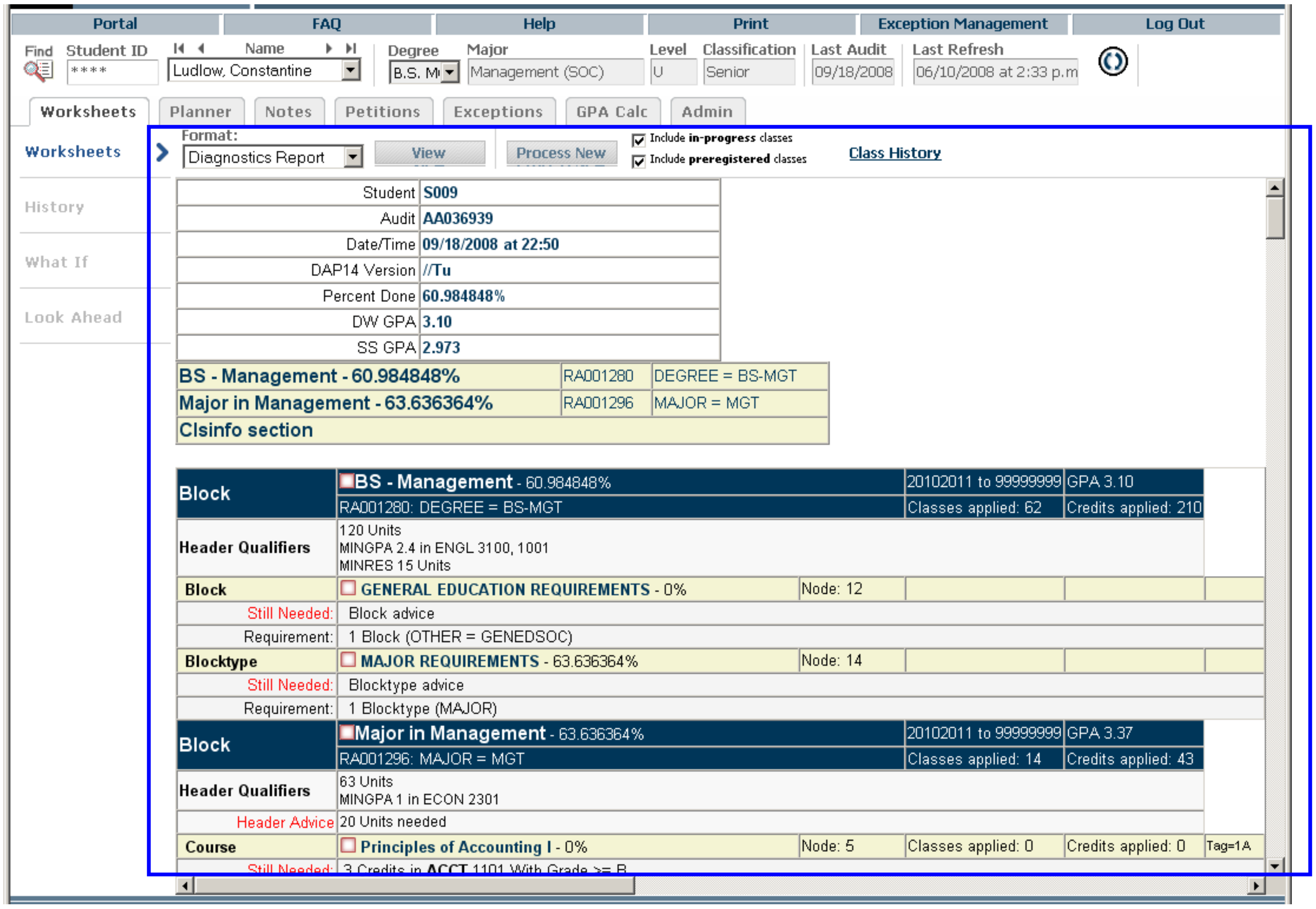
Major in Management

Unmet conditions for this set of requirements: 20 Units needed

Block Qualifiers: 63 Units; MINGPA 1 in ECON 2301

Principles of Accounting I	☐	Still Needed: 3 Units in **ACCT** 1101 Grade >= B
Principles of Accounting II	☐	Still Needed: 3 Units in **ACCT** 1102
Introduction to Computers	✓	CIS 1270 (T)
Principles of Economics (Macro)	✓	ECON 2301 (T)
Principles of Economics (Micro)	✓	ECON 2302 (T)
Intermediate Macroeconomics	☐	Still Needed: 3 Units in **ECON** 3005
Writing Strategies and Concepts	✓	ENGL 1001 (T)
Writing Purposes and Research	✓	ENGL 1002 (T)
Prof Writing in the Disciplines	✓	ENGL 3100

AuditHD.xsl (Do not modify content. It is used by the Ellucian support team.)



Find	Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
	****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

Format: Diagnostics Report

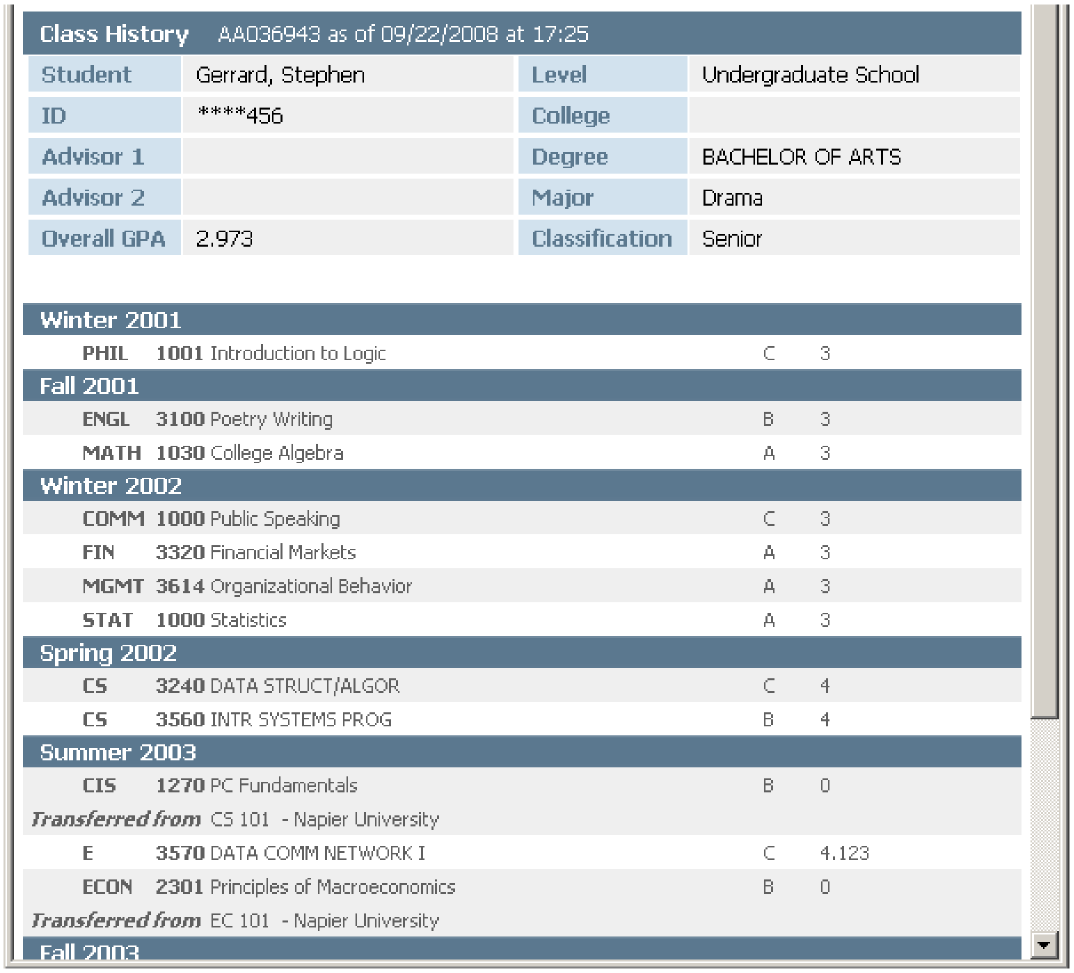
☑ Include **in-progress** classes
☑ Include **preregistered** classes

Student	**S009**
Audit	**AA036939**
Date/Time	**09/18/2008 at 22:50**
DAP14 Version	**//Tu**
Percent Done	**60.984848%**
DW GPA	**3.10**
SS GPA	**2.973**

BS - Management - 60.984848%	RA001280	DEGREE = BS-MGT
Major in Management - 63.636364%	RA001296	MAJOR = MGT
Clsinfo section		

Block	**BS - Management** - 60.984848%		20102011 to 99999999	GPA 3.10
	RA001280: DEGREE = BS-MGT		Classes applied: 62	Credits applied: 210
Header Qualifiers	120 Units MINGPA 2.4 in ENGL 3100, 1001 MINRES 15 Units			
Block	**GENERAL EDUCATION REQUIREMENTS** - 0%	Node: 12		
Still Needed:	Block advice			
Requirement:	1 Block (OTHER = GENEDSOC)			
Blocktype	**MAJOR REQUIREMENTS** - 63.636364%	Node: 14		
Still Needed:	Blocktype advice			
Requirement:	1 Blocktype (MAJOR)			
Block	**Major in Management** - 63.636364%		20102011 to 99999999	GPA 3.37
	RA001296: MAJOR = MGT		Classes applied: 14	Credits applied: 43
Header Qualifiers	63 Units MINGPA 1 in ECON 2301			
Header Advice	20 Units needed			
Course	**Principles of Accounting I** - 0%	Node: 5	Classes applied: 0	Credits applied: 0 Tag=1A
Still Needed:	3 Credits in **ACCT** 1101 With Grade >= B			

AuditTranscript.xsl

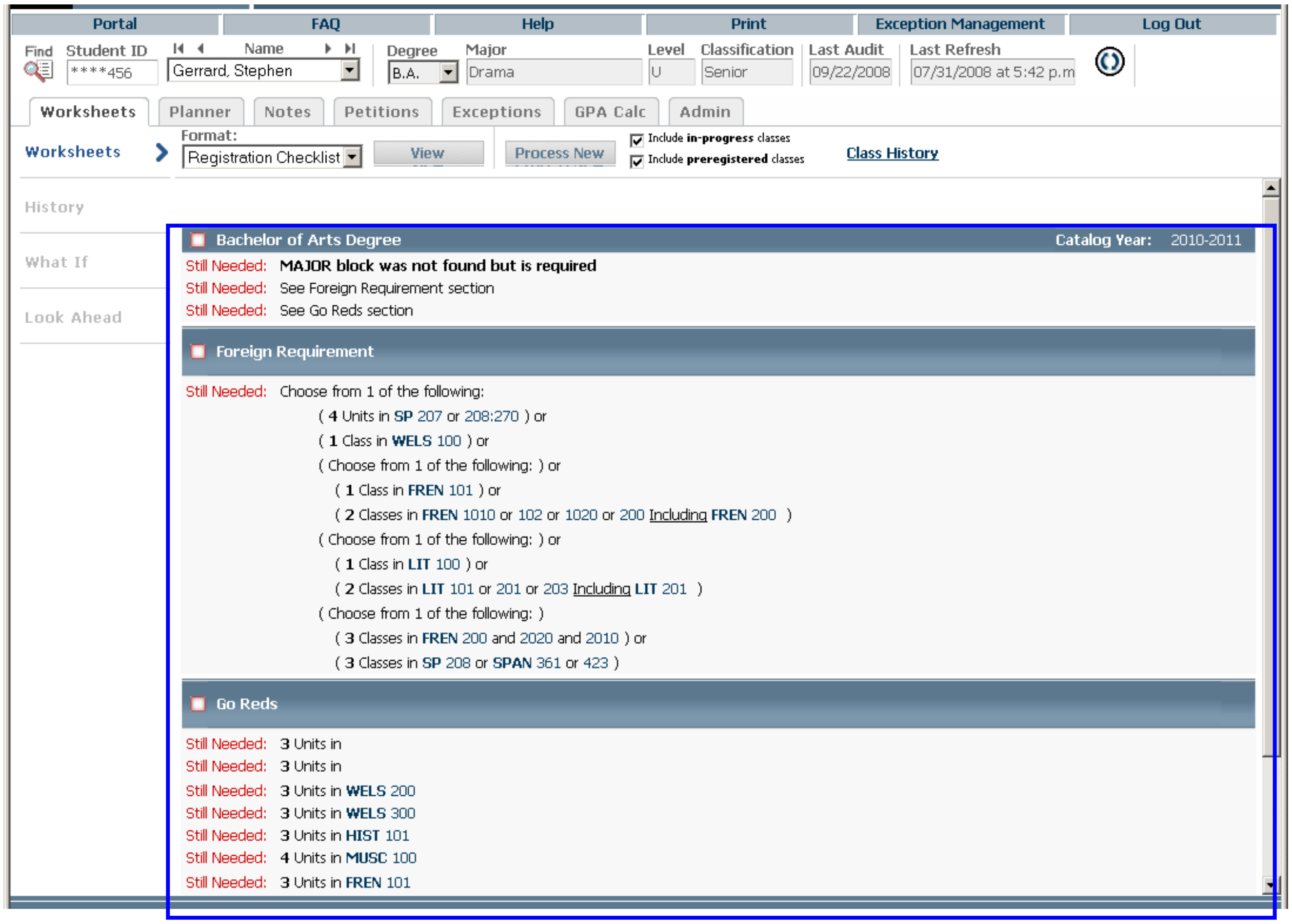


Class History AA036943 as of 09/22/2008 at 17:25

Student	Gerrard, Stephen	Level	Undergraduate School
ID	****456	College	
Advisor 1		Degree	BACHELOR OF ARTS
Advisor 2		Major	Drama
Overall GPA	2.973	Classification	Senior

Winter 2001				
PHIL	**1001**	Introduction to Logic	C	3
Fall 2001				
ENGL	**3100**	Poetry Writing	B	3
MATH	**1030**	College Algebra	A	3
Winter 2002				
COMM	**1000**	Public Speaking	C	3
FIN	**3320**	Financial Markets	A	3
MGMT	**3614**	Organizational Behavior	A	3
STAT	**1000**	Statistics	A	3
Spring 2002				
CS	**3240**	DATA STRUCT/ALGOR	C	4
CS	**3560**	INTR SYSTEMS PROG	B	4
Summer 2003				
CIS	**1270**	PC Fundamentals	B	0
Transferred from		CS 101 - Napier University		
E	**3570**	DATA COMM NETWORK I	C	4.123
ECON	**2301**	Principles of Macroeconomics	B	0
Transferred from		EC 101 - Napier University		
Fall 2003				

DGW_Registration.xsl



Student ID: ****456 | Name: Gerrard, Stephen | Degree: B.A. | Major: Drama | Level: U | Classification: Senior | Last Audit: 09/22/2008 | Last Refresh: 07/31/2008 at 5:42 p.m

Format: Registration Checklist

☑ Include **in-progress** classes
☑ Include **preregistered** classes

Bachelor of Arts Degree — **Catalog Year:** 2010-2011

Still Needed: **MAJOR block was not found but is required**
Still Needed: See Foreign Requirement section
Still Needed: See Go Reds section

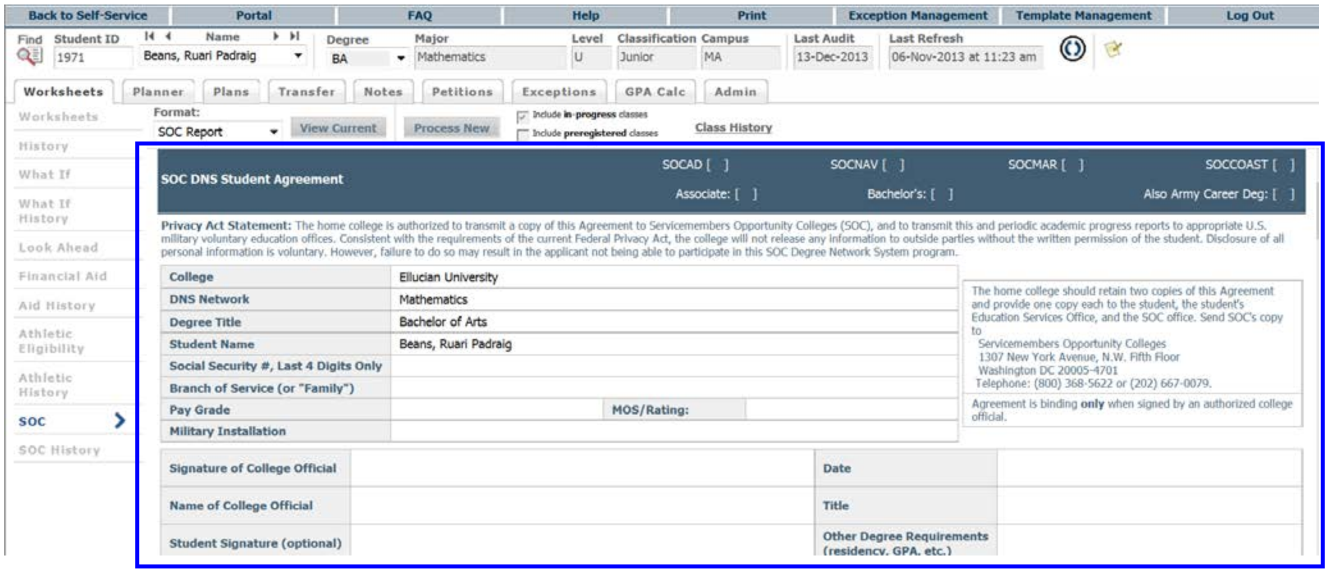
Foreign Requirement

Still Needed: Choose from 1 of the following:
(**4** Units in **SP** 207 or 208:270) or
(**1** Class in **WELS** 100) or
(Choose from 1 of the following:) or
(**1** Class in **FREN** 101) or
(**2** Classes in **FREN** 1010 or 102 or 1020 or 200 Including **FREN** 200)
(Choose from 1 of the following:) or
(**1** Class in **LIT** 100) or
(**2** Classes in **LIT** 101 or 201 or 203 Including **LIT** 201)
(Choose from 1 of the following:)
(**3** Classes in **FREN** 200 and 2020 and 2010) or
(**3** Classes in **SP** 208 or **SPAN** 361 or 423)

Go Reds

Still Needed: **3** Units in
Still Needed: **3** Units in
Still Needed: **3** Units in **WELS** 200
Still Needed: **3** Units in **WELS** 300
Still Needed: **3** Units in **HIST** 101
Still Needed: **4** Units in **MUSC** 100
Still Needed: **3** Units in **FREN** 101

DGW_SOC.xsl



Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit	Last Refresh
1971	Beans, Ruari Padraig	BA	Mathematics	U	Junior	MA	13-Dec-2013	06-Nov-2013 at 11:23 am

Format: SOC Report

SOC DNS Student Agreement

SOCAD [ ] SOCNAV [ ] SOCMAR [ ] SOCCOAST [ ]

Associate: [ ] Bachelor's: [ ] Also Army Career Deg: [ ]

Privacy Act Statement: The home college is authorized to transmit a copy of this Agreement to Servicemembers Opportunity Colleges (SOC), and to transmit this and periodic academic progress reports to appropriate U.S. military voluntary education offices. Consistent with the requirements of the current Federal Privacy Act, the college will not release any information to outside parties without the written permission of the student. Disclosure of all personal information is voluntary. However, failure to do so may result in the applicant not being able to participate in this SOC Degree Network System program.

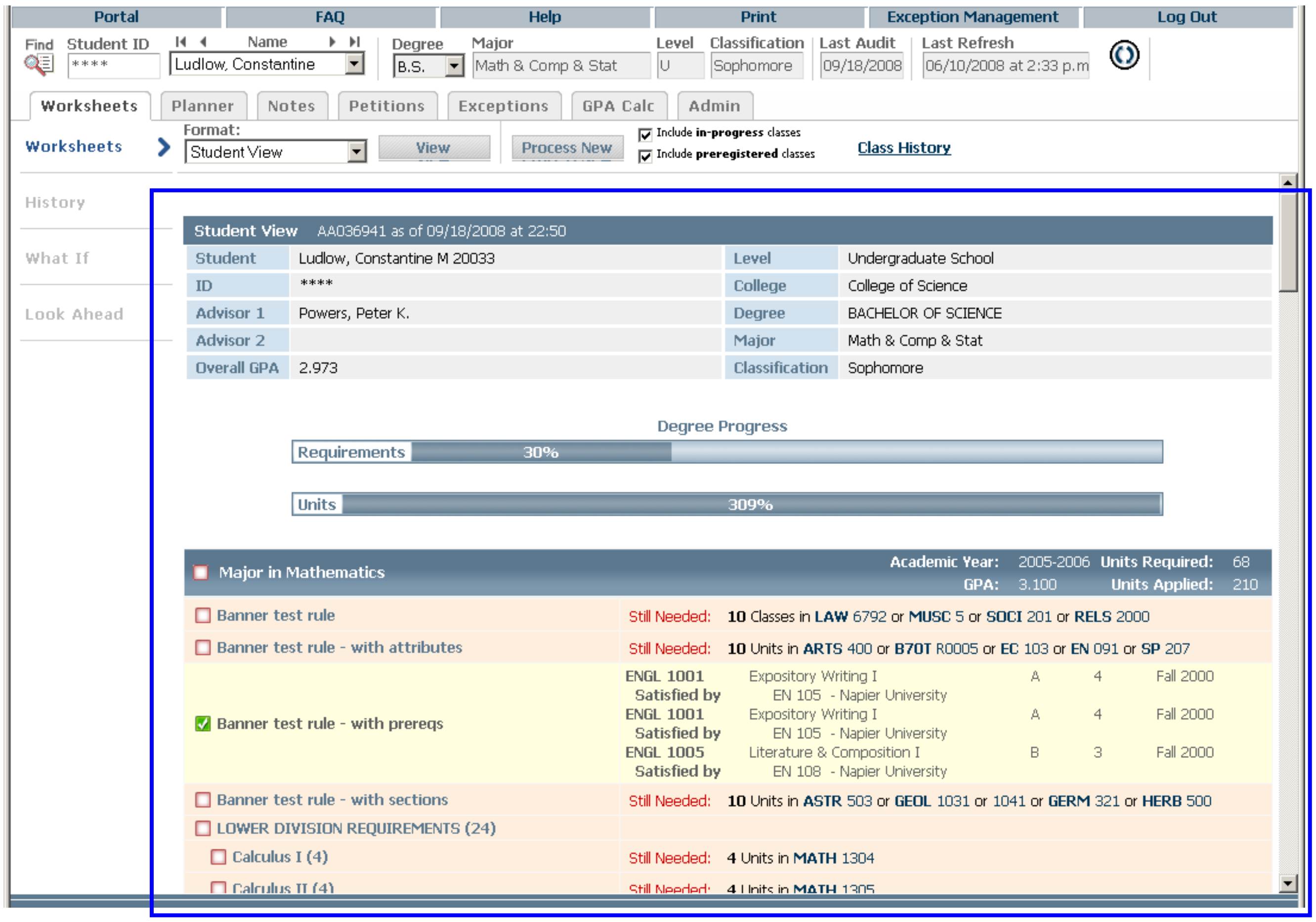
College	Ellucian University		
DNS Network	Mathematics		
Degree Title	Bachelor of Arts		
Student Name	Beans, Ruari Padraig		
Social Security #, Last 4 Digits Only			
Branch of Service (or "Family")			
Pay Grade		MOS/Rating:	
Military Installation			

The home college should retain two copies of this Agreement and provide one copy each to the student, the student's Education Services Office, and the SOC office. Send SOC's copy to
Servicemembers Opportunity Colleges
1307 New York Avenue, N.W. Fifth Floor
Washington DC 20005-4701
Telephone: (800) 368-5622 or (202) 667-0079.

Agreement is binding **only** when signed by an authorized college official.

Signature of College Official		Date	
Name of College Official		Title	
Student Signature (optional)		Other Degree Requirements (residency, GPA, etc.)	

DGW_Report.xsl



Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
****	Ludlow, Constantine	B.S.	Math & Comp & Stat	U	Sophomore	09/18/2008	06/10/2008 at 2:33 p.m

Format: Student View

☑ Include **in-progress** classes
☑ Include **preregistered** classes

Student View AA036941 as of 09/18/2008 at 22:50

Student	Ludlow, Constantine M 20033	Level	Undergraduate School
ID	****	College	College of Science
Advisor 1	Powers, Peter K.	Degree	BACHELOR OF SCIENCE
Advisor 2		Major	Math & Comp & Stat
Overall GPA	2.973	Classification	Sophomore

Degree Progress

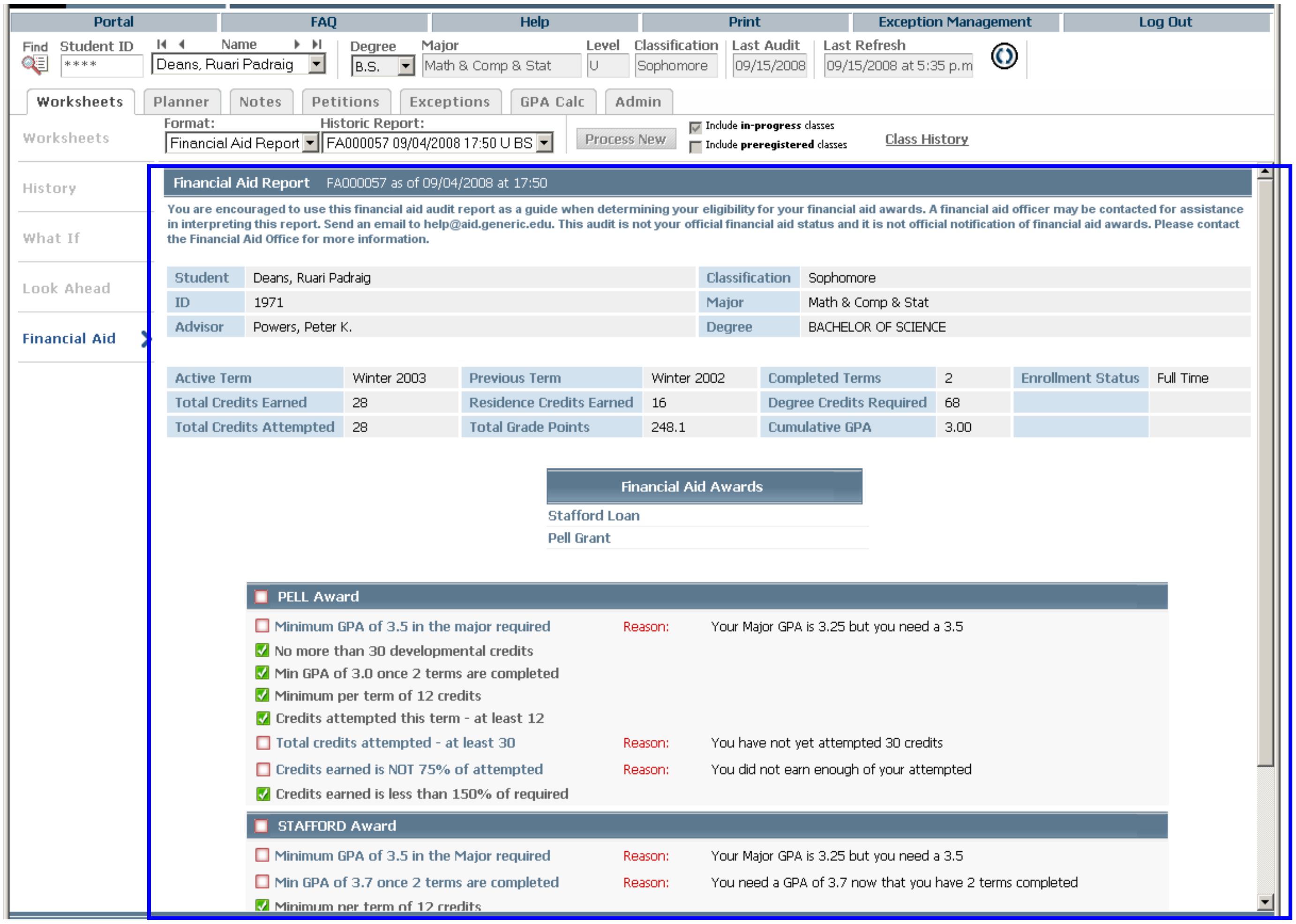
Requirements 30%

Units 309%

Major in Mathematics Academic Year: 2005-2006 Units Required: 68
GPA: 3.100 Units Applied: 210

Banner test rule	Still Needed:	**10** Classes in **LAW** 6792 or **MUSC** 5 or **SOCI** 201 or **RELS** 2000				
Banner test rule - with attributes	Still Needed:	**10** Units in **ARTS** 400 or **B70T** R0005 or **EC** 103 or **EN** 091 or **SP** 207				
✓ Banner test rule - with prereqs	ENGL 1001	Expository Writing I	A	4	Fall 2000
	Satisfied by	EN 105 - Napier University				
	ENGL 1001	Expository Writing I	A	4	Fall 2000
	Satisfied by	EN 105 - Napier University				
	ENGL 1005	Literature & Composition I	B	3	Fall 2000
	Satisfied by	EN 108 - Napier University				
Banner test rule - with sections	Still Needed:	**10** Units in **ASTR** 503 or **GEOL** 1031 or 1041 or **GERM** 321 or **HERB** 500				
LOWER DIVISION REQUIREMENTS (24)						
Calculus I (4)	Still Needed:	**4** Units in **MATH** 1304				
Calculus II (4)	Still Needed:	**4** Units in **MATH** 1305				

DGW_Aid_Report.xsl



Student ID: **** | Name: Deans, Ruari Padraig | Degree: B.S. | Major: Math & Comp & Stat | Level: U | Classification: Sophomore | Last Audit: 09/15/2008 | Last Refresh: 09/15/2008 at 5:35 p.m

Format: Financial Aid Report | Historic Report: FA000057 09/04/2008 17:50 U BS

✓ Include **in-progress** classes
Include **preregistered** classes

Financial Aid Report FA000057 as of 09/04/2008 at 17:50

You are encouraged to use this financial aid audit report as a guide when determining your eligibility for your financial aid awards. A financial aid officer may be contacted for assistance in interpreting this report. Send an email to help@aid.generic.edu. This audit is not your official financial aid status and it is not official notification of financial aid awards. Please contact the Financial Aid Office for more information.

Student	Deans, Ruari Padraig	Classification	Sophomore
ID	1971	Major	Math & Comp & Stat
Advisor	Powers, Peter K.	Degree	BACHELOR OF SCIENCE

Active Term	Winter 2003	Previous Term	Winter 2002	Completed Terms	2	Enrollment Status	Full Time
Total Credits Earned	28	Residence Credits Earned	16	Degree Credits Required	68		
Total Credits Attempted	28	Total Grade Points	248.1	Cumulative GPA	3.00		

Financial Aid Awards

Stafford Loan
Pell Grant

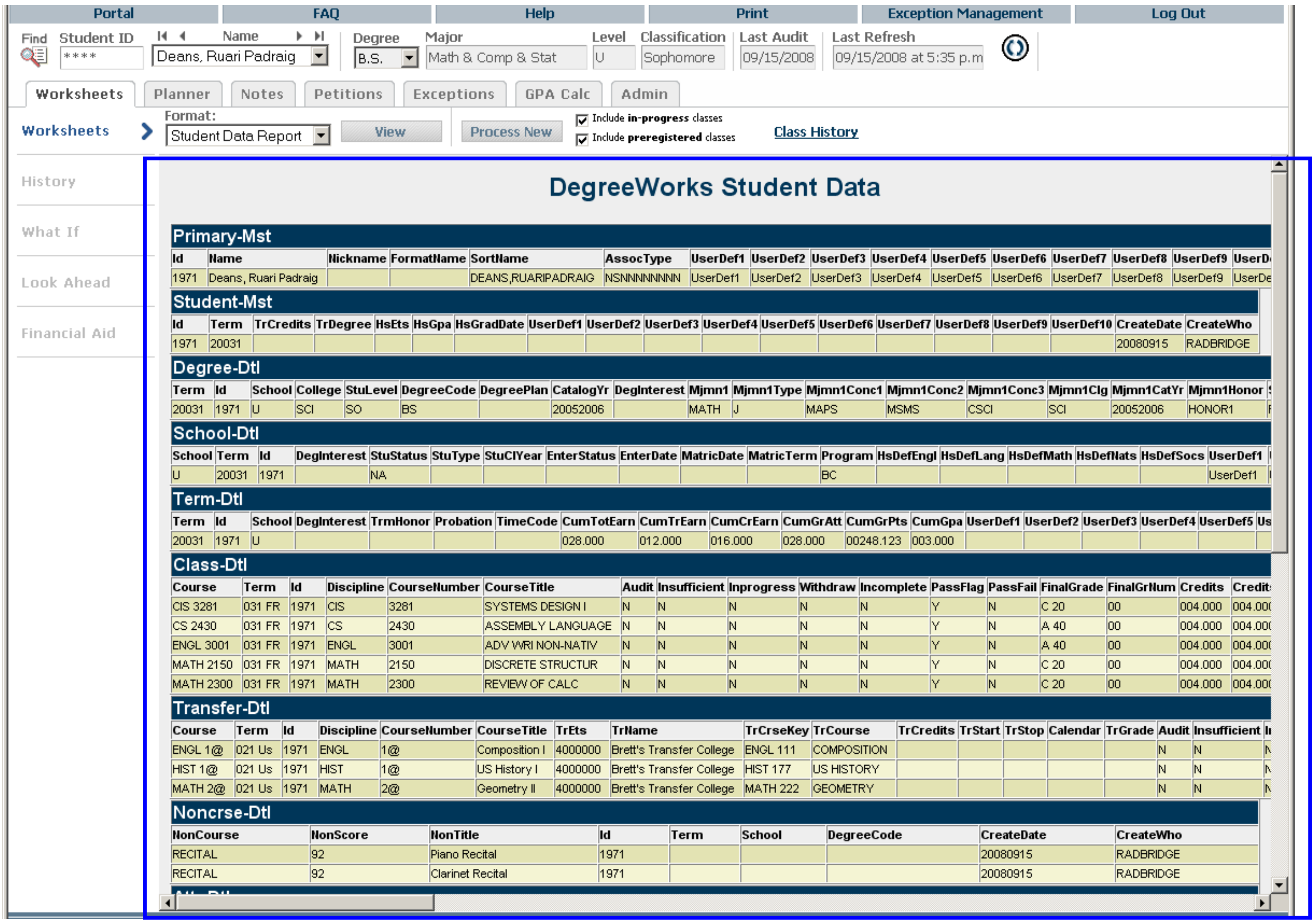
PELL Award

☐ Minimum GPA of 3.5 in the major required	Reason:	Your Major GPA is 3.25 but you need a 3.5
✓ No more than 30 developmental credits		
✓ Min GPA of 3.0 once 2 terms are completed		
✓ Minimum per term of 12 credits		
✓ Credits attempted this term - at least 12		
☐ Total credits attempted - at least 30	Reason:	You have not yet attempted 30 credits
☐ Credits earned is NOT 75% of attempted	Reason:	You did not earn enough of your attempted
✓ Credits earned is less than 150% of required		

STAFFORD Award

☐ Minimum GPA of 3.5 in the Major required	Reason:	Your Major GPA is 3.25 but you need a 3.5
☐ Min GPA of 3.7 once 2 terms are completed	Reason:	You need a GPA of 3.7 now that you have 2 terms completed
✓ Minimum per term of 12 credits		

RADData.xsl



Student ID: ****
Name: Deans, Ruari Padraig
Degree: B.S.
Major: Math & Comp & Stat
Level: U
Classification: Sophomore
Last Audit: 09/15/2008
Last Refresh: 09/15/2008 at 5:35 p.m

Format: Student Data Report

☑ Include **in-progress** classes
☑ Include **preregistered** classes

DegreeWorks Student Data

Primary-Mst

Id	Name	Nickname	FormatName	SortName	AssocType	UserDef1	UserDef2	UserDef3	UserDef4	UserDef5	UserDef6	UserDef7	UserDef8	UserDef9	UserD
1971	Deans, Ruari Padraig			DEANS,RUARIPADRAIG	NSNNNNNNNN	UserDef1	UserDef2	UserDef3	UserDef4	UserDef5	UserDef6	UserDef7	UserDef8	UserDef9	UserDe

Student-Mst

Id	Term	TrCredits	TrDegree	HsEts	HsGpa	HsGradDate	UserDef1	UserDef2	UserDef3	UserDef4	UserDef5	UserDef6	UserDef7	UserDef8	UserDef9	UserDef10	CreateDate	CreateWho
1971	20031																20080915	RADBRIDGE

Degree-Dtl

Term	Id	School	College	StuLevel	DegreeCode	DegreePlan	CatalogYr	DegInterest	Mjmn1	Mjmn1Type	Mjmn1Conc1	Mjmn1Conc2	Mjmn1Conc3	Mjmn1Clg	Mjmn1CatYr	Mjmn1Honor
20031	1971	U	SCI	SO	BS		20052006		MATH	J	MAPS	MSMS	CSCI	SCI	20052006	HONOR1

School-Dtl

School	Term	Id	DegInterest	StuStatus	StuType	StuClYear	EnterStatus	EnterDate	MatricDate	MatricTerm	Program	HsDefEngl	HsDefLang	HsDefMath	HsDefNats	HsDefSocs	UserDef1
U	20031	1971		NA							BC						UserDef1

Term-Dtl

Term	Id	School	DegInterest	TrmHonor	Probation	TimeCode	CumTotEarn	CumTrEarn	CumCrEarn	CumGrAtt	CumGrPts	CumGpa	UserDef1	UserDef2	UserDef3	UserDef4	UserDef5	Us
20031	1971	U					028.000	012.000	016.000	028.000	00248.123	003.000						

Class-Dtl

Course	Term	Id	Discipline	CourseNumber	CourseTitle	Audit	Insufficient	Inprogress	Withdraw	Incomplete	PassFlag	PassFail	FinalGrade	FinalGrNum	Credits	Credit
CIS 3281	031 FR	1971	CIS	3281	SYSTEMS DESIGN I	N	N	N	N	N	Y	N	C 20	00	004.000	004.00
CS 2430	031 FR	1971	CS	2430	ASSEMBLY LANGUAGE	N	N	N	N	N	Y	N	A 40	00	004.000	004.00
ENGL 3001	031 FR	1971	ENGL	3001	ADV WRI NON-NATIV	N	N	N	N	N	Y	N	A 40	00	004.000	004.00
MATH 2150	031 FR	1971	MATH	2150	DISCRETE STRUCTUR	N	N	N	N	N	Y	N	C 20	00	004.000	004.00
MATH 2300	031 FR	1971	MATH	2300	REVIEW OF CALC	N	N	N	N	N	Y	N	C 20	00	004.000	004.00

Transfer-Dtl

Course	Term	Id	Discipline	CourseNumber	CourseTitle	TrEts	TrName	TrCrseKey	TrCourse	TrCredits	TrStart	TrStop	Calendar	TrGrade	Audit	Insufficient
ENGL 1@	021 Us	1971	ENGL	1@	Composition I	4000000	Brett's Transfer College	ENGL 111	COMPOSITION						N	N
HIST 1@	021 Us	1971	HIST	1@	US History I	4000000	Brett's Transfer College	HIST 177	US HISTORY						N	N
MATH 2@	021 Us	1971	MATH	2@	Geometry II	4000000	Brett's Transfer College	MATH 222	GEOMETRY						N	N

Noncrse-Dtl

NonCourse	NonScore	NonTitle	Id	Term	School	DegreeCode	CreateDate	CreateWho
RECITAL	92	Piano Recital	1971				20080915	RADBRIDGE
RECITAL	92	Clarinet Recital	1971				20080915	RADBRIDGE

Localizing the Student Educational Planner

The Student Educational Planner (SEP) interface utilizes a number of XSL stylesheets. Here is the complete list.

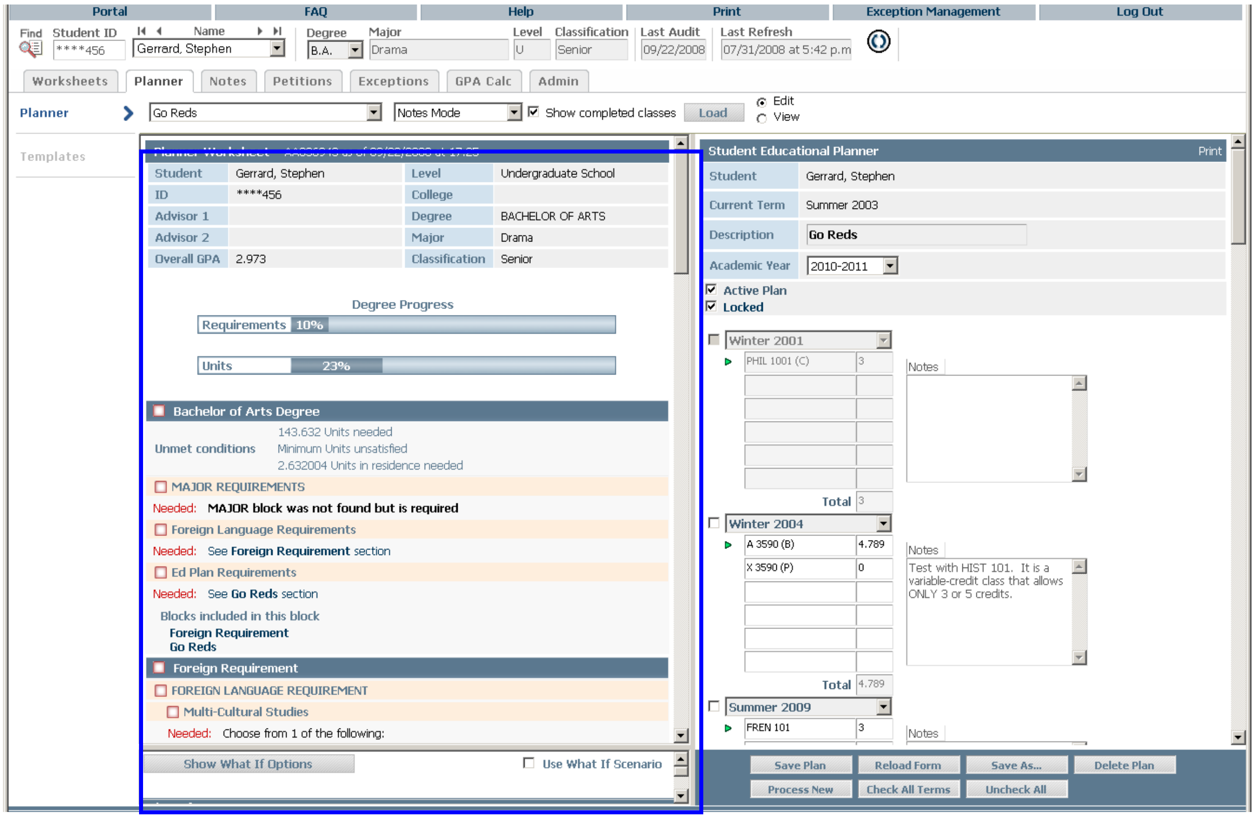
Note: This section provides details for the “classic” Student Educational Planner. For information about the new-generation SEP, see the *Student Educational Planner Administration Guide*.

XSL	**Where it is used**
AuditSEP.xsl	Student Educational Planner audit report (left frame)
PlanSEP.xsl	Student Educational Planner “View Mode”
PlanSEPCompareView.xsl	Student Educational Planner “Planned vs. Taken”
SEP_Context.xsl	Student Educational Planner context frame (containing plan name, mode, and Load button)
SEP_Save.xsl	Student Educational Planner return status after save attempt (typically not modified)
SEP_TemplateSearch.xsl	Student Educational Planner searching for template (after clicking on “Load in a pre-defined plan”)
SEP_TemplateSearchResults.xsl	Student Educational Planner search results (bottom frame)
TMP_Buttons.xsl	SEP Template buttons frame
TMP_Context.xsl	SEP Template context frame (containing template search results, “mode”, and Load button)
TMP_Save.xsl	SEP Template return status after save attempt (typically not modified)
TMP_Search.xsl	SEP Template search frame containing picklists for search criteria

In order to localize SEP you must make modifications to DashboardStyles.css. This is where color changes are made. To localize content, see the XSL files above.

One exception to the SEP technology is the use of XML Data Islands. When you choose to edit a SEP plan (the right frame), you are invoking the JavaScript file “DGW_SEPEdit.js.” This JavaScript code parses through the Plan data, which is XML embedded within the HTML document itself. Rather than using XSL to render the XML as HTML, the use of XML Data Islands is more robust. It allows you to parse through the XML and generate more JavaScript, which is how the SEP Plan form is created.

AuditSEP.xsl



Find | Student ID ****456 | Name Gerrard, Stephen | Degree B.A. | Major Drama | Level U | Classification Senior | Last Audit 09/22/2008 | Last Refresh 07/31/2008 at 5:42 p.m

Planner: Go Reds | Notes Mode | ☑ Show completed classes | Load | [✓ Edit] View

Student	Gerrard, Stephen	Level	Undergraduate School
ID	****456	College	
Advisor 1		Degree	BACHELOR OF ARTS
Advisor 2		Major	Drama
Overall GPA	2.973	Classification	Senior

Degree Progress

Requirements 10%

Units 23%

Bachelor of Arts Degree

Unmet conditions
143.632 Units needed
Minimum Units unsatisfied
2.632004 Units in residence needed

MAJOR REQUIREMENTS
Needed: MAJOR block was not found but is required

Foreign Language Requirements
Needed: See Foreign Requirement section

Ed Plan Requirements
Needed: See Go Reds section

Blocks included in this block
Foreign Requirement
Go Reds

Foreign Requirement

FOREIGN LANGUAGE REQUIREMENT

Multi-Cultural Studies
Needed: Choose from 1 of the following:

Show What If Options | ☐ Use What If Scenario

Student Educational Planner

Student	Gerrard, Stephen
Current Term	Summer 2003
Description	Go Reds
Academic Year	2010-2011

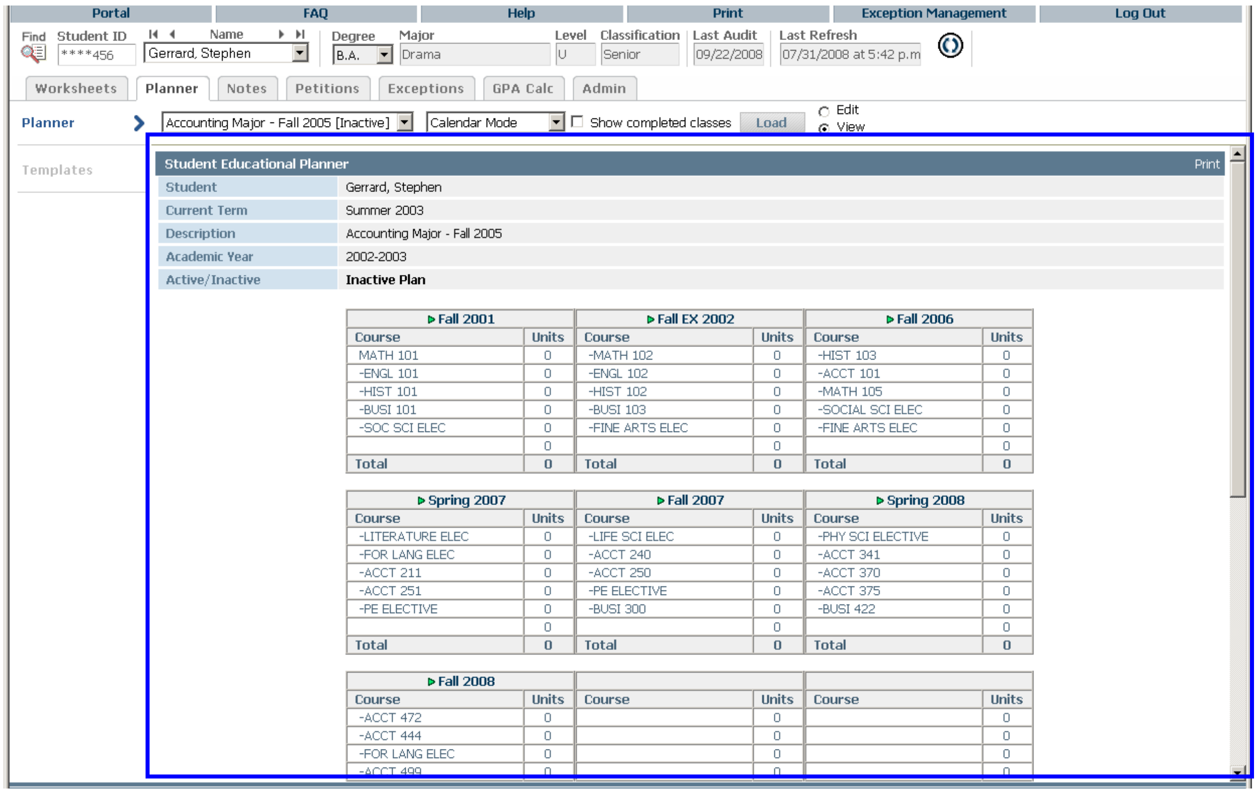
☑ Active Plan
☑ Locked

Winter 2001
PHIL 1001 (C) 3
Total 3

Winter 2004
A 3590 (B) 4.789
X 3590 (P) 0
Notes: Test with HIST 101. It is a variable-credit class that allows ONLY 3 or 5 credits.
Total 4.789

Summer 2009
FREN 101 3

PlanSEP.xsl (used by both Notes Mode and Calendar Mode for the “View” option)



Find	Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
	****456	Gerrard, Stephen	B.A.	Drama	U	Senior	09/22/2008	07/31/2008 at 5:42 p.m

Accounting Major - Fall 2005 [Inactive] | Calendar Mode | ☐ Show completed classes | Edit [✓ View]

Student Educational Planner

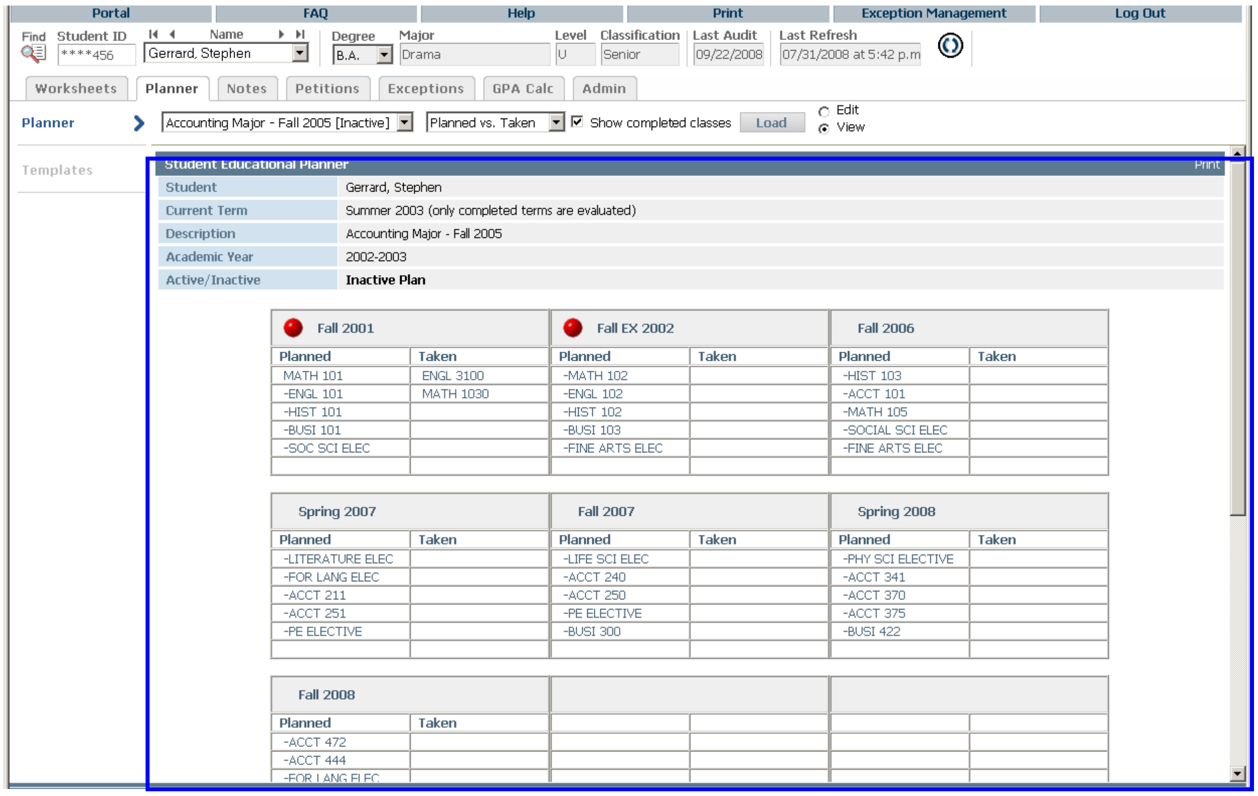
Student	Gerrard, Stephen
Current Term	Summer 2003
Description	Accounting Major - Fall 2005
Academic Year	2002-2003
Active/Inactive	Inactive Plan

Fall 2001		Fall EX 2002		Fall 2006	
Course	Units	Course	Units	Course	Units
MATH 101	0	-MATH 102	0	-HIST 103	0
-ENGL 101	0	-ENGL 102	0	-ACCT 101	0
-HIST 101	0	-HIST 102	0	-MATH 105	0
-BUSI 101	0	-BUSI 103	0	-SOCIAL SCI ELEC	0
-SOC SCI ELEC	0	-FINE ARTS ELEC	0	-FINE ARTS ELEC	0
	0		0		0
Total	0	Total	0	Total	0

Spring 2007		Fall 2007		Spring 2008	
Course	Units	Course	Units	Course	Units
-LITERATURE ELEC	0	-LIFE SCI ELEC	0	-PHY SCI ELECTIVE	0
-FOR LANG ELEC	0	-ACCT 240	0	-ACCT 341	0
-ACCT 211	0	-ACCT 250	0	-ACCT 370	0
-ACCT 251	0	-PE ELECTIVE	0	-ACCT 375	0
-PE ELECTIVE	0	-BUSI 300	0	-BUSI 422	0
	0		0		0
Total	0	Total	0	Total	0

Fall 2008					
Course	Units	Course	Units	Course	Units
-ACCT 472	0		0		0
-ACCT 444	0		0		0
-FOR LANG ELEC	0		0		0
-ACCT 499	0		0		0

PlanSEPCompareView.xsl



Portal | FAQ | Help | Print | Exception Management | Log Out

Find	Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
	****456	Gerrard, Stephen	B.A.	Drama	U	Senior	09/22/2008	07/31/2008 at 5:42 p.m

Worksheets | Planner | Notes | Petitions | Exceptions | GPA Calc | Admin

Planner — Accounting Major - Fall 2005 [Inactive] — Planned vs. Taken — [✓] Show completed classes — Load — Edit [✓ View]

Templates

Student Educational Planner Print

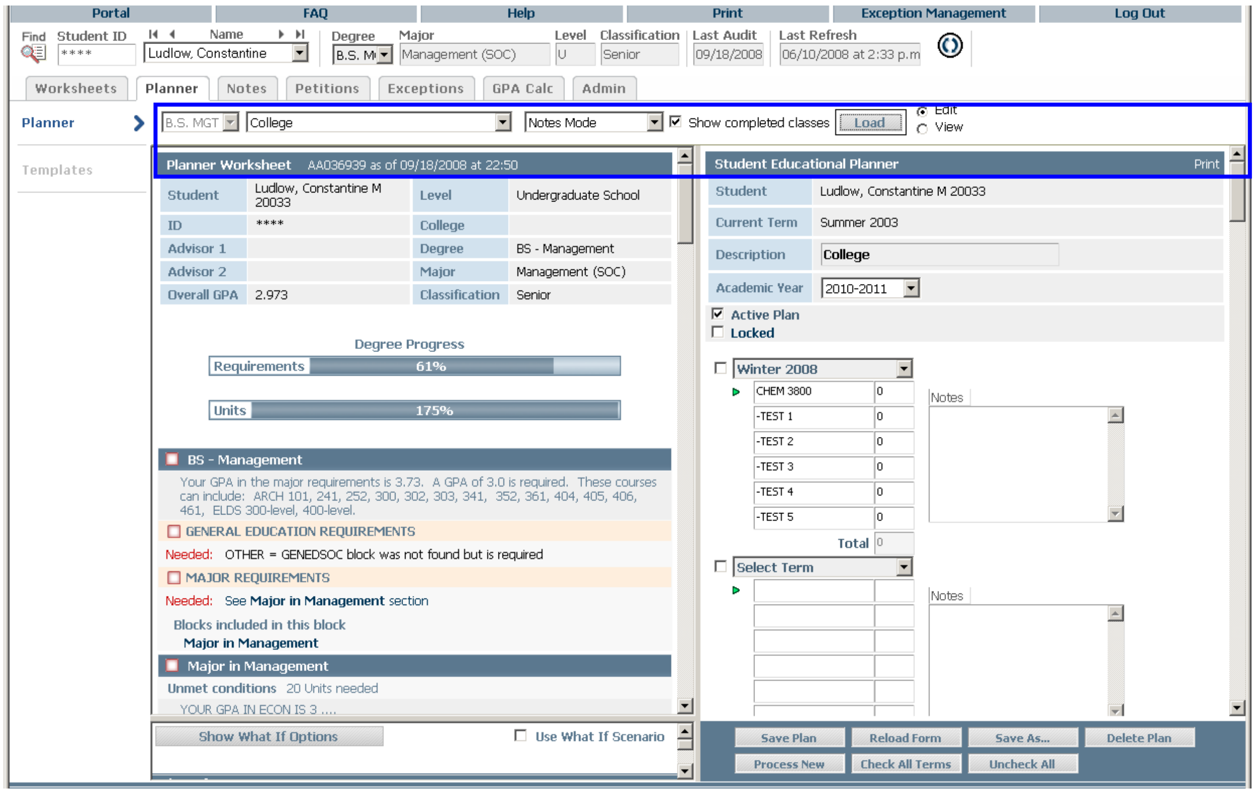
Student	Gerrard, Stephen
Current Term	Summer 2003 (only completed terms are evaluated)
Description	Accounting Major - Fall 2005
Academic Year	2002-2003
Active/Inactive	**Inactive Plan**

● Fall 2001		● Fall EX 2002		Fall 2006	
Planned	Taken	Planned	Taken	Planned	Taken
MATH 101	ENGL 3100	-MATH 102		-HIST 103	
-ENGL 101	MATH 1030	-ENGL 102		-ACCT 101	
-HIST 101		-HIST 102		-MATH 105	
-BUSI 101		-BUSI 103		-SOCIAL SCI ELEC	
-SOC SCI ELEC		-FINE ARTS ELEC		-FINE ARTS ELEC	

Spring 2007		Fall 2007		Spring 2008	
Planned	Taken	Planned	Taken	Planned	Taken
-LITERATURE ELEC		-LIFE SCI ELEC		-PHY SCI ELECTIVE	
-FOR LANG ELEC		-ACCT 240		-ACCT 341	
-ACCT 211		-ACCT 250		-ACCT 370	
-ACCT 251		-PE ELECTIVE		-ACCT 375	
-PE ELECTIVE		-BUSI 300		-BUSI 422	

Fall 2008					
Planned	Taken				
-ACCT 472					
-ACCT 444					
-FOR LANG ELEC					

SEP_Context.xsl



Portal | FAQ | Help | Print | Exception Management | Log Out

Find	Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
	****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

Worksheets | Planner | Notes | Petitions | Exceptions | GPA Calc | Admin

Planner

Templates

B.S. MGT | College | Notes Mode | [✓] Show completed classes | Load | [✓ Edit] View

Planner Worksheet AA036939 as of 09/18/2008 at 22:50

Student	Ludlow, Constantine M 20033	Level	Undergraduate School
ID	****	College	
Advisor 1		Degree	BS - Management
Advisor 2		Major	Management (SOC)
Overall GPA	2.973	Classification	Senior

Degree Progress

Requirements 61%

Units 175%

BS - Management

Your GPA in the major requirements is 3.73. A GPA of 3.0 is required. These courses can include: ARCH 101, 241, 252, 300, 302, 303, 341, 352, 361, 404, 405, 406, 461, ELDS 300-level, 400-level.

GENERAL EDUCATION REQUIREMENTS

Needed: OTHER = GENEDSOC block was not found but is required

MAJOR REQUIREMENTS

Needed: See **Major in Management** section

Blocks included in this block

Major in Management

Major in Management

Unmet conditions 20 Units needed

YOUR GPA IN ECON IS 3 ....

Show What If Options | Use What If Scenario

Student Educational Planner Print

Student	Ludlow, Constantine M 20033
Current Term	Summer 2003
Description	College
Academic Year	2010-2011

[✓] Active Plan

Locked

Winter 2008

CHEM 3800	0
-TEST 1	0
-TEST 2	0
-TEST 3	0
-TEST 4	0
-TEST 5	0
Total	0

Notes

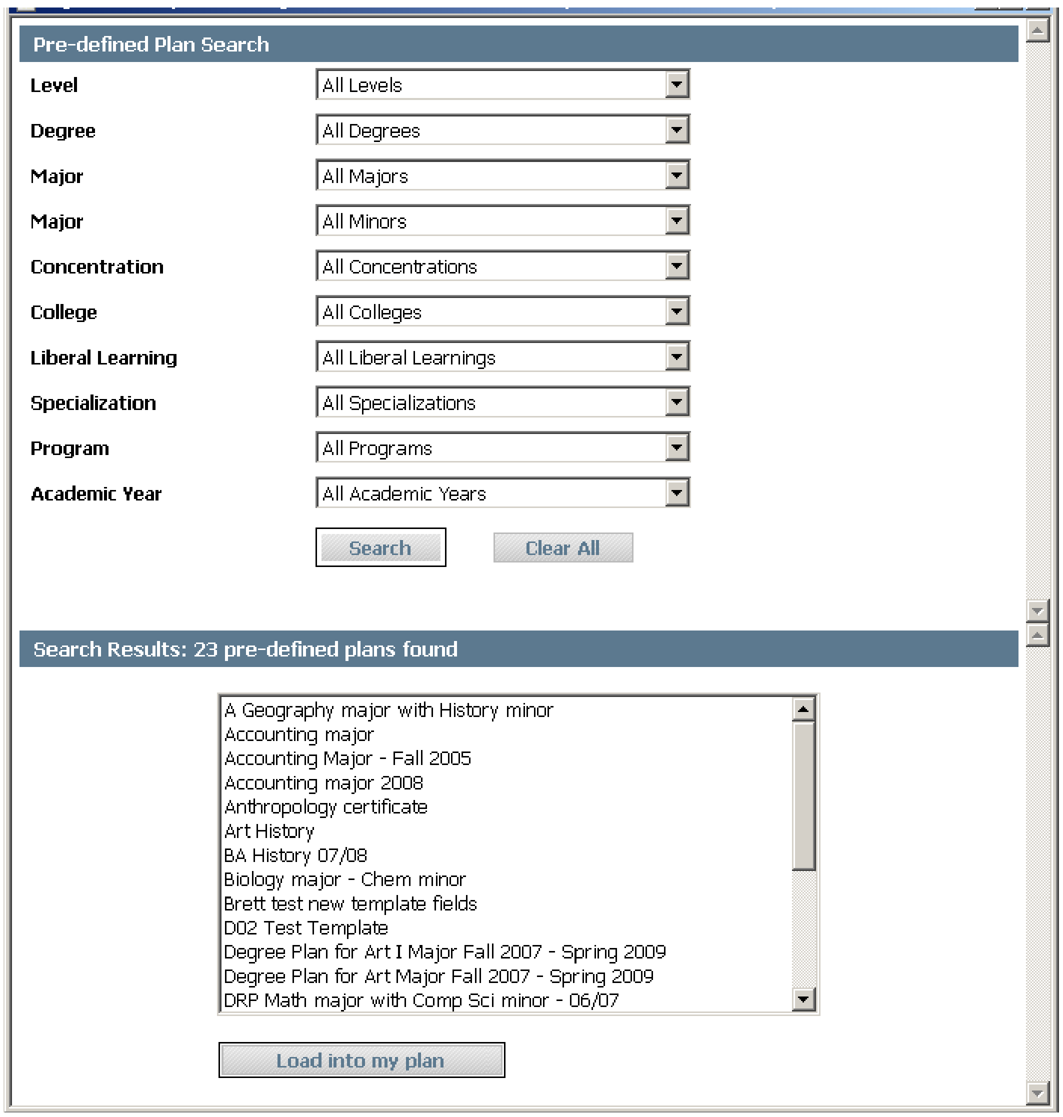
Select Term

Notes

Save Plan | Reload Form | Save As... | Delete Plan

Process New | Check All Terms | Uncheck All

SEP_TemplateSearch.xsl (top), SEP_TemplateSearchResults.xsl (bottom)



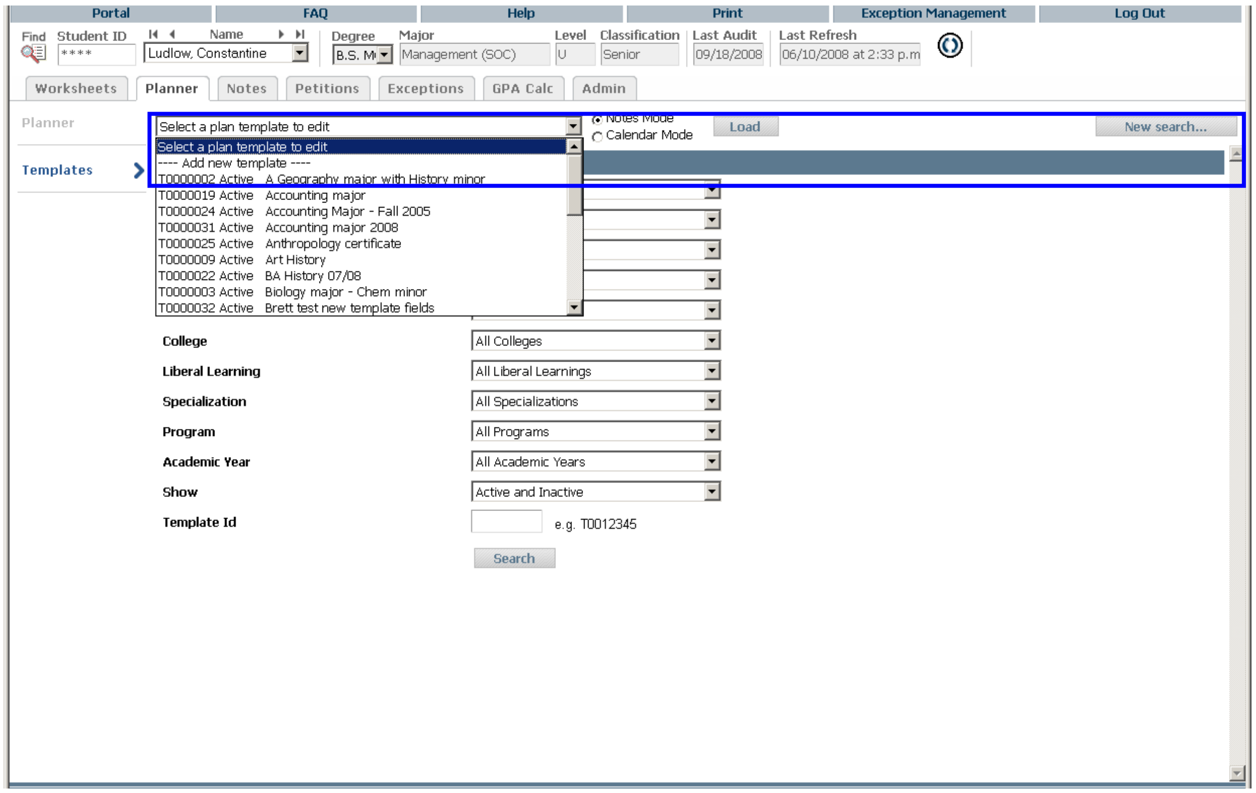
Pre-defined Plan Search

Level	All Levels
Degree	All Degrees
Major	All Majors
Major	All Minors
Concentration	All Concentrations
College	All Colleges
Liberal Learning	All Liberal Learnings
Specialization	All Specializations
Program	All Programs
Academic Year	All Academic Years

Search Results: 23 pre-defined plans found

A Geography major with History minor
Accounting major
Accounting Major - Fall 2005
Accounting major 2008
Anthropology certificate
Art History
BA History 07/08
Biology major - Chem minor
Brett test new template fields
D02 Test Template
Degree Plan for Art I Major Fall 2007 - Spring 2009
Degree Plan for Art Major Fall 2007 - Spring 2009
DRP Math major with Comp Sci minor - 06/07

TMP_Context.xsl



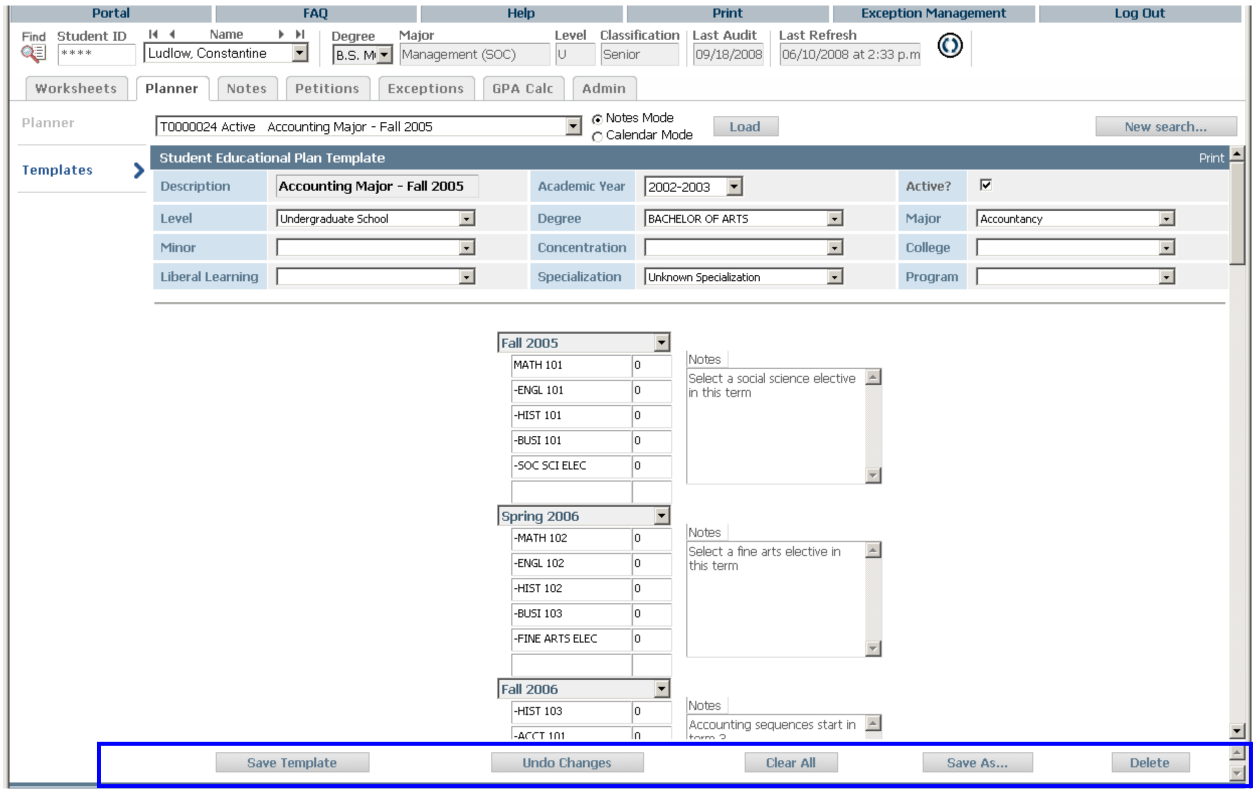
Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

Select a plan template to edit

- Select a plan template to edit
- ---- Add new template ----
- T0000002 Active A Geography major with History minor
- T0000019 Active Accounting major
- T0000024 Active Accounting Major - Fall 2005
- T0000031 Active Accounting major 2008
- T0000025 Active Anthropology certificate
- T0000009 Active Art History
- T0000022 Active BA History 07/08
- T0000003 Active Biology major - Chem minor
- T0000032 Active Brett test new template fields

College	All Colleges
Liberal Learning	All Liberal Learnings
Specialization	All Specializations
Program	All Programs
Academic Year	All Academic Years
Show	Active and Inactive
Template Id	e.g. T0012345

TMP_Buttons.xsl



Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

T0000024 Active Accounting Major - Fall 2005 [✓ Notes Mode] Calendar Mode

Student Educational Plan Template

Description	Accounting Major - Fall 2005	Academic Year	2002-2003	Active?	✓
Level	Undergraduate School	Degree	BACHELOR OF ARTS	Major	Accountancy
Minor		Concentration		College	
Liberal Learning		Specialization	Unknown Specialization	Program	

Fall 2005

MATH 101	0
-ENGL 101	0
-HIST 101	0
-BUSI 101	0
-SOC SCI ELEC	0

Notes: Select a social science elective in this term

Spring 2006

-MATH 102	0
-ENGL 102	0
-HIST 102	0
-BUSI 103	0
-FINE ARTS ELEC	0

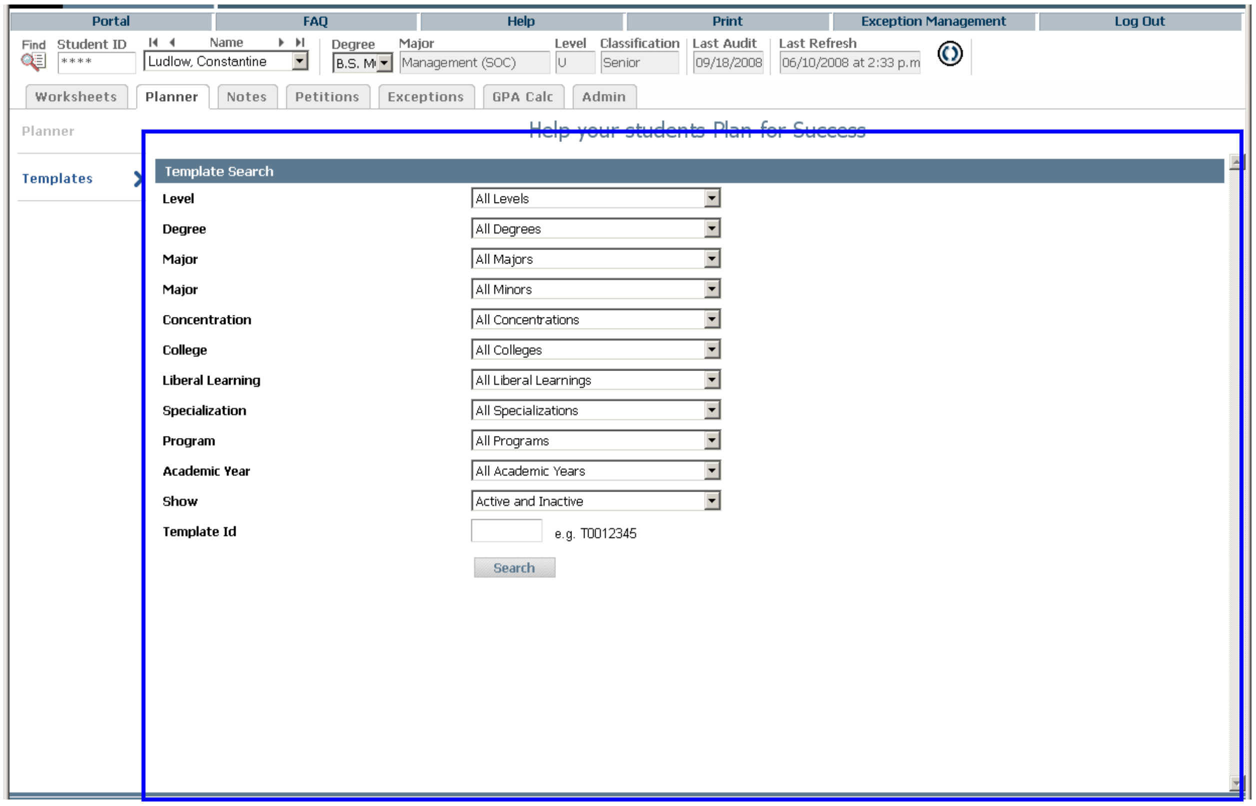
Notes: Select a fine arts elective in this term

Fall 2006

-HIST 103	0
-ACCT 101	0

Notes: Accounting sequences start in term 3

TMP_Search.xsl



Student ID: ****
Name: Ludlow, Constantine
Degree: B.S. M
Major: Management (SOC)
Level: U
Classification: Senior
Last Audit: 09/18/2008
Last Refresh: 06/10/2008 at 2:33 p.m

Template Search

Level	All Levels
Degree	All Degrees
Major	All Majors
Major	All Minors
Concentration	All Concentrations
College	All Colleges
Liberal Learning	All Liberal Learnings
Specialization	All Specializations
Program	All Programs
Academic Year	All Academic Years
Show	Active and Inactive
Template Id	e.g. T0012345

Search

Special Topic: Reintegrating Localizations

It is important to note that you are responsible for reintegrating your localizations when processing an update to Degree Works. Because they are stored in the database, your localizations will not be overwritten, but steps should be taken to ensure that you merge your localizations with the latest version of the record to take advantage of new features and functionality.

Before the update: Use Controller to set all the localization.*.enable settings to false. This will ignore any Shepherd Scripts, CSS, XSL, properties and image localizations you have made and use the baseline versions in all Degree Works applications. This is a good way to test the update and verify that the applications work without the the layer of complexity localizations can add. Alternatively, you may set the Enabled flag for your localized records to false, which will direct all Degree Works applications to use the baseline version instead.

After the update: You will need to integrate all of your localizations with the new baseline versions delivered in the update. There are several ways to approach this. One option is to copy and paste your localized record and the baseline record from the Composer editor into an external tool to "diff" between the two records. Again using copy and paste in the Composer editor, you may choose to simply add the new lines into your existing localized record, or overwrite your localized record with the entire new baseline and reapply your localizations.

After reintegrating your localizations, you will need to enable them either by setting all the localization.*.enable settings to true, or setting the Enabled flag on each localized record to true.

For additional information on how to manage your localizations, please see the *Degree Works Composer Administrative Guide*.

Special Topic: Shepherd Scripts

Not all source code in the Degree Works web interface comes from the webapp directory in the servlet war file. In fact, most comes from Shepherd Scripts, which are stored in the shp_composer_mst database table. On startup of the web daemons these scripts are unloaded to the admin/web07 directory. What are they? Shepherd Scripts are HTML and JavaScript code that contain special commands that are interpreted by the Degree Works software. These commands are written in a proprietary format. The Degree Works web interface is designed with the intent that any "look and feel" localization you may require will be made to the web files. The colors, images, and styles are all configurable in the servlet war file, but much of the content is controlled in the Shepherd Scripts.

The Composer interface is used to localize and manage Shepherd Scripts. For additional information on how to localize your scripts, please see the *Degree Works Composer Administrative Guide*.

Using UCX-SCR001 Literals in the Web Interface

You may change the Description for some of the UCX-SCR001 items and the labels on the web will change. These include: Discipline, Number, CatYr, School, Degree, Level, Major, Minor, Concentration, Status, Conc, Spec, Program, College, Credits, Libl, and ID.

Please see the Responsive Dashboard Administration Guide for configuring the Responsive Dashboard; it does not use UCX-SCR001.

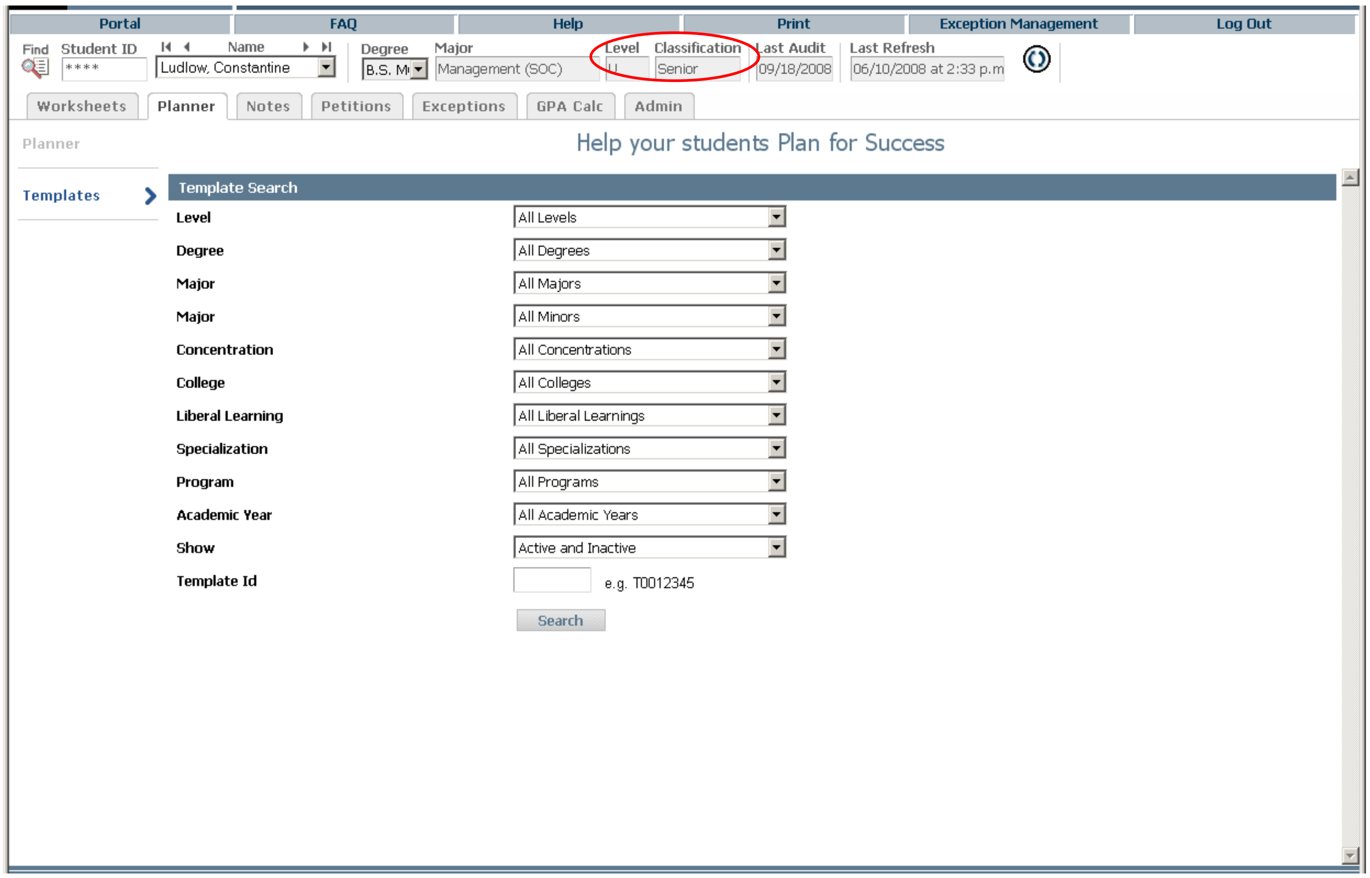
(For Banner schools some of them are appropriately on installation.)

Examples of how these literals are used in the web interface follow:

STUDENT CONTEXT AREA

The Student Context area uses the Student ID, Degree, Major School, and level literals from UCX-SCR001.

For Banner we should see “Level” instead of “School” and “Class Standing” instead of “Level”



Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
****	Ludlow, Constantine	B.S. M	Management (SOC)	U	Senior	09/18/2008	06/10/2008 at 2:33 p.m

Level	All Levels
Degree	All Degrees
Major	All Majors
Major	All Minors
Concentration	All Concentrations
College	All Colleges
Liberal Learning	All Liberal Learnings
Specialization	All Specializations
Program	All Programs
Academic Year	All Academic Years
Show	Active and Inactive
Template Id	e.g. T0012345

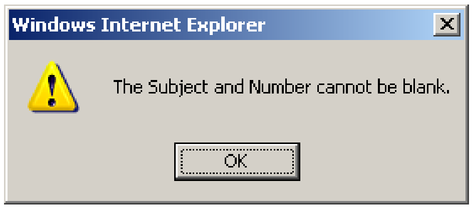
WHAT-IF page:

All literals should come from UCX-SCR001.

For Banner we should see **Level** instead of School and **Academic Year** instead of Catalog Year and

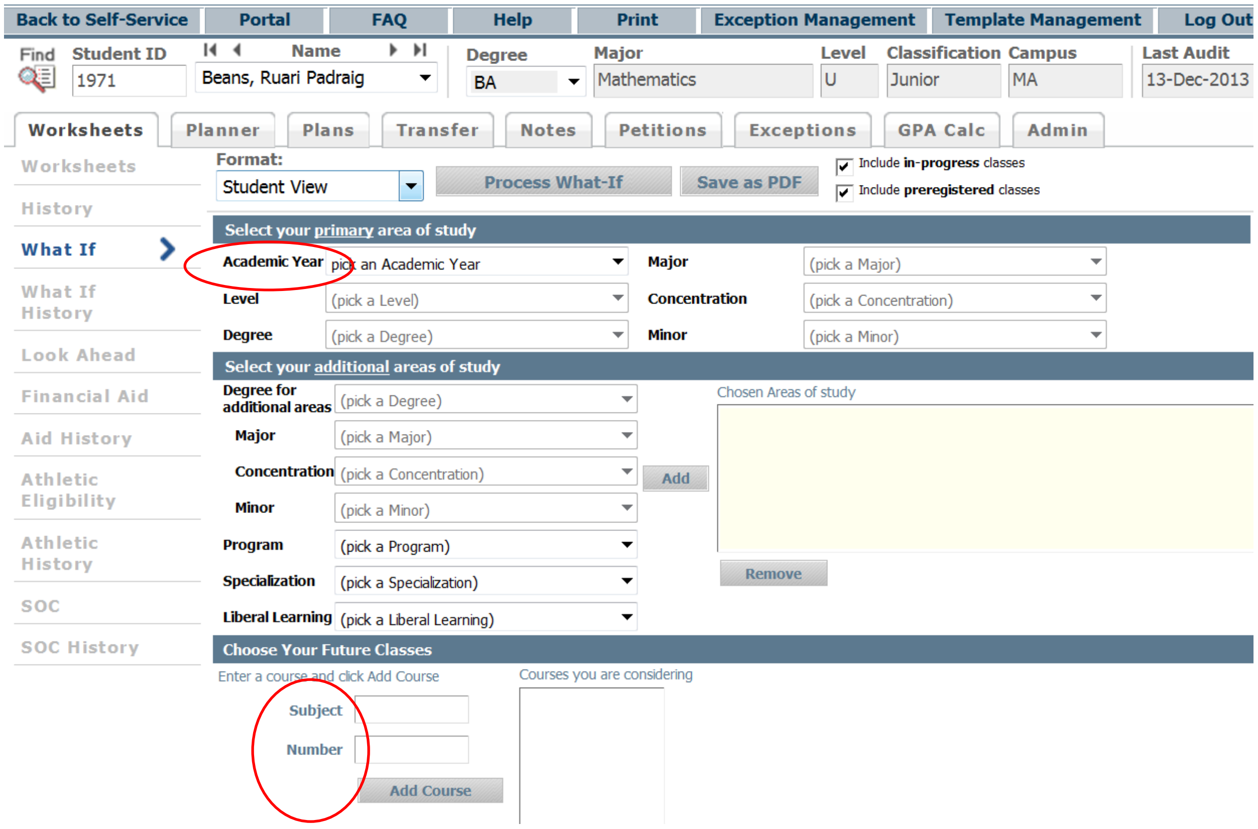
Subject instead of Discipline

Test: plug in a discipline but don't type in a number – click **Add Course** – the error message should use the discipline and number literals from UCX-SCR001 that match the screen



Windows Internet Explorer

The Subject and Number cannot be blank.



Find	Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit
	1971	Beans, Ruari Padraig	BA	Mathematics	U	Junior	MA	13-Dec-2013

Format: Student View

✓ Include **in-progress** classes
✓ Include **preregistered** classes

Select your primary area of study

Academic Year	pick an Academic Year	Major	(pick a Major)
Level	(pick a Level)	Concentration	(pick a Concentration)
Degree	(pick a Degree)	Minor	(pick a Minor)

Select your additional areas of study

Degree for additional areas	(pick a Degree)
Major	(pick a Major)
Concentration	(pick a Concentration)
Minor	(pick a Minor)
Program	(pick a Program)
Specialization	(pick a Specialization)
Liberal Learning	(pick a Liberal Learning)

Choose Your Future Classes

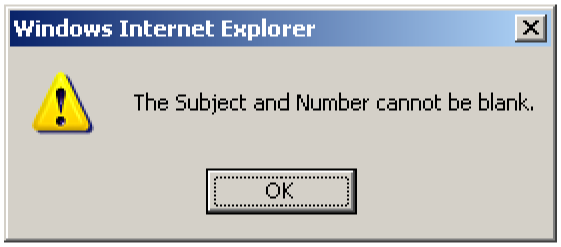
Enter a course and click Add Course

Subject

Number

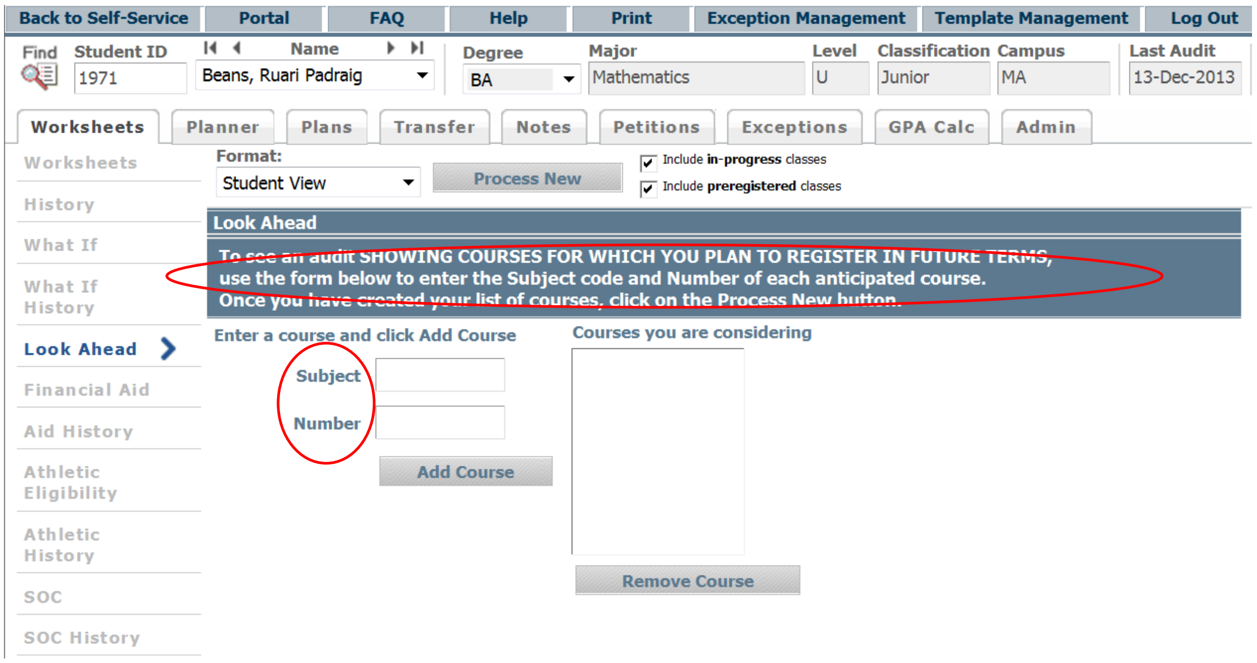
LOOK AHEAD page

Same test as on what-if; discipline/number should be subject/number for Banner.



Windows Internet Explorer

The Subject and Number cannot be blank.



Find	Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit
	1971	Beans, Ruari Padraig	BA	Mathematics	U	Junior	MA	13-Dec-2013

Format: Student View

☑ Include **in-progress** classes

☑ Include **preregistered** classes

Look Ahead

To see an audit SHOWING COURSES FOR WHICH YOU PLAN TO REGISTER IN FUTURE TERMS, use the form below to enter the Subject code and Number of each anticipated course. Once you have created your list of courses, click on the Process New button.

Enter a course and click Add Course

Subject

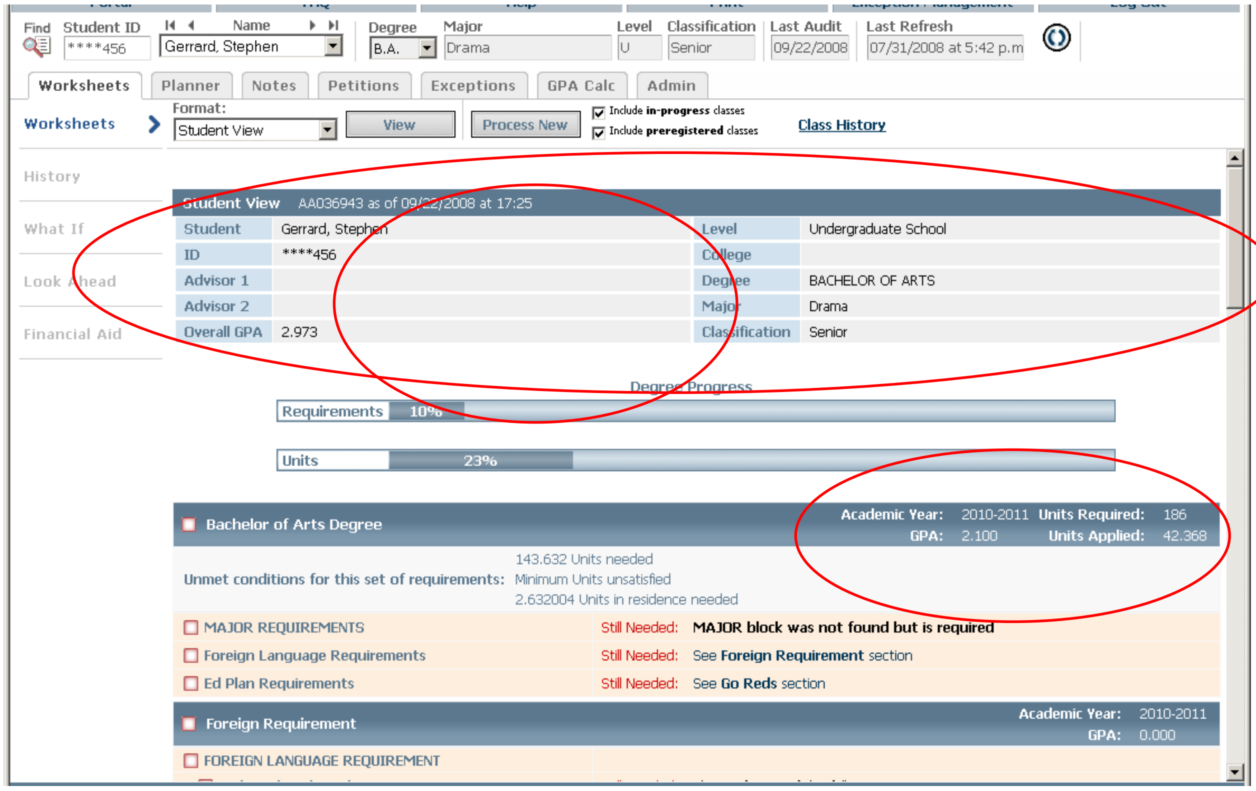
Number

Courses you are considering

Worksheets:

The student header section in the worksheets is not driven from UCX-SCR001 but is driven from new variables at the top of the xsl. This allows for easy localization.

The Catalog Year label does come from UCX-SCR001 however as does the Credits label.



Find	Student ID	Name	Degree	Major	Level	Classification	Last Audit	Last Refresh
	****456	Gerrard, Stephen	B.A.	Drama	U	Senior	09/22/2008	07/31/2008 at 5:42 p.m

Worksheets | Planner | Notes | Petitions | Exceptions | GPA Calc | Admin

Worksheets

History

What If

Look Ahead

Financial Aid

Format: Student View | View | Process New

☑ Include **in-progress** classes
☑ Include **preregistered** classes

Class History

Student View AA036943 as of 09/22/2008 at 17:25

Student	Gerrard, Stephen	Level	Undergraduate School
ID	****456	College	
Advisor 1		Degree	BACHELOR OF ARTS
Advisor 2		Major	Drama
Overall GPA	2.973	Classification	Senior

Degree Progress

Requirements 10%

Units 23%

Bachelor of Arts Degree — **Academic Year:** 2010-2011 **Units Required:** 186 **GPA:** 2.100 **Units Applied:** 42.368

Unmet conditions for this set of requirements: 143.632 Units needed
Minimum Units unsatisfied
2.632004 Units in residence needed

MAJOR REQUIREMENTS	Still Needed:	**MAJOR block was not found but is required**
Foreign Language Requirements	Still Needed:	See **Foreign Requirement** section
Ed Plan Requirements	Still Needed:	See **Go Reds** section

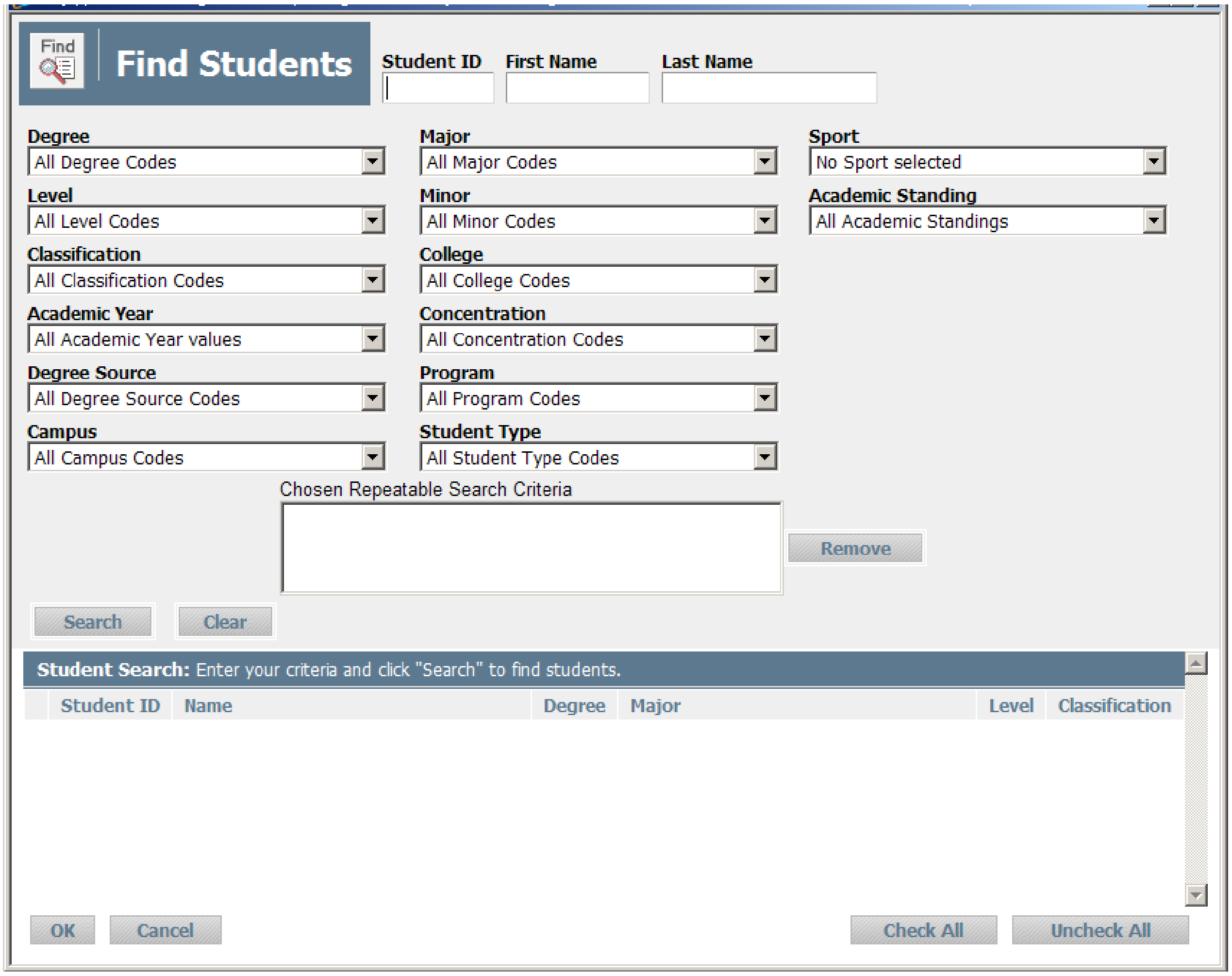
Foreign Requirement — **Academic Year:** 2010-2011 **GPA:** 0.000

FOREIGN LANGUAGE REQUIREMENT

Search Page

All labels here (except for First, Last Name) come from UCX-SCR001

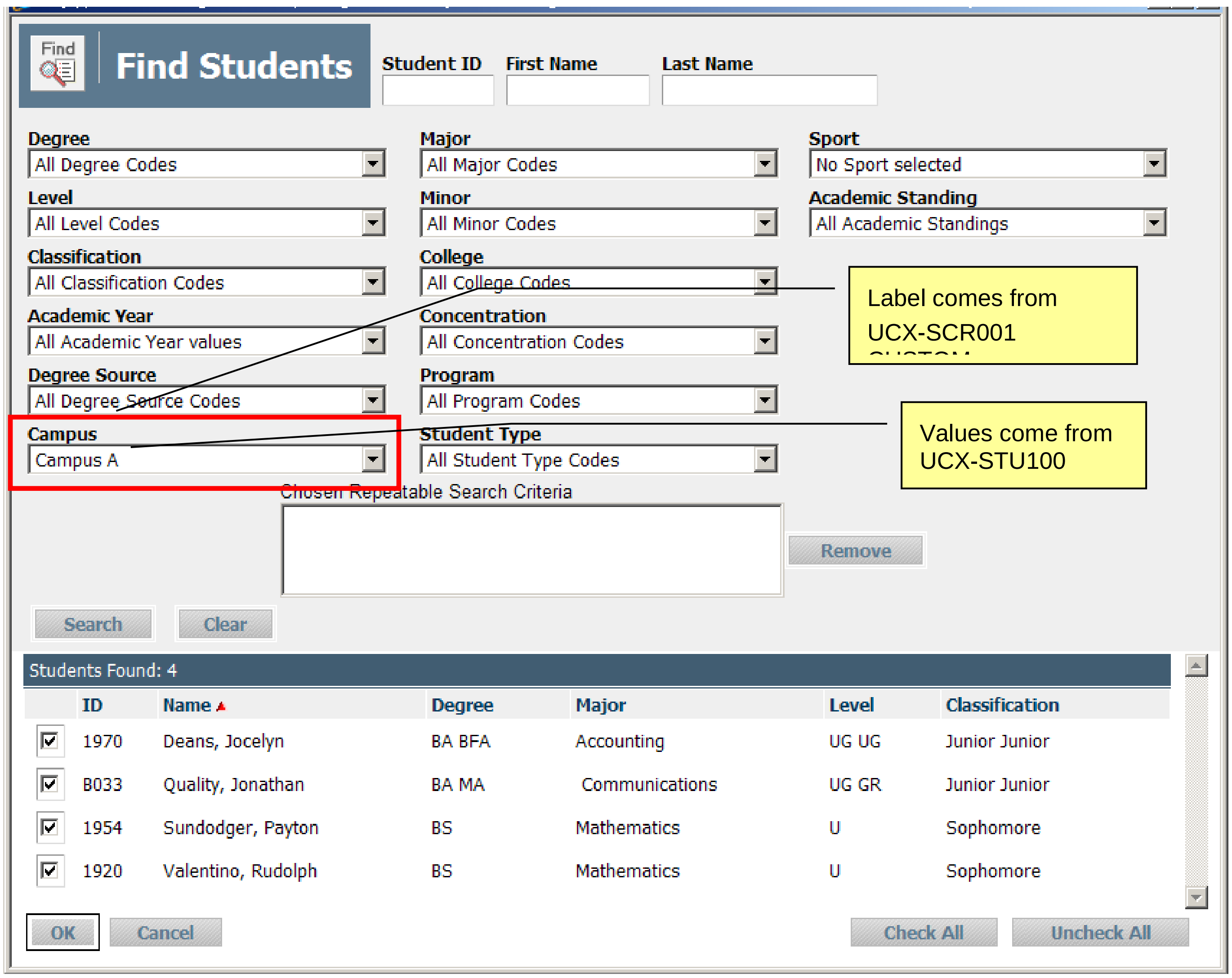
For Banner we are using Level (not School), Student Type (not Status) and Class Standing (not Level)



Single Custom Search Item

You may add an additional search item to the Find Students page to search on data bridged to the rad-custom-dtl. Not only can you search on this item but the data value for each student can also appear in the student context area.

Here we see that a Campus selection item has been added to the Find window. (In this example we have also suppressed the displaying of the Liberal Learning field.) We are using “Campus” in this example but it can be any piece of data associated with the student bridged to the rad-custom-dtl.



Student ID	
First Name	
Last Name	
Degree	All Degree Codes
Major	All Major Codes
Sport	No Sport selected
Level	All Level Codes
Minor	All Minor Codes
Academic Standing	All Academic Standings
Classification	All Classification Codes
College	All College Codes
Academic Year	All Academic Year values
Concentration	All Concentration Codes
Degree Source	All Degree Source Codes
Program	All Program Codes
Campus	Campus A
Student Type	All Student Type Codes

Label comes from UCX-SCR001

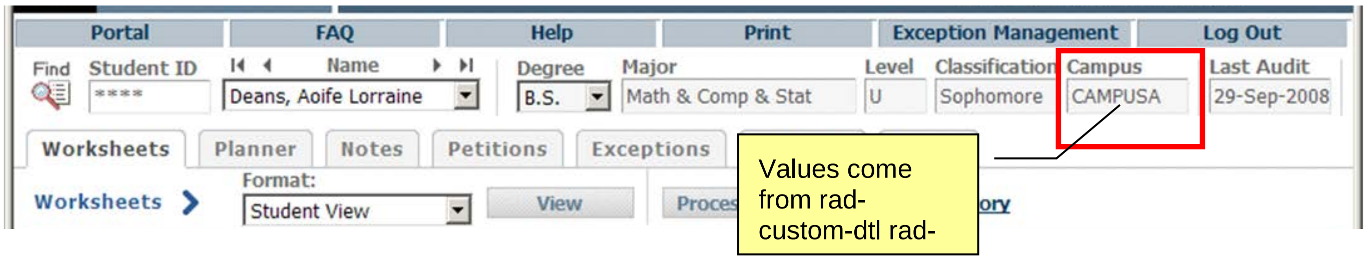
Values come from UCX-STU100

Chosen Repeatable Search Criteria

Students Found: 4

	ID	Name ▲	Degree	Major	Level	Classification
✓	1970	Deans, Jocelyn	BA BFA	Accounting	UG UG	Junior Junior
✓	B033	Quality, Jonathan	BA MA	Communications	UG GR	Junior Junior
✓	1954	Sundodger, Payton	BS	Mathematics	U	Sophomore
✓	1920	Valentino, Rudolph	BS	Mathematics	U	Sophomore

We also see the custom data value of Campus appearing in the student context area.



Student ID	Name	Degree	Major	Level	Classification	Campus	Last Audit
****	Deans, Aoife Lorraine	B.S.	Math & Comp & Stat	U	Sophomore	CAMPUSA	29-Sep-2008

Worksheets

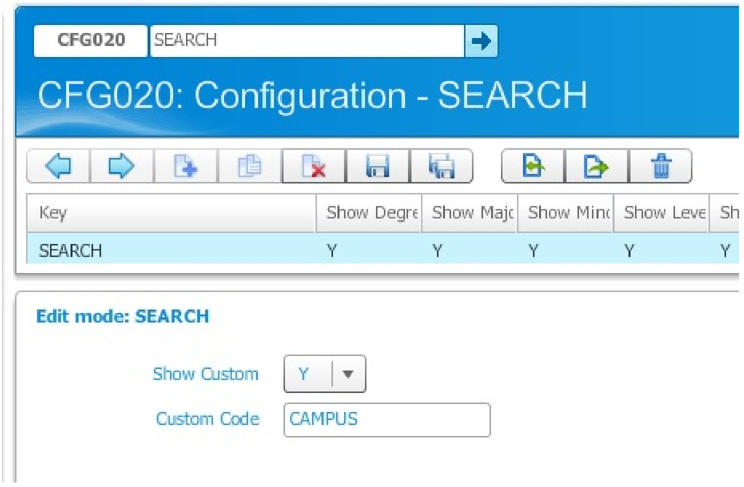
Format: Student View

Values come from rad-custom-dtl rad-

This same campus value can be added to the student header of the worksheets by modifications to the worksheet xsl files.

To have this new search field appear on the Find Students window and to have this field displayed in the student context area of the main window set the UCX-CFG020 SEARCH “Show Custom” field to Y. Set the “Custom Code” field to the value in the rad-custom-dtl rad-custom-code field – this is the record on which the search will be performed and also will be the record that is retrieved when all student information is retrieved regardless of how the search is performed (searching by ‘level’, ‘major’, or similar criteria).

Note that the custom code needs to be an overall value associated with the student and not tied to the student’s degree. For this reason using something like the Degree Status may not be appropriate because the student may have multiple simultaneous degrees. It is assumed that each student only has one of these rad-custom-dtl records – having multiple records with this rad-custom-code will give you interesting results.



CFG020: Configuration - SEARCH

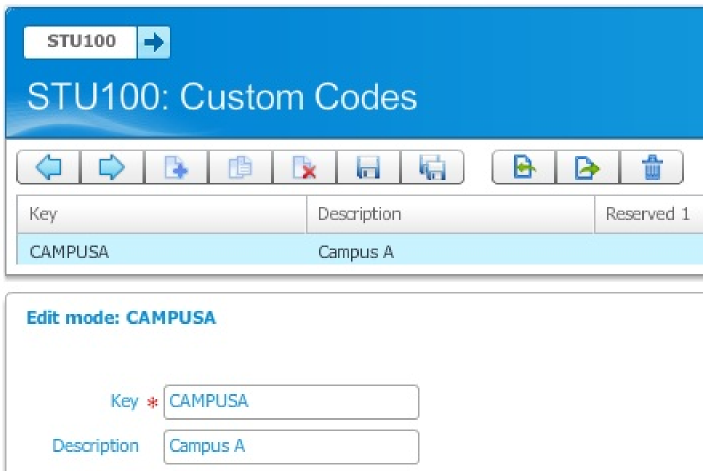
Key	Show Degre	Show Majc	Show Minc	Show Leve	Sh
SEARCH	Y	Y	Y	Y	Y

Edit mode: SEARCH

Show Custom: Y

Custom Code: CAMPUS

The UCX-STU100 table must also be populated to indicate what values will appear in the drop-down list on the Find Students window. The key into each record must be the values found in the rad-custom-dtl on the rad-custom-value field for your students. The Description for each record is the display value that will appear in the drop-down list for each key.

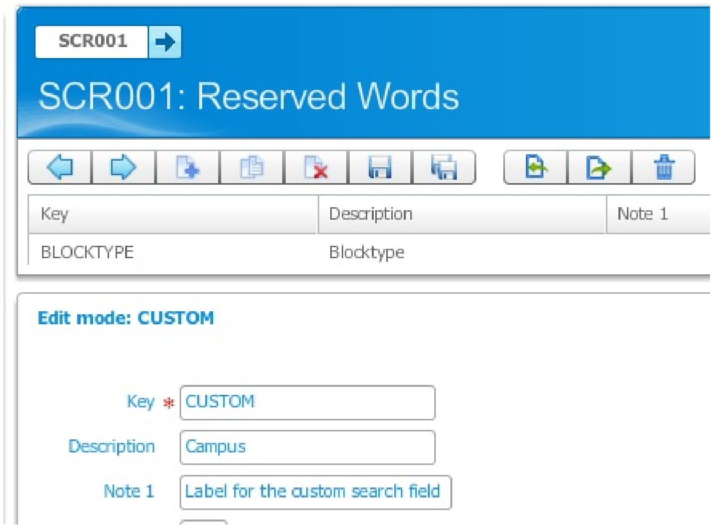


Key	Description	Reserved 1
CAMPUSA	Campus A	

Edit mode: CAMPUSA

Key *	CAMPUSA
Description	Campus A

To get the label on the Find Students and the student context area to describe the type of data you are searching on or displaying you need to modify UCX-SCR001 – change the CUSTOM entry to have whatever label you want. Here we have changed it to “Campus” to get the appropriate label to display.

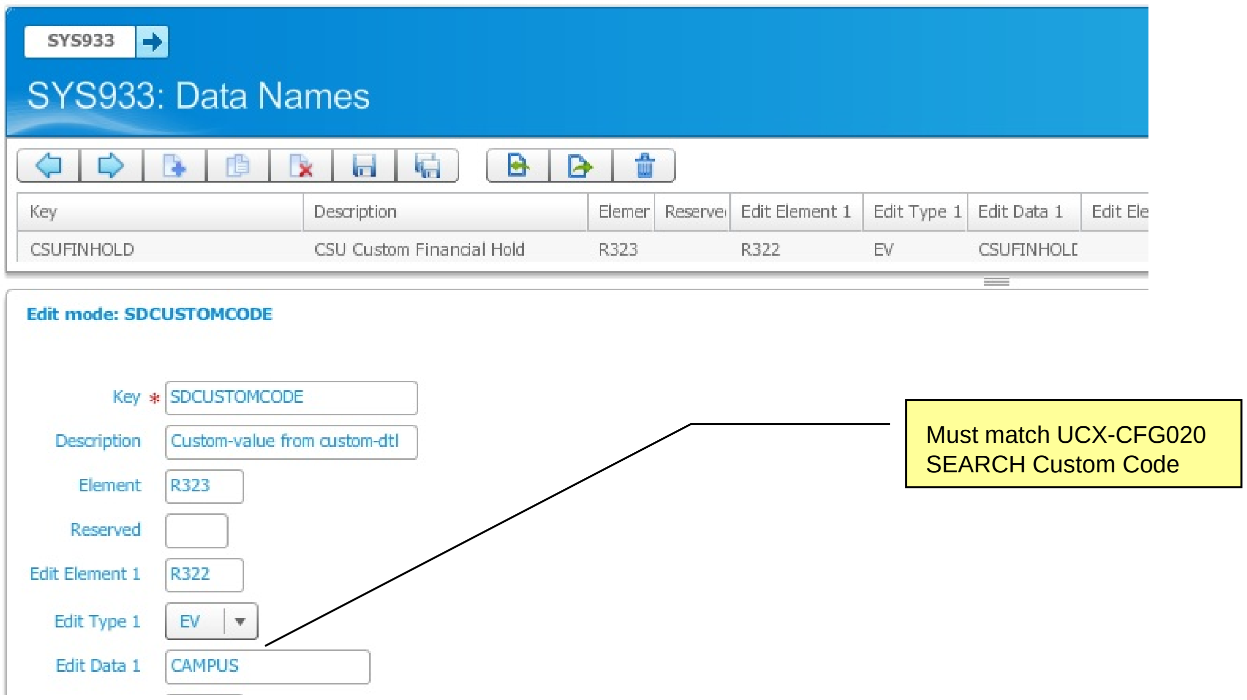


Key	Description	Note 1
BLOCKTYPE	Blocktype	

Edit mode: CUSTOM

Key *	CUSTOM
Description	Campus
Note 1	Label for the custom search field

To get the ID search to work from the main page you must change the UCX-SYS933 SDCUSTOMCODE entry to contain the same value you entered in UCX-CFG020 SEARCH "Custom code".



Key	Description	Elemer	Reservei	Edit Element 1	Edit Type 1	Edit Data 1	Edit Ele
CSUFINHOLD	CSU Custom Financial Hold	R323		R322	EV	CSUFINHOLI	

Edit mode: SDCUSTOMCODE

Key *	SDCUSTOMCODE
Description	Custom-value from custom-dtl
Element	R323
Reserved	
Edit Element 1	R322
Edit Type 1	EV
Edit Data 1	CAMPUS

Using the example of campus following the examples above Banner sites can use entries like this in UCX-BAN080 to get the campus code from Banner pulled into Degree Works. Non-Banner sites can use the normal custom bridge record to push any custom values desired into the rad-custom-dtl.

```
BAN080CAMPUS:COLUMN                  SGBSTDN_CAMP_CODE
BAN080CAMPUS:ORDERBY                 SGBSTDN_TERM_CODE_EFF
BAN080CAMPUS:TABLE                   SGBSTDN a
BAN080CAMPUS:WHERE_1                 a.SGBSTDN_TERM_CODE_EFF =
BAN080CAMPUS:WHERE_2                  (SELECT MAX(b.SGBSTDN_TERM_CODE_EFF)
BAN080CAMPUS:WHERE_3                 FROM SGBSTDN b
BAN080CAMPUS:WHERE_4                  WHERE b.SGBSTDN_PIDM =
a.SGBSTDN_PIDM)
```

As always, remember to issue a webrestart after making any UCX changes.

Additional Custom Search Items

You may add additional search items to the search page as shown below. These search items allow users to search on the other student data stored on the rad_custom_dtl. In the example below, a select box was added to search on SPORT, another to search on ACADSTANDING and another to search on ATTRIBUTE – all searching on the rad_custom_dtl.

Ellucian delivers a SEARCHCUSTOM shepherd script. You should modify it as needed. Make sure each <select> object has an "id" attribute that contains "Custom" in its value – for example:

```
    id="idCustom4"
```

This tells the search page that you are searching on the rad_custom_dtl.

If you want to allow your users to select multiple values from the same select box you should use this onChange event:

```
              onChange="AddSearchItem(this);"
```

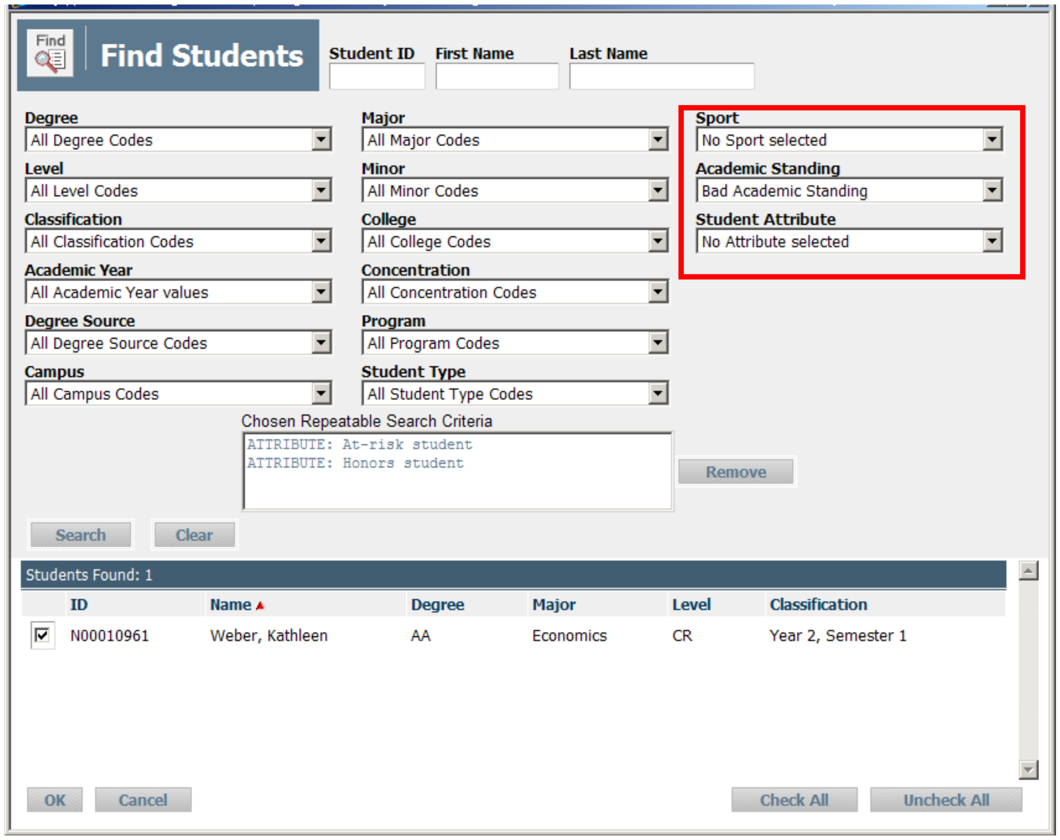
For example, if you setup an Attribute select box you may want to allow users to select on students who have both the HONR attribute as well as the FRGN attribute. Using this onChange event places the values into the collection box that is then processed when the user clicks Search. When this onChange is not used the user is limited to only one value from the select box.

If you need to resize your window because of the number of extra select boxes you added you need to modify the search window height and width values in DGW_Control.js

```
       var iSearchWidth  = 850;
       var iSearchHeight = 690;
```

You may also need to adjust the height of the top section of this window set in SD2SEARCH also.

```
       <frameset rows="440,*,40,0<$ILENV-DWHOLDHEIGHT>"
```



```
ATTRIBUTE: At-risk student
ATTRIBUTE: Honors student
```

Students Found: 1

	ID	Name ▲	Degree	Major	Level	Classification
☑	N00010961	Weber, Kathleen	AA	Economics	CR	Year 2, Semester 1

For details on localizing Transfer Equivalency Self-Service, see the *Transfer Equivalency Self-Service Administration Guide*.

What-if Configuration

The What-if page can operate on one of two modes:

Mode 1) Classic mode. The curriculum rules are not followed. The only filtering is by degree/major based on the UCX-CFG020 WHATIF Degree Drives Major and Major Drives Degree flags. The SD2WIFBODY shpscript is used in this mode.

Mode 2) Curriculum Rules mode. In this mode the rules stored on the rad_CurrRule_dtl are obeyed by the what-if page. The SD2WIFBODYCURR shpscript (in conjuction with SD2WIFBODYCURRMAJORS) is used in this mode with the controlling logic located in the CurrRules.xsl stylesheet. In this mode the two “Drives” flags mentioned above are ignored.
To operate in Curriculum Rules mode set the UCX-CFG020 Obey Curriculum Rules flag to “Y”.

Curriculum Rules mode details

Multiple scenarios are supported under the curriculum rules mode. These scenarios are directly related to how faculty and students understand the rules you have built for your institution. In every scenario the catalog year is used as the first item to select a set of rules. Some curriculum rules may have been discontinued and therefore should not appear on the what-if page. In addition to the catalog year the user must then either choose a program or a combination of school, degree and college. Campus can also be included under either of those scenarios to further filter the list of rules.

Here are the scenarios mode 2 supports:
Scenario A: school-degree-college; degree is chosen before the college

Catalog Year
School
Degree
College

Scenario B: school-college-degree; college is chosen before the degree

Catalog Year
School
College
Degree

Scenario C: school-degree; college is not used for selecting rules

Catalog Year
School
Degree

Scenario D: community college – no school; school on student’s academic record is used

Catalog Year
Degree

Scenario E: multi-campus institution type of school; **campus** used in selection

Catalog Year
Campus
School
Degree
College

Concentrations tied to Majors

Under each scenario concentrations can either be tied to majors or to the overall rule. In the first case the what-if page waits until the major is selected to populate the list of concentrations allowed. The what-if page knows to tie concentrations to majors based on the UCX-CFG020 WHATIF Conc Tied To Major flag.

Majors requiring Concentrations

For majors that require a concentration you can set the UCX-AUD027 Conc Required flag. In the major picklists on the what-if page the user will see an indicator next to the major description for these majors requiring a concentration. In the CurrRules.xsl stylesheet there is a vMajorRequiresConcIndicator setting that allows you to change the asterisk to some other character or text or to nothing at all. When the user attempts to click Add in the additional areas section the page will tell the user that a concentration must be selected for the major chosen. Similarly, when the user attempts to run the what-if audit an alert will tell the user about the missing concentration on the primary major.

One Major per Rule

For curriculum rules that have exactly one major the what-if page pre-selects the major in both the primary and in the additional major picklists. If the user is selecting the program in the primary area the associated major will be chosen automatically with little else for the user to do. If the user is selecting the school, degree and college then the major will be chosen as soon as all three are selected and the user has clicked the “Show related areas of study” button.

Auto-selecting picklists

Much like with the situation of one major per rule the other primary picklists are automatically selected when there is only one option available. For example, if a school is chosen that only offers one degree that degree will be auto-selected.

Disabling empty picklists

Conversely to the auto-selecting feature some picklists are disabled if they contain no entries. For example, when a curriculum rule is chosen and no minors are found the minor picklist will be disabled to communicate this fact to the user.

Program driving curriculum rule

There are two ways to choose a curriculum rule. One option is to select the program. In doing this the what-if page uses the program to determine the school, degree and college tied to it on the curriculum rule and populates those picklists. These picklists are disabled but are visible. The other option is to not use the program but to instead have the user select the school, degree and college to select the curriculum rule.

Program as Degree

When the UCX-CFG020 BANNER Program As Degree flag is set to Y, the Banner extract for the curriculum rules will swap the program and degree values on all records passed to Degree Works. When you have Program-as-Degree enabled, you need to make sure you have UCX-CFG020 WHATIF Program On Curr Rule set to N. This will allow the user to choose the Degree (which is really the program code) to drive the curriculum rule. In addition, the UCX-CFG020 WHATIF Degree College Hierarchy flag must be set to D.

Degree drives College

When school, degree and college are being used to select the rule you have the option of either having the user first select the degree and then the college or the other way around. When the degree is chosen first the college picklist is populated with the valid colleges found on the curriculum rules for the given catalog year, school and degree. The same happens if the college instead drives the degree: the degree picklist is populated with the valid degrees found on the curriculum rules for the given catalog year, school and college.

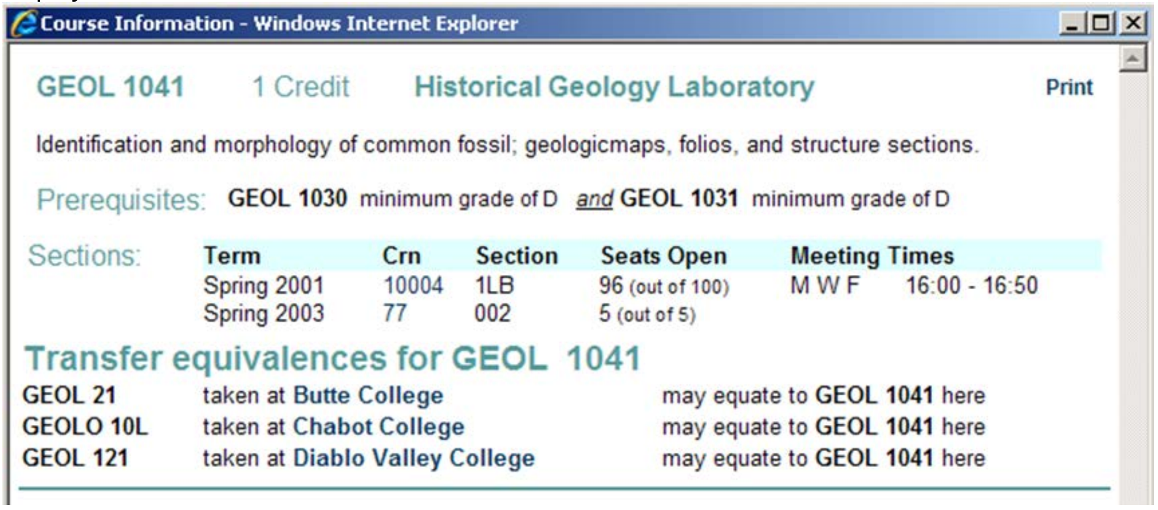
Additional areas of study

You have the option of restricting the user to choose the additional areas of study (second major, minor or concentration for example) from the primary curriculum rule or you can allow the user to choose the additional areas from a different curriculum rule. When the user is restricting to selecting the areas from the primary rule the contents of the major, minor and concentration picklists in the additional areas mirror those shown in the primary area of the page. When the user is free to choose areas from a different curriculum rule the user is shown either a program picklist or a degree and college picklist so that the new curriculum rule can be chosen. Once the rule has been chosen the major, minor and concentration picklists are populated based on what is found for the specified rule.

Course Link

Degree Works has the ability to allow users to click on courses listed in the advice to see a description of the course. The description may contain a configurable listing of course content, pre-requisites, course name changes, etc. The UCX-RPT036 CreateCourse Link flag enables or disables this feature by worksheet, by user role.

When the user places the mouse over a course the course becomes underlined indicating that it may be clicked. For example, “GEOL 1041” becomes “**GEOL 1041**” when the mouse is placed over it. After clicking the hyperlink, a sample (using the Standard configuration settings) of the display returned is as follows:



GEOL 1041   1 Credit   Historical Geology Laboratory   Print

Identification and morphology of common fossil; geologicmaps, folios, and structure sections.

Prerequisites: GEOL 1030 minimum grade of D *and* GEOL 1031 minimum grade of D

Sections:

Term	Crn	Section	Seats Open	Meeting Times	
Spring 2001	10004	1LB	96 (out of 100)	M W F	16:00 - 16:50
Spring 2003	77	002	5 (out of 5)		

Transfer equivalences for GEOL 1041

GEOL 21	taken at Butte College	may equate to GEOL 1041 here
GEOLO 10L	taken at Chabot College	may equate to GEOL 1041 here
GEOL 121	taken at Diablo Valley College	may equate to GEOL 1041 here

One of Course Link's responsibilities is to retrieve information and display it in a customer-defined format, using XML-based technology. To achieve a maintainable format, Course Link will depend upon a combination of "configuration" tables and "customer localization" of the XSL and/or XML sheets.

New settings have been established that allow each audit worksheet to define what a Course Link information window will contain in terms of the pre-defined groupings outlined below. The actual content of each of these groups is fixed from our point of view, but could be localized by the customer to meet their particular needs.

For example, a customer can define via the configuration table that the window should show the TITLE, the ATTRIBUTE, and the TRANSFER groups – in that order. Further, the TRANSFER group should be displaying the "BRIEF" version, not the "STANDARD" or "VERBOSE" version of the text.

It is important to consider the role of the "framework" within which that information is displayed. The window should have the capability of displaying the information defined by the groups for a single course, as well as a collection of courses selected by list, in a logical and consistent manner. Any "print button" relates to the framework, that is, the window, and not to an individual group.

In summary, the customer can use the configuration table to say – for each worksheet -

Which information group
What order
Which version of the group (BRIEF, STANDARD, VERBOSE)

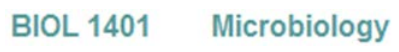
without any localization being required. If the customer does not like the actual arrangement of the information within the group, they take the responsibility for "localizing" the content.
The following text gives a brief overview of each of the information groups.

Configuration Options

TITLE – (UCX-RPT050)

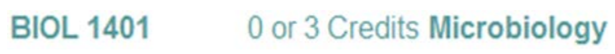
The BRIEF version would include only the course key and title, while the STANDARD/VERBOSE versions would have the course key, title, and credits.

Brief



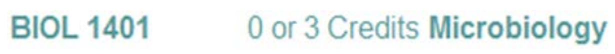
BIOL 1401 Microbiology

Standard



BIOL 1401 0 or 3 Credits Microbiology

Verbose

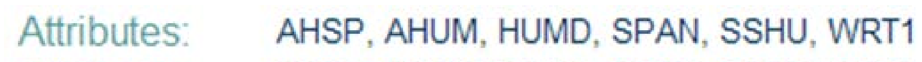


BIOL 1401 0 or 3 Credits Microbiology

ATTRIBUTE – (UCX-RPT052)

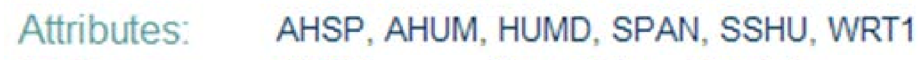
For this group, BRIEF/STANDARD will be just the code, while VERBOSE will be the code and a label.

Brief



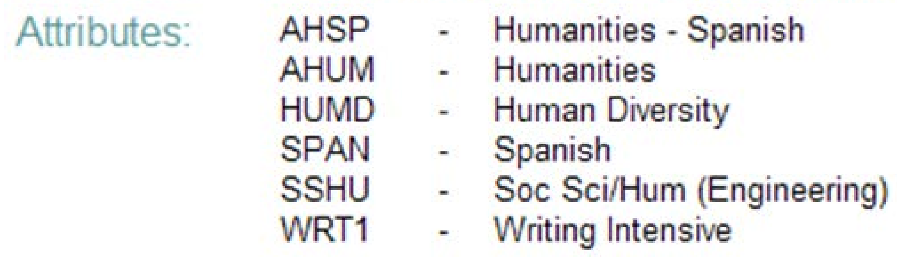
Attributes: AHSP, AHUM, HUMD, SPAN, SSHU, WRT1

Standard



Attributes: AHSP, AHUM, HUMD, SPAN, SSHU, WRT1

Verbose



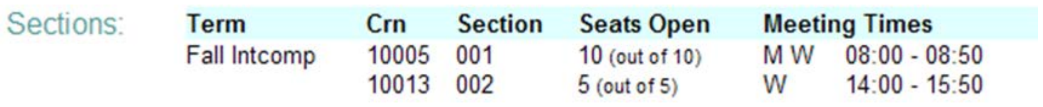
Attributes:

AHSP	-	Humanities - Spanish
AHUM	-	Humanities
HUMD	-	Human Diversity
SPAN	-	Spanish
SSHU	-	Soc Sci/Hum (Engineering)
WRT1	-	Writing Intensive

SECTIONS – (UCX-RPT054)

For the course sections, it checks for a start date newer than two weeks ago. This means the most current term will appear if it started within the last two weeks - and all future terms will appear also. This logic may be altered by doing the following: Change the UCX-CFG020COURSELINK TERM_DAYS_OLD or START_TERM to overwrite "2 week logic". Set the START_TERM to be the first term that should appear; or change the TERM_DAYS_OLD to be more or less than 14 days of the 2-week logic.

The SECTIONS group displays the same format for Brief, Standard and Verbose:



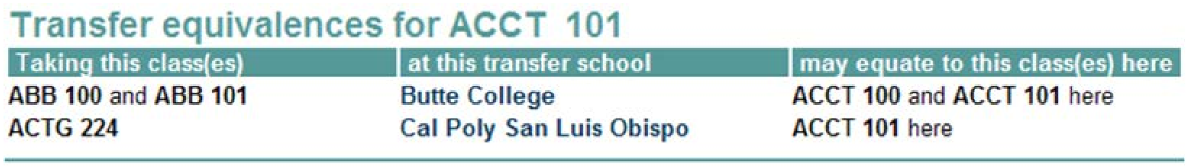
Sections:

Term	Crn	Section	Seats Open	Meeting Times	
Fall Intcomp	10005	001	10 (out of 10)	M W	08:00 - 08:50
	10013	002	5 (out of 5)	W	14:00 - 15:50

TRANSFER – (UCX-RPT056)

The Transfer Equivalency product maintains a library of course mappings for articulation between the institution and feeder schools from which students transfer, or attend classes not offered by the institution. The basic Transfer (often referenced as Course Finder) functionality is a feature of the Transfer Equivalency Self-Service component of Transfer Equivalency, and will be accessible through Course Link as an information group if the customer has a license for Transfer Equivalency.

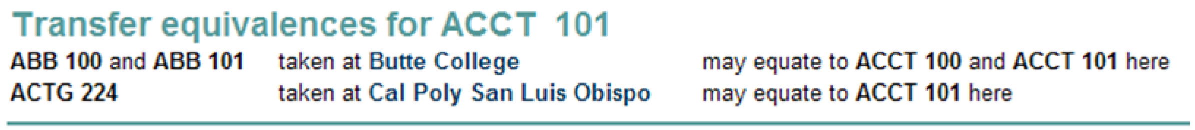
Brief



Transfer equivalences for ACCT 101

Taking this class(es)	at this transfer school	may equate to this class(es) here
ABB 100 and ABB 101	Butte College	ACCT 100 and ACCT 101 here
ACTG 224	Cal Poly San Luis Obispo	ACCT 101 here

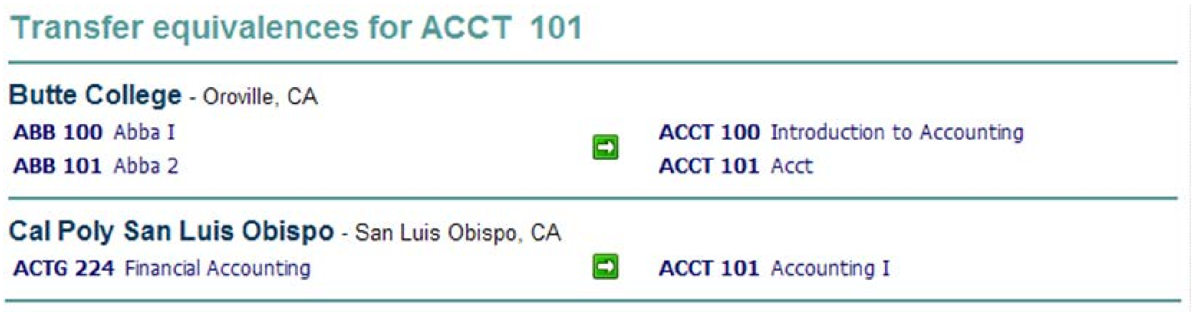
Standard



Transfer equivalences for ACCT 101

ABB 100 and ABB 101	taken at Butte College	may equate to ACCT 100 and ACCT 101 here
ACTG 224	taken at Cal Poly San Luis Obispo	may equate to ACCT 101 here

Verbose



Transfer equivalences for ACCT 101

Butte College - Oroville, CA

ABB 100 Abba I **ABB 101** Abba 2	➡	**ACCT 100** Introduction to Accounting **ACCT 101** Acct

Cal Poly San Luis Obispo - San Luis Obispo, CA

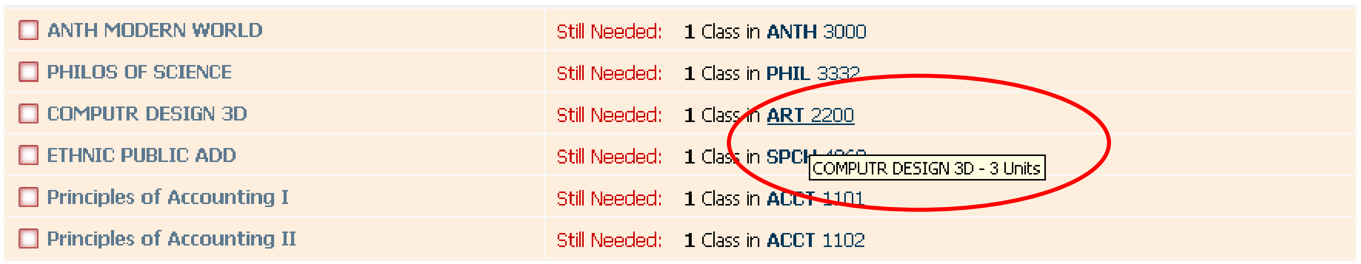
ACTG 224 Financial Accounting	➡	**ACCT 101** Accounting I

Showing Title/Credits as Hint

The standard Degree Works worksheets show the title and credits for each course in the advice as a HTML hint. The hint appears when the user places the mouse over a course in the advice. In addition, in the planner the hint appears on the edit boxes for each course – placing the mouse over an edit box will show the hint containing the course title and credits. When a course is dragged from the worksheet into the planner the title and credits are brought over also. However, if a user types in the course by hand the title and credits will not appear – though saving and reloading the plan will make them appear.

The title and credits to not appear for courses in rules containing a wildcard or a range.

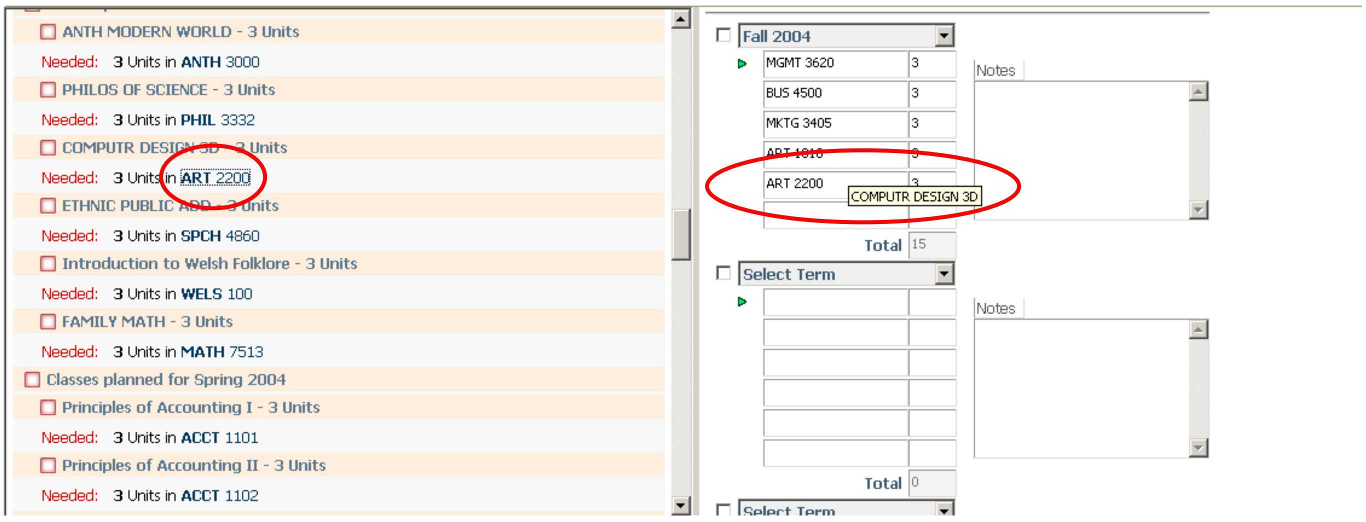
This hint appeared when the mouse was placed over the ART 2200 course:



ANTH MODERN WORLD	Still Needed:	1 Class in ANTH 3000
PHILOS OF SCIENCE	Still Needed:	1 Class in PHIL 3332
COMPUTR DESIGN 3D	Still Needed:	1 Class in ART 2200
ETHNIC PUBLIC ADD	Still Needed:	1 Class in SPCH 4860
Principles of Accounting I	Still Needed:	1 Class in ACCT 1101
Principles of Accounting II	Still Needed:	1 Class in ACCT 1102

COMPUTR DESIGN 3D - 3 Units

Here we clicked on ART 2200 in the worksheet and dragged it into the plan. The title is brought over as a hint and the credits are brought over into the credits field (Internet Explorer only). When a saved plan is redisplayed, the title and credits for all courses appear as hints.



ANTH MODERN WORLD - 3 Units
Needed: 3 Units in ANTH 3000
PHILOS OF SCIENCE - 3 Units
Needed: 3 Units in PHIL 3332
COMPUTR DESIGN 3D - 3 Units
Needed: 3 Units in ART 2200
ETHNIC PUBLIC ADD - 3 Units
Needed: 3 Units in SPCH 4860
Introduction to Welsh Folklore - 3 Units
Needed: 3 Units in WELS 100
FAMILY MATH - 3 Units
Needed: 3 Units in MATH 7513
Classes planned for Spring 2004
Principles of Accounting I - 3 Units
Needed: 3 Units in ACCT 1101
Principles of Accounting II - 3 Units
Needed: 3 Units in ACCT 1102

Fall 2004

MGMT 3620	3
BUS 4500	3
MKTG 3405	3
ART 1010	3
ART 2200	3
Total	15

COMPUTR DESIGN 3D

Select Term

Total 0

Select Term

Install Notes
You do not have to have the UCX-CFG020 DAP13 ValidateCourses=Y – we will still lookup each course on the course-mst even if this flag is turned off. However, when this flag is Y we will give an error if the course is not found. This is to allow sites who do not have all of their courses recorded in the course-mst to still use this feature.

You do need to reparse all of your blocks however – run DAP16.

Once all blocks have been parsed you need to rerun new audits so that he audit tree contains this new title/credits information for each course.

If you wish to turn off this feature you can set the vShowTitleCreditsInHint to N at top of each XSL.

Financial Aid Audits

The Aid tab appears for anyone with the SDAIDAUD key assigned to them. On the Aid tab the user may view the most recent audit, view an historic audit or run a new Financial Aid audit.

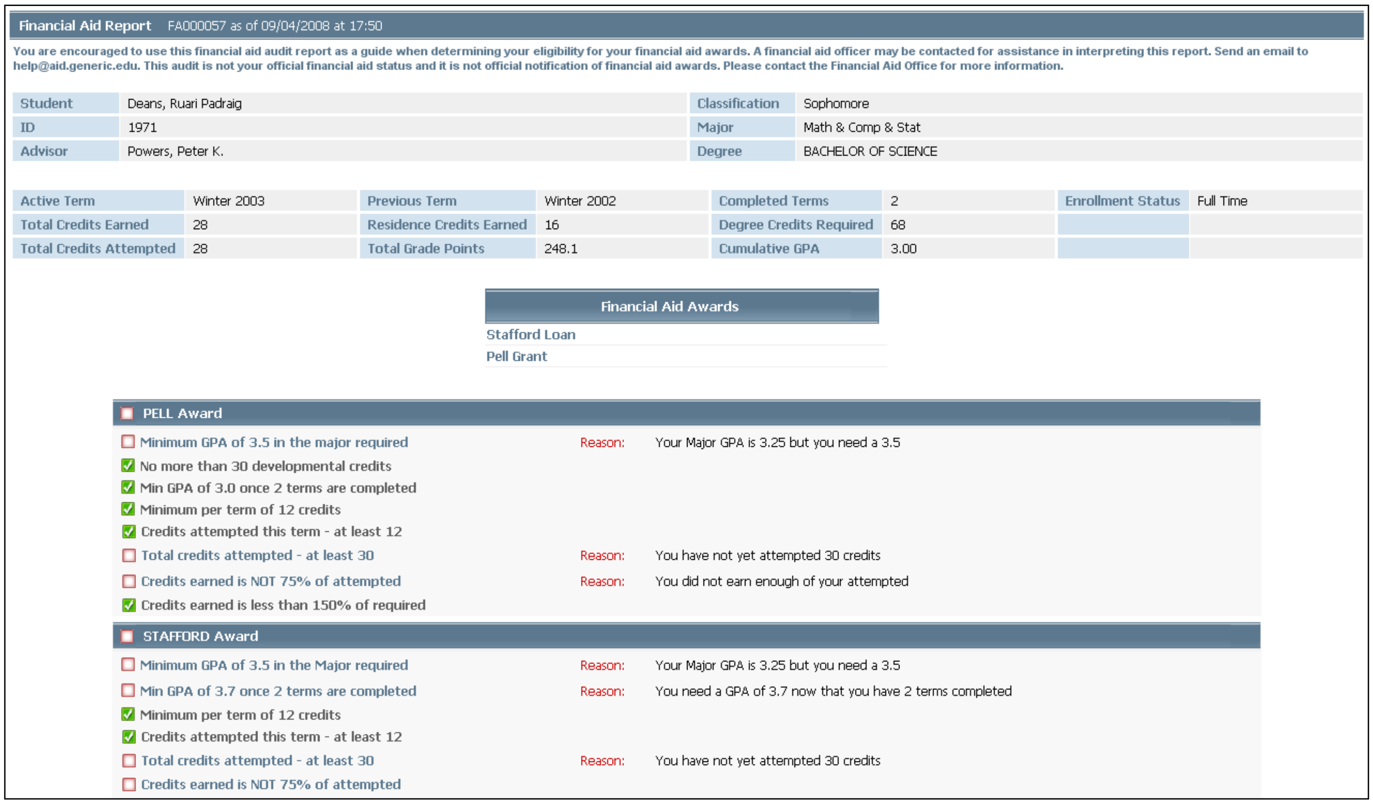
The *Include In-progress classes* and *Include preregistered classes* checkboxes are disabled and only the former is checked. For Financial Aid purposes it is necessary to include the student's current class load while ignoring those classes set in future registration periods. However, you may localize the SD2AIDCON shpscript to alter show this behaves as you wish.

The DGW_Aid_Report.xsl controls the content of the report. A student header with only a select set of data values is shown – but more can be added as needed. In addition, certain term and credit information is also displayed just below the student header – this too can be changed as needed for your audience.

The "Financial Aid Awards" section shows all of the AWARD values found in the rad-aid-dtl table for this student. The XSL allows you to map each of the awards you offer to a nice looking description.

This report shows only the AWARD blocks and suppresses the DEGREE, MAJOR, etc blocks from showing on the report. These other blocks are available for display as all of the normal degree audit information is in the XML being used but by default only the AWARD blocks appear.

This report assumes that each header qualifier in the AWARD block contains a Label and thus is able to show each qualifier as if it were a normal rule.



Financial Aid Report FA000057 as of 09/04/2008 at 17:50

You are encouraged to use this financial aid audit report as a guide when determining your eligibility for your financial aid awards. A financial aid officer may be contacted for assistance in interpreting this report. Send an email to help@aid.generic.edu. This audit is not your official financial aid status and it is not official notification of financial aid awards. Please contact the Financial Aid Office for more information.

Student	Deans, Ruari Padraig	Classification	Sophomore
ID	1971	Major	Math & Comp & Stat
Advisor	Powers, Peter K.	Degree	BACHELOR OF SCIENCE

Active Term	Winter 2003	Previous Term	Winter 2002	Completed Terms	2	Enrollment Status	Full Time
Total Credits Earned	28	Residence Credits Earned	16	Degree Credits Required	68		
Total Credits Attempted	28	Total Grade Points	248.1	Cumulative GPA	3.00		

Financial Aid Awards
Stafford Loan
Pell Grant

PELL Award

☐ Minimum GPA of 3.5 in the major required	Reason:	Your Major GPA is 3.25 but you need a 3.5
✓ No more than 30 developmental credits		
✓ Min GPA of 3.0 once 2 terms are completed		
✓ Minimum per term of 12 credits		
✓ Credits attempted this term - at least 12		
☐ Total credits attempted - at least 30	Reason:	You have not yet attempted 30 credits
☐ Credits earned is NOT 75% of attempted	Reason:	You did not earn enough of your attempted
✓ Credits earned is less than 150% of required		

STAFFORD Award

☐ Minimum GPA of 3.5 in the Major required	Reason:	Your Major GPA is 3.25 but you need a 3.5
☐ Min GPA of 3.7 once 2 terms are completed	Reason:	You need a GPA of 3.7 now that you have 2 terms completed
✓ Minimum per term of 12 credits		
✓ Credits attempted this term - at least 12		
☐ Total credits attempted - at least 30	Reason:	You have not yet attempted 30 credits
☐ Credits earned is NOT 75% of attempted		

Exception Management

The Degree Advisory Processor requires strong exception handling. Although it is possible to create custom requirements for a student through Scribe, doing so for every student that needs an exception to the rules is not tenable. Minor adjustments for a student can be recorded in Degree Works through Degree Works on the Web's Exception Management. These adjustments are called Degree Works exceptions. They are used to alter the results of a student's audit when the student's coursework varies slightly from the normally expected pattern. A Degree Works exception is a way of "bending the rules", not writing new or different rules. If a particular student needs more than a bend in the rules, then use Scribe to write a custom requirement block for that student.

A Degree Works exception can be categorized as one of these types: **force-complete**, **substitute**, **also allow**, **apply here**, **remove course**, and **change-the-limit**. These exception types are designed to handle specific situations that are commonly encountered when doing degree audits. Each exception type has a specific effect on the audit results derived by the Auditor Engine.

In Degree Works Exception Management, an exception is made for a particular student/school/degree combination for a specific requirement. Each exception is assigned one of the exception types defined below. The scope of the exception can be a course, noncourse, rule, rule qualifier, block or block qualifier. Each exception type is limited in scope and can be used only in certain places within requirements.

Use of Labels
The modification of rules within the same block will not unhook the exception from the rule. Modification of the rule on which the exception was placed will, in most cases, also not cause the exception to become unhooked. When exceptions are placed on a rule Degree Works uses the label-tag or label as an identifier. As long as the label-tag or label is not changed Degree Works should always find the rule on which the exception was placed. It is best to have label-tags on all of your rules to ensure exceptions will not become unhooked. If the Allow Duplicate Labels flag in UCX-CFG020, DAP13 is N, Scribe will not allow duplicate labels to be Scribed in the same block. However, if you use label-tags on your labels you can set that UCX-CFG020 DAP13 flag to Y to allow duplicate labels as Degree Works will use the label-tags and not the label text to ensure exceptions are placed on the correct rules. See the *Label Tags* section in the *Scribe User Guide* for more information.

Example:
A Force Complete exception is placed on the Humanities Requirement:

```
9 Credits in SPAN 1@, FREN 1@, ITAL 1@
      Label “Language Requirement”;
6 Credits in ANTH @, HIST @, SOC @
      Label “Humanities Requirement”;
3 Credits in MATH 112, PHIL 118
      Label “Logic Requirement”;
```

The block is changed as follows:

```
9 Credits in SPAN 1@, FREN 1@, ITAL 1@, IRISH @
      Label “Language Requirement”;
3 Credits in MATH 112, PHIL 118
      Label “Logic Requirement”;
8 Credits or 2 Classes in ARTH @, ANTH @, HIST @, SOC @
      Label “Humanities Requirement”;
```

Since Degree Works uses the label to locate the exception location the Force Complete exception will be applied and will not become unhooked even though the rules around the Humanities Requirement have changed and even though the requirement itself has changed.

Substitute and **Remove Course** exceptions may not be applied to a changed rule if the course being substituted or removed cannot be found.

Exception Types

The Exception Type is a code that is assigned to each Degree Works exception. It indicates which kind of exception is being made. The Auditor Engine is programmed to take certain actions based on the Exception Type. UCX-AUD014 is used to validate the Exception Type code.

When deciding what type of exception to make in a given situation, it is important to consider the reason for the exception. If the student has taken a course that is similar to the one required, but does not plan to take the required course, then a Substitution is probably in order. If the student has taken all the coursework, but cannot achieve the required minimum number of courses in residence, then a Force-Complete of the MINRES qualifier may be made. If the Auditor does not apply a course in the "best" place then use a Lock-In to force the Auditor to do what is desired. If the student has job experience in finance, and forcing the student to take the core finance courses is undesirable, then Waive the finance requirement. If the student has taken too many transfer courses, and the MAXTRANSFER limit is exceeded, then use a Change-the-Limit exception to raise the MAXTRANSFER limit.

It is also important to consider the magnitude of the exception. Remember that it is possible to create a custom requirements block in Scribe for a student. If many exceptions are being made for a student, then consider a custom requirements block for that student ID.

ALSO ALLOW

Exception Type = "AA"

Definition: Also Allow is used when a course rule needs to allow more choices. Typically, an advisor decides to let the student use a non-standard course to satisfy a requirement. Therefore, Also Allow exceptions are typically made after the student has already taken the course but this is not a required condition. For example, a student took MATH 210 but a requirement is “5 CREDITS IN MATH 311, 312”. In order to allow the student to graduate on time the advisor wants to allow MATH 210 as a choice on the rule. Using Also Allow the new rule would be “5 CREDITS IN MATH 311, 312, 210”.

Also Allow can only be made on a course list in a rule, not for a course in a list associated with a qualifier. The course rule must be separated by commas (OR); a plus (AND) separated list is not allowed.

Auditor: If the student takes a course that is applied to the specific course specified in the Also Allow exception then the auditor attempts to apply the course to the requirement. If the course fits on another rule or the course causes a maximum to be exceeded then the course may be removed from the rule; the class is not locked in to the rule. If the student does not take a course specified by the exception then the auditor ignores the exception other than showing it in the advice.

APPLY HERE

Exception Type = "AH"

Definition: Apply Here is used when a course should be applied to a specific course rule. Typically, an advisor decides to let the student use a non-standard course to satisfy a requirement. Therefore, Apply Here exceptions are typically made after the student has already taken the course but this is not a required condition. For example, a student took MATH 210 but a requirement is “5 CREDITS IN MATH 311, 312”. In order to allow the student to graduate on time the advisor wants to apply MATH 210 to the rule. Using Apply Here the new rule would be treated as “5 CREDITS IN MATH 311, 312, 210” and MATH 210 would be applied to the rule.

Apply Here can only be made on a course list in a rule, not for a course in a list associated with a qualifier. The course rule must be separated by commas (OR); a plus (AND) separated list is not allowed.

Auditor: If the student takes a course that is applied to the specific course specified in the Apply Here exception then the auditor attempts to apply the course to the requirement. Even if the course fits on another rule or the course causes a maximum to be exceeded the course will *not* be removed from the rule; the class is locked in to the rule. The fact that the course is locked in to the rule is the key distinguishing factor with regard to the Also Allow exception. If the student does not take a course specified by the exception then the auditor ignores the exception other than showing it in the advice.

REPLACE REQUIREMENT (Substitution)

Exception Type = "RR"

Definition: Replace one course with another. Course A is replaced by Course B, where Course A is the requirement and Course B is the replacement requirement. Typically, a student has already taken Course B and the advisor decides to let the student use Course B instead of Course A to satisfy a requirement. Therefore, Replace Requirement exceptions are typically made after the student has already taken Course B. For example, a senior took MATH 310 in his final semester because MATH 305 wasn't offered. In this case, make an exception that replaces MATH 310 with MATH 305.

A substitution can only be made at the rule or block level. The rule may be a subset, group or a simple course rule. A course is replaced anywhere that it is found within the group, subset or course rule. If the exception is on an entire block then the course is replaced anywhere that it is found in the block header or in any of the rules. Wildcards and ranges can be used in a Replace Requirement. This exception, unlike the Substitution exception, replaces courses in the EXCEPT and INCLUDING lists on a course rule.

Auditor: The Auditor replaces Course B with Course A anywhere that it is found in the rule or block. There is no assumption as to whether the student took Course B. An exact match must be made for the replacement to take place: If a rule is specified as “3 credits in ENGL 112:115”, an attempt to replace ENGL 113 with another course on this rule will fail.

NOT NEEDED (Remove course)

Exception Type = "NN"

Definition: Remove a course from a rule’s course list or change the number of required classes or credits on a course rule. Typically, a course is waived and the rule where the course is required needs to be changed to remove the course from the list so that it does not show up in the advice. The number of classes or credits on the rule may also need to be changed to allow the rule to be completed without this course being applied. This exception supports removing a course or changing the classes/credits or both.

This exception can only be made at the rule level. If a course is simply being removed the rule may be a subset, group or a simple course rule. If the number of classes/credits is being changed the rule must be a course rule.

A course is removed anywhere that it is found within the group, subset or course rule. Courses in the EXCEPT and INCLUDING lists on a course rule are also removed.
Auditor: The Auditor replaces Course B with Course A anywhere that it is found in the rule or block. There is no assumption as to whether the student took Course B. An exact match must be made for the replacement to take place: If a rule is specified as “3 credits in ENGL 112:115”, an attempt to replace ENGL 113 with another course on this rule will fail.

This exception is used with Web Degree Works.

FORCE COMPLETION

Exception Type = "FC"

Definition: Force this rule, rule qualifier, or block qualifier to be complete. Force-Complete does not change the requirement -- it completes a requirement that the Auditor Engine would otherwise mark as incomplete. Typically, Force-Complete exceptions are made immediately prior to graduation when a student cannot possibly meet the requirement in time to graduate.

For example, the requirement for a minimum of 36 credits in residence (MINRES 36 CREDITS) cannot be completed because the student has only taken 33 credits in residence and extenuating circumstances prevent the student from taking an additional 3 credits. In cases like this, which usually involve MIN qualifiers, use the Force-Complete exception to indicate that you want to let the student slide.

Force-Complete exceptions are not for a specific course. Most frequently they are used for block or rule qualifiers, but can also be used to complete a course or noncourse rule. This type of exception occurs with: CLASSES, CREDITS, LASTRES, MINCLASS, MINCREDITS, MINGPA, MINPERDISC, MINRES, MINSPREAD, MINTERM, and NONCOURSE.

Note: Force-complete behaves like a de facto change-the-limit exception, where the limit is not specified as part of the exception but is assumed to be whatever limit the student can meet. For example, if the student has a GPA of 1.985 and the requirement is MINGPA 2.0, then a force-complete exception has the same effect as changing MINGPA to 1.985 with a change-the-limit exception.

Auditor: The Auditor will follow its standard algorithm for applying courses to requirements. However, when the Auditor checks if the rule qualifier, block qualifier, or rule is complete, it marks this requirement as complete. Even though the Auditor would normally treat the requirement as unsatisfied, that will not happen if the requirement has a force-complete exception.

The Force-Complete occurs at the node level. If a rule or qualifier is forced complete then the exception is made on a specific node.

REMOVE COURSE & CHANGE THE LIMIT

Exception Type = "NN"

Definition: Replace the number of classes, credits, noncourses, etc. associated with a minimum or maximum in a requirement with a new number. Like Force-Complete, this type of exception typically is made immediately prior to graduation when a student cannot possibly meet the requirement in time to graduate. For example, the requirement for a maximum of 24 transfer credits could be changed to 30 credits for a student. Unlike the other exception types, Change-the-limit changes the limit associated with the requirement so the student is audited with the replacement limit.

Change-the-Limit exceptions are not for a specific course. Most frequently they are used for block or rule qualifiers, but can also be used to change the number of classes, credits, blocktypes, groups, and noncourses in a rule. This type of exception occurs with: CLASSES, CREDITS, LASTRES, MINCLASS, MINCREDITS, MINGPA, MINGRADE, MINPERDISC, MINRES, MINSPREAD, MINTERM, MAXCLASS, MAXCREDITS, MAXPASSFAIL, MAXPERDISC, MAXRES, MAXSPREAD, MAXTERM, MAXTRANSFER, NONEXCLUSIVE, EXCLUSIVE, BLOCKTYPE, GROUP, and NONCOURSE.

The new limit cannot be zero unless zero is used as the lower limit of a range of classes/credits (e.g. 0:2 CLASSES). Disallowing zero avoids unpredictable behavior of the Auditor and keeps the requirements in synch with what the Parser allows. Use Force-Complete instead of Change-the-Limit in situations where zero is the desired limit, or create a custom requirement block for the student that does not include the requirement at all.

Auditor: The Auditor will follow its standard algorithm for applying courses to requirements. However, the limit (number) associated with the keyword in the exception, will be replaced by the new limit.

Change-the-Limit occurs at the node level. The limit associated with a specific node is changed.

Cascading Exceptions

When making exceptions, it is possible for there to be a domino effect of cascading exceptions. This situation arises when one exception is the catalyst for a second exception, which is the catalyst for a third, etc. For example:

```
BEGIN
  10 CLASSES
  MINGPA 2.0
  MINCLASSES 2 IN (SOC);
4 CLASSES IN SOC@, PSY@, POL@;                       #rule 1
3 CLASSES IN MATH115 + 116 + 117;                    #rule 2
3 CLASSES IN BIO 1@, CHE 1@, PHY 1@  MINGPA 2.0;     #rule 3
```

Exceptions made:

```
1. Change the Limit from MINGPA 2.0 to MINGPA 1.5 in rule 3.

   Cascade: The block header MINGPA of 2.0 needs a change the limit
            exception if the block GPA is pulled below 2.0 for this
            student.

2. Substitute SOST@ for SOC@ in rule 1.

   Cascade: MINCLASSES 2 IN (SOC) is impossible to fill so an
            exception that forces it complete is needed
```

Degree Works Accessibility Compliance (Section 508, ADA and WCAG)

Degree Works has two broad types of user interfaces:
One provides functionality which is used only by "back office" staff for building degree requirements, updating validation tables, and launching batch processes. These are referenced as "Administrative applications".
The other provides functionality which is used by the majority of end-users for access to audits, advice, planning, and reporting. These are referenced as "End User applications".

As each user interface is modernized and rewritten, it follows WCAG 2.0 AA guidelines for accessibility. Ellucian has completed a Voluntary Product Accessibility Template (VPAT) for each End User application to better specify the level of compliance in the product.

Web Server Components

For information about the Web Server and its components, refer to the *Degree Works Installation Guide*.

Other Configuration Options

Leepfrog

To use Leepfrog's CourseLeaf tools with Degree Works, you must first license this product from Leepfrog Technologies, Inc.

For information on Leepfrog's product line visit http://courseleaf.com/

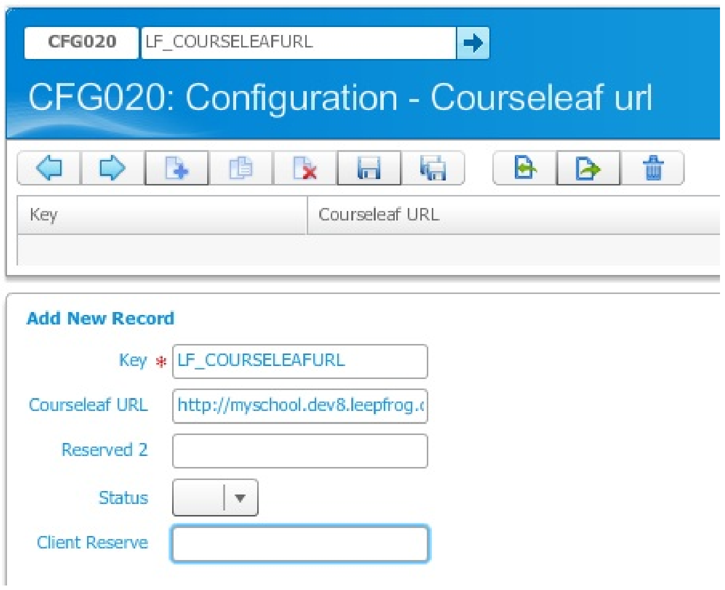
Degree Works integrates with Leepfrog's products in the following ways:

CourseLeaf

Get course information from CourseLeaf.

1. Leepfrog asks Degree Works for a student's class history and planned classes. CourseLeaf creates degree catalog showing what the student has already completed and is planning to take.
2. When a user clicks on a class in the worksheet advice Degree Works sends a request to CourseLeaf to get back a course description.

To enable this link in the Degree Works worksheets create an LF_COURSELEAFURL record in UCX-CFG020 with the URL pointing to the location where your course information resides on the Leepfrog server. The vGetCourseInfoFromServer flag in DGW_Report.xsl also has to be set to "Y".


CFG020 LF_COURSELEAFURL

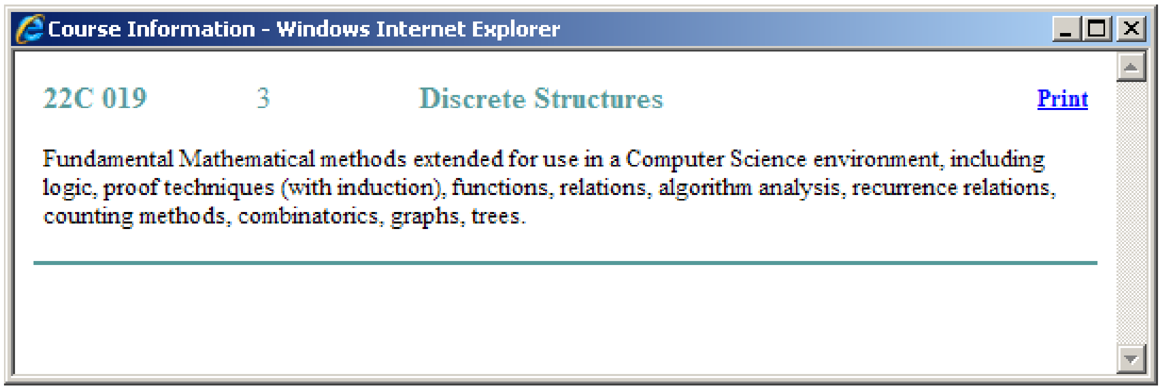
CFG020: Configuration - Courseleaf url

Key	Courseleaf URL

Add New Record

Key *	LF_COURSELEAFURL
Courseleaf URL	http://myschool.dev8.leepfrog.c
Reserved 2	
Status	
Client Reserve	

When the 22C 019 course, for example, is clicked in the worksheet advice a window like this will appear to the user.


Course Information - Windows Internet Explorer

22C 019 3 Discrete Structures Print

Fundamental Mathematical methods extended for use in a Computer Science environment, including logic, proof techniques (with induction), functions, relations, algorithm analysis, recurrence relations, counting methods, combinatorics, graphs, trees.

u.select from redLantern

To use redLantern’s u.select tool with Degree Works you must first license this product from redLantern. For information on redLantern’s product line visit http://redLanternu.com

Degree Works support of u.select Articulation/Audit request

The redLantern u.select product sends to Degree Works a request to perform an articulation and degree audit for the given transfer information. Degree Works first performs an articulation against the mappings that exist in the Degree Works database and then uses the intended degree/major information passed in along with the articulation results to perform a degree audit. The results of the articulation and the audit are passed back to u.select as an XML tree for processing.

You must have your mappings stored in Degree Works in the dap-mapping-dtl. You can create them using Transfer Equivalency, bridge them from Banner or import them from another source. However, you do not need to license Transfer Equivalency from Ellucian – you must ensure you have your mappings built and your transfer schools listed in the rad-ets-mst.

Database Tables

Introduction

The next section gives a list of all database tables used within Degree Works.

dap Tables

Table	Description
dap_appdata_dtl	This table stores additional goal data for applicants processed in Transfer Equivalency.
dap_applicnt_mst	This table stores information on applicants processed in Transfer Equivalency.
dap_audit_dtl	This table contains the description of an audit. Each degree audit that is performed, other than What-If, has an entry. There is one dap_audit_dtl for each dap_audit_id. Audit results that may need to be used as selection criteria are stored here.
dap_audtree_dtl	This table stores the steno audit trees created by the Auditor Engine. Each entry is one line of an audit tree. The audit tree is binary data.
dap_college_dtl	This table stores information about transfer schools for applicants processed in Transfer Equivalency.
dap_eqv_crs_mst	This table tracks course equivalence through history across catalog years. As course numbers change or are reused, entries in this table map from the course the student took to the course for a specific catalog year. The key is a concatenated list of catalog year of student course plus discipline and number of student course plus catalog year being evaluated. For example, "1990 MATH 301 1995 " maps to "MATH 310 ", i.e. MATH 301 taken in the 1990 catalog year is equivalent to MATH 310 in the 1995 catalog year.
dap_except_dtl	This table records manual overrides to the Auditor engine. Sample exceptions are waivers, substitutions, or lock-ins. There is one dap_except_dtl per exception, with a maximum of 9,999 exceptions per student/school/degree combination. Each exception is tied to a particular requirement in a requirements block. This table links the requirement to the exception and the exception to the student.
dap_gpa_history	This table stores overall and major GPA information by term from the student's audit. Used in the Tracking component of the new generation SEP.
dap_map_attr_dtl	This table stores the attributes related to the mappings in the dap_mapping_dtl.
dap_map_cond_dtl	This table stores the conditions related to the mappings in the dap_mapping_dtl.
dap_mapping_dtl	This table stores the mappings created in Transfer Equivalency.
dap_next_id_mst	This table contains the next available ID for Mappings, Requirement blocks and Audits. dap_next_key is "M" for mappings, "R" for requirements and "A" for audits. dap_next_id is of the form "Mxnnnnnn" for mappings, "Rxnnnnnn" for requirements and "Axnnnnnn" for audits. "x" is a letter, A-Z. "nnnnnn" is a number, sequentially assigned from 000001 to 999999. When 999999 is reached the "x" value is changed to the next letter. There is a maximum of 26 million requirement blocks or degree audits.
dap_note_dtl	This table contains the header information about a Degree Works note. It contains global information about a note for a student, such as note-status and note-type. There is one dap_note_dtl per student per note.

Table	Description
dap_note_txt_dtl	This table contains the text of a Degree Works note. Each entry contains one line of text per note. Sorting on note_num and note_seq displays the text of the note in order.
dap_plancrs_dtl	This table stores term and class information for student plans in classic SEP.
dap_planner_dtl	This table stores student planner information: degree/school, plan-id, etc. in classic SEP.
dap_plannote_dtl	This table stores notes about a student's plan in classic SEP.
dap_pt_crs_dtl	This table stores class information for the plan template in classic SEP.
dap_pt_note_dtl	This table stores notes for the plan template in classic SEP.
dap_req_block	This table describes a block of requirements. It associates a requirements ID with the database tags (Catalog Year, Degree, etc.) for the block.
dap_req_crs_dtl	This tables houses the courses referenced in each of the requirement blocks. This table allows you to easily find out the blocks in which a course is referenced – for maintenance purposes.
dap_req_link_dtl	This table points to other blocks that were referenced in the requirements text using the BLOCK keyword. Degree Works uses this to know when a change to one block may affect another block and to ensure that a block is not circular (block A links to block B which links back to block A).
dap_resclass_dtl	This table stores class information used in an audit to be used by CPA.
dap_resnoncr_dtl	This table stores noncourse information used in an audit to be used by CPA.
dap_result_dtl	This table stores audit information to be used by CPA.
dap_student_mst	This table tracks the most recent activity for a student. An entry is created for each student who is audited or has exceptions or notes. There is one entry per student. This table also tracks whether or not the student is locked for processing.
dap_template_mst	This table stores information about a plan to be used by a set of students – a plan template in classic SEP.
dap_title_dtl	This table stores the titles of transfer courses entered in Transfer Equivalency.
dap_transfer_dtl	This table stores transfer and advanced placement exam information for Transfer Equivalency.
dap_undecide_dtl	This table stores information about unresolved articulation results.

rad Tables

Table	Description
rad_aid_dtl	Stores financial aid data to be used in the Financial Aid audit worksheet.
rad_aid_hsh	Stores hash value for the aid_dtl data
rad_applicnt_dtl	Transfer Equivalency application data
rad_applicnt_hsh	Stores hash value for the applicnt_dtl data
rad_attr_dtl	Stores attributes about each class the student has taken – transfer_dtl and class_dtl
rad_attr_hsh	Stores hash value for the attr_dtl data
rad_class_dtl	Stores in-residence classes taken – historic and in-progress
rad_class_hsh	Stores hash for the class_dtl data
rad_course_mst	Stores courses offered by the institution: title, credits, etc
rad_crs_attr_dtl	Stores attributes about each class offered by the institution – associated with the course-mst
rad_currrule_dtl	Stores curriculum rule data bridged from the SIS and used for What-If and Transfer Equivalency goal data filtering
rad_custom_dtl	Stores other information about the student need by scribe requirements
rad_custom_hsh	Stores hash for the custom_dtl data

Table	Description
rad_ets_mst	Transfer Equivalency list of transfer schools
rad_goal_dtl	Stores school, degree, student level, catalog year information
rad_goal_hsh	Stores hash for the rad_goal_dtl
rad_goaldata_dtl	Stores fields of study, such as major, minor, concentration, as well as advisor information
rad_goaldata_hsh	Stores hash for the rad_goaldata_dtl
rad_hash_mst	Stores a record of what was loaded for this student; works with all rad_*_hsh tables
rad_log_dtl	Log of bridge activity
rad_next_id_mst	Next-id information for courses, students and ETS
rad_noncrse_dtl	Stores student non-course data
rad_noncrse_hsh	Stores hash for the noncrse_dtl data
rad_previnst_dtl	Stores student’s previous degree information
rad_previnst_hsh	Stores hash for the previnst_dtl data
rad_primary_mst	Stores student name
rad_report_dtl	Stores other student data that needs to appear on the worksheet
rad_report_hsh	Stores hash for the report_dtl data
rad_student_hsh	Stores hash value for the primary (name), biog (birthdate and SSN) and student (active-term) data
rad_student_mst	Stores the student’s active term
rad_swap_id_dtl	Stores a record of when a student ID was changed from one value to another via the bridge
rad_term_dtl	Stores student cum GPA/credits
rad_term_hsh	Stores hash for the term_dtl data
rad_test_dtl	Stores student test score information
rad_test_hsh	Stores hash for the test_dtl data
rad_transfer_dtl	Stores transfer class information
rad_transfer_hsh	Stores hash for the transfer_dtl data

shp Tables

Table	Description
shp_composer_mst	Stores baseline and localized Shepherd Script records
shp_group_mst	Loaded from UCX_SHP077; specifies default keys/access for each user-class
shp_log_dtl	Stores web activity information
shp_resource_mst	Stores baseline and localized CSS, XSL, image and application properties records
shp_service_mst	Stores key-ring for tabs, functions, etc – though normally key and service names match
shp_user_mst	Stores user’s ID and password and primary user class

sep Tables

Student Educational Planner

Table	Description
sep_plan_class	Plan class information.
sep_plan_class_note	Plan class note.
sep_plan_gpa	Plan gpa information
sep_plan_gpa_note	Plan gpa notes.
sep_plan_group	Plan group information.
sep_plan_noncourse	Plan non-course information.
sep_plan_noncourse_note	Plan non-course note.
sep_plan_note	Plan notes.
sep_plan_placeholder	Plan placeholder information.
sep_plan_placeholder_note	Plan placeholder note.
sep_plan_term	Plan term information.
sep_plan_term_note	Plan term note.
sep_plan_test	Plan test information.
sep_plan_test_note	Plan test notes.
sep_tmpl_class	Template class information.
sep_tmpl_class_note	Template class note.
sep_ tmpl _gpa	Template gpa information
sep_ tmpl _gpa_note	Template gpa notes.
sep_ tmpl _group	Template group information.
sep_tmpl_noncourse	Template non-course information.
sep_tmpl_noncourse_note	Template non-course note.
sep_ tmpl _note	Template notes.
sep_ tmpl _placeholder	Template placeholder information.
sep_ tmpl _placeholder_note	Template placeholder note.
sep_ tmpl _term	Template term information.
sep_ tmpl _term_note	Template term note.
sep_ tmpl _test	Template test information.
sep_ tmpl _test_note	Template test notes.

Transit Tables

The Transit tables are in a different schema than the rest of the Degree Works tables. On the classic server see the $DB_LOGIN_TRANSIT variable to determine where the Transit tables reside. On the classic server you can run *dbt* to run sqlplus to connect to the Transit schema.

Table	Description
TRANSIT_JOB_INSTANCE	This table stores information on requested, running, and completed jobs.
TRANSIT_ARTIFACT	This table hold one artifact of a job. An artifact is some kind of output from the job such as a report, stdout, or action report.
TRANSIT_SEED	Holds the next available job number.
DATABASECHANGELOG	Used by the database maintenance utility to track database changes. Not used by Degree Works applications directly.
DATABASECHANGELOGLOCK	Used by the database maintenance utility to track database changes. Not used by Degree Works applications directly.

Special Scripts

The scripts identified below will not always be accessed through a user interface. Most of these scripts are used within other scripts and processes, and are listed here for informational purposes.

Scripts that have additional documentation needs are listed after the table and have an asterisk (*) next to their names in the first column.

Warning: Do not place scripts into the local/scripts directory as they are not used; only those used in app/scripts are used.

List of Scripts used by Degree Works

Script name	Description
`bannerextract`	Batch extract Banner data – see the Banner Considerations documentation for more information
`colleagueextract`	Batch extract Colleague data – see the Colleague Considerations documentation for more information.
`Changepassword*`	Finds the shp_user_mst using the input Access ID (shp_access_id) and replaces the shp_access_code with the input Password.
`convertplans*`	Converts classic Student Educational Planner (SEP) plan tables into the new generation SEP plan table structure.
`converttemplates*`	Converts classic Student Educational Planner (SEP) template tables into the new generation SEP template table structure.
`dap16all`	Reparses all the blocks in the database; it simply calls DAP16JOB.
`dap22dbg`	Runs and tar up lots of good information on the audit for this student. $ dap22dbg 123456 mytarfile
`dap22ids`	Runs audits based on the student IDs in specified file – must be in *data* directory; use parameter file common/DAP22IDS: example: `dap22ids MYSTUIDS`
`dapauditstopdffiles*`	Takes an input file of audit IDs and the name of a FOP stylesheet and creates a separate PDF file for each audit. See below for more details.
`dapauditstoxmlfiles*`	Takes an input file of audit IDs and creates a separate XML file for each audit. See below for more details.
`dapauditstoxml*`	Runs the getxmlaudit against the audits in the db and sends results to a single file; you can modify the script to select a subset of audits/students
`dapblockinsert*`	Inserts each of the blocks found in the admin/*blocks* directory into the db; see the script for the format of the header each block must have for it to find the block type and title; the script prompts the user for the start and stop catalog year. See below for more details.
`dapblockload`	Loads the dap-req-block and dap-next-id-mst (domain of RA or RB) from files. You must specify a parameter of R to do all requirements or RB to only load in the blocks starting with RB or RA for the non-planner blocks. The RB blocks are those generated from plans for students. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works. Please see the *As Neeeded Tasks* section for more information.

Script name	Description
`dapblocksget`	Use this to back-up your blocks to a text file named **blocks.out.** The output file will contain all primary and secondary tags and the block text. $ **dapblocksget** MAJOR 2015 # get the major blocks for this catalog year $ **dapblocksget** MAJOR # get the major blocks for all catalog years $ **dapblocksget** # get all blocks
`dapblockunload`	Unload the dap-req-block and dap-next-id-mst (domain of RA or RB) to files. The default is to unload only the RA blocks to files. You can specify a parameter of R to do all requirements or RB to only unload the blocks starting with RB. The RB blocks are those generated from plans for students. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works. Please see the *As Neeeded Tasks* section for more information.
`dapdelaudits`	Deletes audits older than the specified date. See the Freezing Audits section for more information about deleting frozen audits.
`dapfindbadaudits*`	Displays a list of audits by student ID, audit ID and audit date that are believed to be corrupt. See the section below for more information.
`dapfindorphanedaudits*`	Displays a list of audits by student ID, audit ID, old degree and new degree that are associated with a student in the rad_goal_dtl that now has a different school/degree. See the section below for more information.
`daphits`	Shows how many times the Degree Works web services have been hit; see *dapreset*
`dapmapcopy*`	Copies all of the mappings from one school to another school (See Special Topic)
`dapmapload`	Loads the dap-mapping-dtl, dap-map-cond-dtl, dap-title-dtl and dap-next-id-mst (domain of M) from files. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.
`dapmapunload`	Unloads the dap-mapping-dtl, dap-map-cond-dtl, dap-title-dtl and dap-next-id-mst (domain of M) to files. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.
`dapreqcrs`	Finds the blocks the reference a particular course (used by SCR02JOB)
`dapreqgrep`	Searches through the text on the dap-req-text-dtl for the string specified (used by SCR10JOB)
`dapreqlist`	Gets a listing of block type, block value, title, catalog years and parse status for all blocks. (used by SCR05JOB)
`dapreq2ndlist`	Gets a listing of block type, block value, title, catalog years and parse status and all of the secondary tags for all blocks. Used by SCR06JOB.
`dapreset`	Resets the hit counter on the Degree Works web services to zero; see *daphits*
`daprestart`	Runs dapstop and then dapstart – and optionally with debugging on
`dapsendemail`	Send an email to this address with the contents of this file as the body with this subject. Example: $ dapsendemail myfriend@myschool.edu thisfile.log “Results of query”
`dapshow`	Shows the dap10 and dap08 processes
`dapstart`	Starts dap10 and dap08 for use with the Scribe. See the System Admin section for more information.
`dapstop`	Stops the dap10 and dap08 processes
`dapucx2eqv`	Loads the equivalences from UCX-CFG070 into the dap-eqv-crs-mst
`dapucxload`	Loads the UCX from a file. See the *System Administration* section for more details.
`dapucxunload`	Unloads all of the UCX to a file. See the *System Administration* section for more details.
`dapxpt`	Shows the list of exceptions saved to the database

Script name	Description
`dbbuild*`	Modify or create Degree Works database tables and objects.
`debugoff`	Exports DWDEBUG=0; unsets DW_LOGDEBUG_PID
`debugon`	Exports DWDEBUG=1; export DW_LOGDEBUG_PID to the 1st param specified
`deleteall`	Used to delete ALL student data from a Degree Works database. This script calls deleteall.sql which truncates the Degree Works database tables which store student data. Only students are affected, other users are not deleted.
`deletestu`	Used to delete student data from Degree Works database based on bridge date. Prompts user for date in YYYYMMDD format; all student data bridged <= that date will be deleted. See Systems Administration section for more details. Only Banner or Colleague schools can use this script.
`dgwversion`	Shows version information for Degree Works source code.
`dwsettings*`	Import, Export, Delete, or Overwrite on the shp_settings_mst table – see section below
`etssync`	Synchronize the rad_ets_mst records from the global directory. For more information run “etssync --help”.
`exptable`	Exports the data for a given table to a file – to be later imported using *importall;* example*:* `exptable dap_next_id_mst myfile "dap_next_id like 'P%'"` **Note:** only tables listed in dapdb, ucxdb, shpdb or raddb in the schema directory can be used with this script.
`getxmlaudit*`	Unloads and audit from the db and runs dapext to create an xml <Audit> tree; example: `getxmlaudit AA000123`
`importall`	Imports the data in the given file (created by *exptable*); *example*: `importall SHP myfile`
`launchjob`	A script to launch Transit jobs via cron. See the section *Cron setup for Degree Works* in this document for more information.
`petsend`	Sends an email to the given address notifying of WAITING or APPROVED petitions.
`profiledbg*`	Analyzes a debug file that contains timing entries and creates a data file that can be imported into a spreadsheet for analysis. Requires knowledge of the internals of the Degree Works programs, and so is to be used under direction from the Degree Works Action Line.

`rad30dbg`	Runs and tar up lots of good information on the Banner extract. The tar file created will always be “rad30dbg.tar”. It will overwrite a previously existing file. Examples:
	
	`$ rad30dbg student [student ID]`
	`$rad30dbg student 123456`
	
	`$ rad30dbg student [student SQL file.sql]`
	`$rad30dbg student stuselect.sql`
	
	`$ rad30dbg [any bannerextract mode]`
	`$ rad30dbg advisor`

`radrestart`	Runs radstop and then radstart
`radshow`	Shows the rad08 processes and its children
`radstart`	Starts rad08 – dynamic bridge daemon
`radstop`	Stops the rad08 processes
`resrestart`	Runs resstop and then resstart
`resshow`	Shows the dap25 parent and any running child processes
`resstart`	Starts dap25 – dynamic CPA daemon

Script name	Description
`resstop`	Stops the dap25 process

`rmoldfiles`	A new script created to help you keep the logdebug, dgwspool and data directories clean of files building up. You can add rmoldfiles to your chron job running weekly or nightly. The 1st parameter is the directory name and the second parameter is the age of the file in days; if not supplied the default is 7 days:
	Examples:
	`  $ rmoldfiles logdebug`
	`  $ rmoldfiles /dw/admin/data`
	`  $ rmoldfiles dgwspool 7`
	`  $ rmoldfiles /dw/admin/jobdata`
	
	To add the rmoldfiles script to your cron job you can run crontab -e and edit the file to run the rmoldfiles script. The user that owns the crontab file must have proper permissions to delete the logdebug files.
	
	`$ crontab –e`
	`# min hr dm mo dw script`
	`# Run rmoldfiles every Sunday morning to delete`
	`# logdebug files older than 7 days`
	`    0  5  *  *  0 /dw/app/scripts/rmoldfiles /dw/admin/logdebug 7 >/tmp/rmlogdebug.log 2>&1`
	`    0  5  *  *  0 /dw/app/scripts/rmoldfiles /dw/admin/dgwspool 15 >/tmp/rmdgwspool.log 2>&1`
	`    0  5  *  *  0 /dw/app/scripts/rmoldfiles /dw/admin/data 20 >/tmp/rmdata.log 2>&1\`

`sepdeleteplan`	Delete a plan from the sep_plan table and all of its related tables by the student ID or the plan ID. Use “sepdeleteplan -h” for more information.
`sepdeletetemplate`	Delete a template from the sep_tmpl_mst table and all of its related tables by the template ID. Use “sepdeletetemplate -h” for more information.
`setdbpasswords`	Configures the values for the Degree Works database password and the Student Information System database password. You may give the passwords either as option flags to the command or as positional parameters. If given as positional parameters, the Degree Works password is first. If neither of those is provided, the script will prompt for the values. The SIS password is optional. For more information, issue the `setdbpasswords --help` command.
`showdbpasswords`	Displays the encrypted string that represents the database passwords. This is typically used in the configuration of the data source for the java applications. For more information, issue the `showdbpasswords --help` command.
`packdebug`	Reads debugging output from specified log files by sessionId and creates a zip file with an encrypted key. The script should be run in the target directory where log files exist, or should be provided complete log file path for the parameters. For more information, issue the packdebug --help command.
`sharegen*`	Produces schema file(s) for the purpose of separating shared and unique tables for a Degree Works database that supports multiple institutions. Called by the dbbuild script.
`shareinfo*`	Generates a list of database owners and associated table_names with table_names like ‘DGW%’, ‘RAD%’, ‘SHP%’ or ‘UCX%’.

Script name	Description
`tableload`	Load in a file that was created from a tableunload. Before running tableload you should empty the contents of the table in the target database. Use “tableload HELP” for more information. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.
`tableunload`	Unload a database table to a file. This is useful for copy the contents of a table from your test environment to your production environment, for example. Use “tableunload HELP” for more information. Note, you cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.
`ucxsync`	Synchronize certain UCX tables loaded on different classic servers. For more information run “ucxsync --help”.
`tbedelete`	Delete Transit job instances records and associated artifacts by age. This script takes a numeric parameter indicating the number of days old for jobs to be deleted from the Transit database. This script can be run via cron on a regular basis to remove old jobs with status of “DONE” or “FAILED”. Issue *tbedelete –help* for more information.
`tberestart`	Runs tbestop followed by tbestart
`tbeshow`	Shows the transitexecutor process
`tbestart`	Starts the transitexecutor process
`tbestop`	Stops the transitexecutor process
`webanalyze`	Script to analyze your web.log file. You can optionally email the results to someone; setting this up in cron to run daily is a good idea. Example: webanalyze - defaults to admin/logdebug/web.log Example: webanalyze myold.web.log Example: webanalyze web.log me@myschool.edu See additional notes in the *Systems Performance* section on this topic.
`webrestart`	Runs webstop and then webstart – and optionally with debugging on.
`webshow`	Shows the web07 processes
`webstart`	Starts web07 for use with the web. See the System Admin section for more information.
`webstats`	This is a tool to produce statistics on the performance of the web daemons by analyzing the web.log file. Its primary purpose is to produce a data file containing a time series of web daemon metrics. This includes the periodic measurements of the average transaction duration and maximum transaction count for each of web07. The comma-delimited file can be input into a spreadsheet or other statistical program for further analysis and graphing. See the Performance section for more information about this command.
`webstop`	Stops web07 daemon processes.
`webtime`	Script to check how long web requests are taking to process Example: webtime - defaults to admin/logdebug/web.log Example: webtime myold.web.log

changepassword

The changepassword command changes the password stored in the shp_access_code on the shp_user_mst. It accepts a Shepherd Access ID (maximum of 14 contiguous characters) and Password (maximum of 64 contiguous characters) as input parameters. The command must be executed from the system command prompt. The input Access ID and Password are passed onto the SHP31 program which then finds the shp_user_mst for the Access ID and updates the shp_access_code with the new password.

Format:

```
changepassword <AccessId> <Password>
```

Example:

```
$ changepassword 12345678901234 this-is-my-password-with-no-spaces
```

If you do not supply both the Access ID and a Password you will be prompted for them.

Note: NO BLANKS are allowed in the Password. The first BLANK found will signify the end of the Password.

convertplans

This script is to be used to convert classic Student Educational Planner (SEP) database tables for plans into the structure required for the new, third generation of SEP delivered with the DW4.1.0 release. This script should be run prior to beginning implementation of the new generation of SEP. It may be run more than once, however all previously converted plans will be deleted and then reconverted. This means that any changes made to previously converted plans in the new generation SEP will be lost when convertplans is run again.

To execute simply run the command at the UNIX prompt. A “converting plan” message will appear as each plan is processed.

```
$ convertplans

The DAP plans will be converted to the new SEP plans.
  If this script was run before then all previously converted
  plans will first be deleted to prevent duplication.
  The old DAP plans will not be deleted.
Continue with conversion? (y/N) > y

Deleting previously converted plans...  done with deletes.

- converting plan for student 001234     - Accounting major ...
-- converted 13 courses      for plan #1
-- converted  1 notes        for plan #1

- converting plan for student 1116       - Geography major with History...
-- converted  5 courses      for plan #1
-- converted  3 notes        for plan #1

- converting plan for student 1116       - Anthropology Major ...
-- converted  5 courses      for plan #2
-- converted  3 notes        for plan #2

- converting plan for student 2055       - Zoology major - BS...
-- converted  6 courses      for plan #1
-- converted  2 notes        for plan #1
```

```
...


Converted 397 plans

Number of sep_plan             records created = 390
Number of sep_plan_term        records created = 1629
Number of sep_plan_group       records created = 1659
Number of sep_plan_class       records created = 5027
Number of sep_plan_placeholder records created = 373
Number of sep_plan_note        records created = 75
Number of sep_plan_term_note   records created = 1176
```

The classic SEP tables are converted to the new generation SEP tables in this manner:

```
DAP_PLANNER_DTL            -> SEP_PLAN
DAP_PLANCRS_DTL     -> SEP_PLAN_TERM, SEP_PLAN_GROUP, SEP_PLAN_COURSE,
SEP_PLAN_PLACEHOLDER
DAP_PLANNOTE_DTL    -> SEP_PLAN_TERM, SEP_PLAN_TERM_NOTE, SEP_PLAN_NOTE
```

Below are the details of how each SEP table is populated from the DAP tables.
A SEP_PLAN record will be created for each of the students' plans.

sep_plan	**dap_planner_dtl**	**Comments**
plan_id	dap_stu_id + "." + dap_plan_num	
student_id	dap_stu_id	
tmpl_mst_id	dap_plan_id	
description	dap_description	
is_active	dap_active_flag	
is_locked	dap_locked	If blank, will be defaulted to "N"
approval_status	dap_appr_status	If blank, will be defaulted to "NO"
school	dap_school	
degree	dap_degree	
official_tracking_status	"NOTEVALUATED"	
unofficial_tracking_status	"NOTEVALUATED"	
is_tracking_status_current	"N"	
modify_who	dap_mod_id	
modify_date	dap_mod_date	
modify_what	"CONVERSION"	
create_who	dap_mod_id	
create_date	dap_mod_date	

create_what	“CONVERSION”	

A SEP_PLAN_TERM and a SEP_PLAN_GROUP record will be created for each unique term found on the plan for each student.

sep_plan_term	**dap_plancrs_dtl**
term_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
plan_id	dap_stu_id + “.” + dap_plan_num
group_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
term	dap_term
official_tracking_status	“NOTEVALUATED”
unofficial_tracking_status	“NOTEVALUATED”
is_tracking_status_current	“N”
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

sep_plan_group	**dap_plancrs_dtl**
group_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
term_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
group_type	“UN”
group_id_parent	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
sequence	1
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

A SEP_PLAN_CLASS record is created from each course on the classic plan.

sep_plan_class	**dap_plancrs_dtl**

class_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term + “.” + dap_crs_seq
group_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
is_critical	“N”
required_term	(blank)
course_discipline	dap_discipline
course_number	dap_course_num
campus	(blank)
delivery	(blank)
credits	dap_credits
minimum_grade	(blank)
sequence	dap_crs_seq
tracking_status	“NOTEVALUATED”
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

However, if the first byte of the dap_discipline is a hyphen character then a SEP_PLAN_PLACEHOLDER record will be created instead of a SEP_PLAN_CLASS record.

sep_plan_placeholder	**dap_plancrs_dtl**
placeholder_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term + “.” + dap_crs_seq
group_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
is_critical	“N”
placeholder_type	“CONVERSION”
placeholder_value	dap_discipline + “ “ + dap_course_num
sequence	dap_crs_seq
tracking_status	“NOTEVALUATED”
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

When the DAP_PLANNOTE_DTL.dap_term field contains “PLAN” a SEP_PLAN_NOTE record is generated.

sep_plan_note	**dap_plannote_dtl**
plan_note_id	dap_stu_id + “.” + dap_plan_num
plan_id	dap_stu_id + “.” + dap_plan_num
note_text	dap_note_text (all text concatonated)
author	dap_planner_dtl.dap_mod_id
sequence	“1”
internal	“N”
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

When the DAP_PLANNOTE_DTL.dap_term field does not contain “PLAN” a SEP_PLAN_TERM_NOTE record is generated.

sep_plan_term_note	**dap_plannote_dtl**
term_note_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
term_id	dap_stu_id + “.” + dap_plan_num + “.” + dap_term
note_text	dap_note_text (all text concatonated)
author	dap_planner_dtl.dap_mod_id
sequence	“1”
internal	“N”
modify_who	dap_planner_dtl.dap_mod_id
modify_date	dap_planner_dtl.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_planner_dtl.dap_mod_id
create_date	dap_planner_dtl.dap_mod_date
create_what	“CONVERSION”

converttemplates

This script is to be used to convert classic Student Educational Planner (SEP) database tables for templates into the structure required for the new, third generation of SEP delivered with the DW4.1.0 release. This script should be run prior to beginning implementation of the new generation of SEP. It may be run more than once, however all previously converted templates will be deleted and then reconverted. This means that any changes made to previously converted templates in the new generation SEP will be lost when converttemplates is run again.

To execute simply run the command at the UNIX prompt. A "converting template" message will appear as each template is processed.

```
$ converttemplates
The DAP templates will be converted to the new SEP templates.
  If this script was run before then all previously converted
  templates will first be deleted to prevent duplication.
  The old DAP templates will not be deleted.
Continue with conversion? (y/N) > y
Deleting previously converted templates... (this will take a while) done with
deletes.
- converting template T0000002 A Geography major with History minor...
-- converted  2 courses      for template T0000002
-- converted  1 placeholders for template T0000002
-- converted  4 notes        for template T0000002
- converting template T0000003 Biology major - Chem minor...
-- converted  4 courses      for template T0000003
-- converted  1 notes        for template T0000003
- converting template T0000011 Zoology major - BS (PS)...
-- converted  3 courses      for template T0000011
-- converted  1 notes        for template T0000011
…
Converted 30 templates

Number of sep_tmpl_mst       records created = 30
Number of sep_tmpl_tag       records created = 107
Number of sep_tmpl_note      records created = 9
Number of sep_tmpl_term_note records created = 28
Number of sep_tmpl_group     records created = 57
Number of sep_tmpl_class     records created = 209
```

The classic SEP tables are converted to the new generation SEP tables in this manner:

```
      DAP_TEMPLATE_MST           -> SEP_TMPL_MST,
                                    SEP_TMPL_TAG
      DAP_PT_CRS_DTL             -> SEP_TMPL_TERM,
                                    SEP_TMPL_GROUP,
                                    SEP_TMPL_COURSE,
                                    SEP_TMPL_PLACEHOLDER
```

```
DAP_PT_NOTE_DTL             -> SEP_TMPL_TERM,
                               SEP_TMPL_TERM_NOTE,
                               SEP_TMPL_NOTE
```

Below are the details of how each SEP table is populated from the DAP tables.
A SEP_TMPL_MST record will be created for each of the templates.

sep_tmpl_mst	**dap_template_mst**	**Comments**
tmpl_mst_id	dap_plan_id	
template_id	dap_plan_id	
description	dap_description	
is_active	dap_active_flag	
term_scheme	"CONVERSION"	See "Term Scheme" note below
modify_who	dap_mod_id	
modify_date	dap_mod_date	
modify_what	"CONVERSION"	
create_who	dap_mod_id	
create_date	dap_mod_date	
create_what	"CONVERSION"	

A SEP_TMPL_TAG will be created for each curriculum value stored on the dap_template_mst.

sep_tmpl_tag	**dap_template_mst**
tag_id	dap_plan_id + "." + <tag-code>
tmpl_mst_id	dap_plan_id
tag_code	One of SCHOOL, DEGREE, MAJOR, MINOR, CONC, COLLEGE, LIBL, SPEC, PROGRAM, CATYEAR
tag_value	(value from corresponding field on dap_template_mst)
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	"CONVERSION"
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	"CONVERSION"

A SEP_TMPL_TERM and a SEP_TMPL_GROUP record will be created for each unique term found on the template.

sep_tmpl_term	**dap_templatecrs_dtl**
term_id	dap_plan_id + "." + dap_term
tmpl_mst_id	dap_plan_id
group_id	dap_plan_id + "." + dap_term
term_seq	Sequence number of term in template: 1…n
description	Term sequence plus the term code. Example: "Term #2 (201220)"
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	"CONVERSION"
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	"CONVERSION"

sep_tmpl_group	**dap_pt_crs_dtl**
group_id	dap_plan_id + "." + dap_term
term_id	dap_plan_id + "." + dap_term
group_id_parent	dap_plan_id + "." + dap_term
group_type	"UN"
sequence	1
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	"CONVERSION"
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	"CONVERSION"

A SEP_TMPL_CLASS record is created from each course on the classic template.

SEP_TMPL_class	**dap_pt_crs_dtl**
class_id	dap_plan_id + "." + dap_term + dap_crs_seq
group_id	dap_plan_id + "." + dap_term
is_critical	"N"
required_term	(blank)
course_discipline	dap_discipline
course_number	dap_course_num
campus	(blank)
delivery	(blank)

credits	dap_credits
minimum_grade	(blank)
sequence	dap_crs_seq
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	"CONVERSION"
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	"CONVERSION"

However, if the first byte of the dap_discipline is a hyphen character then a SEP_TMPL_PLACEHOLDER record will be created instead of a SEP_TMPL_CLASS record.

sep_tmpl_placeholder	**dap_pt_crs_dtl**
placeholder_id	dap_plan_id + "." + dap_term + dap_crs_seq
group_id	dap_plan_id + "." + dap_term
is_critical	"N"
placeholder_type	"CONVERSION"
placeholder_value	dap_discipline + " " + dap_course_num
sequence	dap_crs_seq
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	"CONVERSION"
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	"CONVERSION"

When the DAP_PT_NOTE_DTL.dap_term field contains "PLAN" a SEP_TMPL_NOTE record is generated.

SEP_TMPL_note	**dap_pt_note_dtl**
tmpl_note_id	dap_plan_id
tmpl_mst_id	dap_plan_id
note_text	dap_note_text (all text concatonated)
author	dap_template_mst.dap_mod_id
copy_to_plan	"Y"
internal_on_plan	"N"

sequence	“1”
internal	“N”
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	“CONVERSION”

When the DAP_TEMPLATENOTE_DTL.dap_term field does not contain “PLAN” a SEP_TMPL_TERM_NOTE record is generated.

SEP_TMPL_term_note	**dap_templatenote_dtl**
term_note_id	dap_plan_id + “.” + dap_term
term_id	dap_plan_id + “.” + dap_term
note_text	dap_note_text (all text concatonated)
author	dap_template_mst.dap_mod_id
sequence	“1”
internal	“N”
modify_who	dap_template_mst.dap_mod_id
modify_date	dap_template_mst.dap_mod_date
modify_what	“CONVERSION”
create_who	dap_template_mst.dap_mod_id
create_date	dap_template_mst.dap_mod_date
create_what	“CONVERSION”

Term Scheme

When templates are converted the SEP_TMPL_MST is created with a term_scheme of “CONVERSION”. When you open the template in the Template Management user interface you will find that this term scheme is not valid in UCX-SEP002. You need to change the term scheme for each of your templates to a valid term scheme you have setup in UCX-SEP002. For example, you may have one template with five terms – three fall terms and two spring terms. You will need to create a new term scheme in UCX-SEP002 perhaps called “FIVE_TERMS” and set the term scheme for your template to “FIVE_TERMS. However, you might also encounter another template with five terms but this one might contain two fall terms and three spring terms – a different combination than the previous one. You should not use the “FIVE_TERMS” term scheme because the arrangement of term types is different. You should then create a different term scheme perhaps called “FIVE_TERMS_B” and assign the appropriate term types to each record.

Before creating any new term schemes in UCX-SEP002 you should first review all of your templates to see what types of term schemes are needed. Doing this will help you create sensible term scheme names and may convince you that some templates need to be changed to conform to a standard set of term schemes. You should not create a term scheme called “CONVERSION” in UCX-SEP002 to get around this issue since it will lead to many problems including errors when creating plans from templates. If you find that all or most of your templates do conform to the same type of term scheme then you may want to us SQL to update the term_scheme field on the sep_tmpl_mst to your newly created term scheme name.

Regardless of the approach you take you should review each and every template for accuracy and completeness.

Review the UCX-SEP002 documentation for more information.

dapauditstopdffiles

This script allows you to create a PDF file for each audit ID specified in the input file. You can generate a file of audit IDs for students who have graduated or simply for archival purposes.

You run the script specifying two parameters. The first parameter is the file of audit IDs while the second parameter is the name of the FOP XSL stylesheet – located in the local/xsl directory. For example:

```
$ dapauditstopdffiles gradstudents.txt myfopaudits.xsl
```

Will use the myfopaudits.xsl file located in local/xsl to process each of the audit IDs found in gradstudents.txt.

You may also want to place a list of audits for some of your student athletes into a file and specify the athletic eligibility stylesheet.

```
$ dapauditstopdffiles stuathletes.txt myfopaudits-athl.xsl
```

When creating PDF files you should run your athletic audits separate from your financial aid audits separate from your academic audits. You need to do this because you will want to specify a different FOP stylesheet for each type of audit.

When getting started, it is recommended you perform a test with 10 audit-ids in a file. Make sure the files get created without error. Also check the size of the files created. You can then extrapolate based on these 10 files to see how much free space you will need on the system when you run the script against your big list of audits.

The PDF creation process is not fast; it may take several seconds per audit. You may want to run the job in the background to allow it to run overnight in the background, as follows:

```
$ dapauditstopdffiles gradstudents.txt myfopaudits.xsl > pdf.out 2>&1 &
```

When it completes you can review the pdf.out file for errors.

The format of each PDF file generated is: <stuid>~<school>~<degree>~<auditid>.pdf
Example: 9837631~UG~BA~AA000123.pdf

The files are placed in the current directory so you may want to create a special directory in which to run this script.

Note: You may want to archive particular frozen audits. You should decide which frozen audits you want to archive and then run SQL like that below to export these IDs to a file. Don’t forget to specify the audit-type also to get either the Academic Audits (AA), Athletic Eligibility Audits (AE) or Financial Aid audits (FA). See UCX-AUD032 for a list of the freeze-type values.

```
select dap_audit_id from dap_audit_dtl
```

where dap_freeze_type='FRZTYP' and dap_audit_type='AA'

order by dap_audit_id

dapauditstoxmlfiles

This script allows you to create an XML file for each audit ID specified in the input file. You can generate a file of audit IDs for students who have graduated or simply for archival purposes.

You run the script specifying a single parameter - the file of audit IDs. For example:

```
$ dapauditstoxmlfiles gradstudents.txt
```

Will process each of the audit IDs found in gradstudents.txt.

When getting started, it is recommended you perform a test with 10 audit-ids in a file. Make sure the files get created without error. Also check the size of the files created. You can then extrapolate based on these 10 files to see how much free space you will need on the system when you run the script against your big list of audits.

Unlike the corresponding PDF script, the XML creation process is fairly fast. However, you still may want to run the job in the background to allow it to run in the background – like this:

```
$ dapauditstoxmlfiles gradstudents.txt > xml.out 2>&1 &
```

When it completes you can review the xml.out file for errors.

The format of each XML file generated is: <stuid>~<school>~<degree>~<auditid>.xml
Example: 9837631~UG~BA~AA000123.xml

The files are placed in the current directory so you may want to create a special directory in which to run this script.

Note: You may want to archive particular frozen audits. You should decide which frozen audits you want to archive and then run SQL like that below to export these IDs to a file. Don't forget to specify the audit-type also to get either the Academic Audits (AA), Athletic Eligibility Audits (AE) or Financial Aid audits (FA). See UCX-AUD032 for a list of the freeze-type values.

select dap_audit_id **from** dap_audit_dtl
where dap_freeze_type='FRZTYP' and dap_audit_type='AA'
order by dap_audit_id

dapauditstoxml

Extracts all audits and converts to xml using the getxmlaudit script. All audits are placed in a single XML file called allaudits.xml. Each audit is enclosed within start and end <Audit> xml tags as with Web XML audits.

The SQL WHERE clause in the CreateSQLFile function may be modified to select a subset of your audits based on date, audit-id, school, degree, student level, etc – anything on the dap-audit-dtl record. You may want to run this script several times using different criteria to keep the resulting xml file from getting too large. A single audit can easily be 100K meaning an extract of 100 audits would result in at least a 10MB file. You should conduct a few tests to see how big your files will be and adjust the WHERE clause accordingly – the size of audits differs from school to school.

To execute simply run the command at the UNIX prompt. The allaudits.xml file will be placed in the current directory. A “Processing audit” message will appear as each audit is processed.

```
$ dapauditstoxml

Processing audit = [AE000268]...
Processing audit = [AE000273]...
Processing audit = [AE000269]...
Processing audit = [AE000259]...
```

You may redirect the output to a file if you do not wish to see the Processing messages:

```
$ dapauditstoxml > outputfile
```

You may want to compress the big xml file using some compression tool (like gzip) and move the file off the system. You can always uncompress the file and view and search through the xml data as needed.

dapblockinsert

Place the blocks you want loaded into the admin/blocks directory.

Ensure that the blocks directory is empty before placing your new set of files there.

The script loads the dap_req_block table with the blocks found. The text of the block is loaded into a CLOB field in the table.

Run the script to load the blocks. Enter the appropriate catalog years to use for all blocks when prompted. These catalog years are used as the default in case a block is encountered that does not have the catalog years defined.

```
$ dapblockinsert
Please enter the start catalog year > 20072008
Please enter the stop catalog year  > 99999999
```

You will see messages go to the screen. When it gets through all of the blocks it will process the SQL Insert statements – this may take a few minutes if you have a lot of blocks.

The script writes information to a **dapblockinsert.log** file in the **logdebug** directory. Please review it.

Be sure to run dap16 to parse the blocks and then fix the errors using Scribe.

Each file must contain header information and start with two #’s characters:
Line 1: can be the school name but it is ignored so it really can be anything
Line 2: must contain the block type and value
Line 3: must contain the block title
Line 4: optional – can contain the starting and ending catalog years separated by a hyphen.

Example:

```

##Ellucian University
##MAJOR=ACCT
##Major in Accounting
```

```
##2012-9999

BEGIN

;

END.
```

dapfindbadaudits

Shows a list of audits by student ID, audit ID and audit date that are believed to be corrupt. The user is then given the opportunity to delete these bad audits and also is given directions on how to run new audits for the students with these corrupt audits. When new audits are created and saved to the database Degree Works may encounter a problem saving the new records to the dap_audtree_dtl table. This is usually caused by running out of space in the database. When this occurs Degree Works tags the dap_audit_dtl record and it is this tag that this script looks for to report the corrupted audits.

Regardless of whether the user answers Y or N to delete the audits a file of student IDs is saved to the admin/data directory. This file can then be run with the dap22ids script to create new audits for these students.

```
$ dapfindbadaudits
These are the audits that are most likely corrupt.
This corruption is usually caused by running out of room in your database.
Here you see the student ID, audit ID and create date for each corrupt
audit.

Processing .. findbad.sql.5579
select dap_stu_id, dap_audit_id, dap_audit_date from   dap_audit_dtl where

STUDENT_ID AUDIT_ID CREATE_DATE
========== ======== ======================
N88665547  AA044158 20100527
N00010771  AA044159 20100527
N00010771  AA044160 20100527
N00011380  AA044162 20100527
N00011380  AA044163 20100527
N00011380  AA044164 20100527

6 rows selected
Do you want to delete these bad audits? (y/N) > y

Deleting bad/corrupt audits now...
Deleted 6 audits from the database
The list of students with bad audits has been saved to this file:
/dwprod/admin/data/studentswithbadaudits.ids
You may run "dap22ids studentswithbadaudits.ids" to create new audits for
these students.
```

dapfindorphanedaudits

This script shows a list of audits by student ID, audit ID and old degree and new degree believed to be orphaned. That is, these audits exist for students who have changed their degree since the audit was generated. The user is then given the opportunity to delete these orphaned audits and the CPA. Note that frozen audits are ignored as are what-if audits.

Please note that this script removes all CPA data for the student identified with the orphaned audit; the script does not only delete the CPA records for the audit. You may want to recreate the CPA data for the students' newest audit after this script deletes the CPA data.

```
$ dapfindorphandaudits
These are the audits that belong to students who changed their degree.
Here you see the student ID, audit ID and old and new degrees for each.
(Note that frozen audits are ignored; they are not considered orphaned.)
(Note that what-if audits are also ignored.)

Processing .. findorphaned.sql.17139
select dap_stu_id Student_Id, dap_audit_id Audit_id, dap_degree Old_Degree,

STUDENT_ID AUDIT_ID OLD_DEGREE   NEW_DEGREE
========== ======== ============ ============
210009206  AA043626 BA           BBA
210009301  AA043632 BA           DIPL
210009206  AA043703 BA           BBA
210009301  AA043709 BA           DIPL
210009206  AA043780 BA           BBA
210009301  AA043786 BA           DIPL
HERMIONE   AA045100 MA           BS
HERMIONE   AA045101 BA           BS
HERMIONE   AA045103 MA           BS
HERMIONE   AA045104 BA           BS
HERMIONE   AA045351 GRAD_MA      BS
HERMIONE   AA045353 MA           BS
HERMIONE   AA045354 BA           BS
HP         AA045372 BA-ENGLISH   BFA
HP         AA045372 BA-ENGLISH   BA
HP         AA045372 BA-ENGLISH   BS

16 rows selected
Do you want to delete these orphaned audits and associated CPA data? (y/N) >
Orphaned audits have not been deleted.
```

dapmapcopy

The dapmapcopy command allows you to copy the mappings for one school to another school. The command must be executed from the system command prompt. The dap-mapping-dtl, dap-map-cond-dtl and dap-title-dtl records are copied from one school to another.

Format:

```
dapmapcopy <from-id> <to-id>
```

Example:

```
$ dapmapcopy 004002 554987
```

If you do not supply both school IDs you will be prompted for them.

A high-level report of the steps dapmapcopy is taking is displayed to the screen. The high-level in addition to the low-level details and errors are sent to $DGWHOME/tmp/dapmapcopy.log. You may examine this log as needed.

The command unloads the all existing mappings for the to-id to $DGWHOME/archive/mappings and then deletes them from the database. In doing this, mappings can be copied from a golden copy multiple times as changes are made to the golden copy.

The school Ids used should already exist in the ETS-MST.

You may create a file containing multiple dapmapcopy commands to copy your mappings en masse. For example, if your golden copy of mappings is for school ID 004002 you can copy the mappings to multiple schools by creating the following lines in your script:

```
dapmapcopy 004002 123456
dapmapcopy 004002 123887
dapmapcopy 004002 323901
dapmapcopy 004002 349434
dapmapcopy 004002 990876
dapmapcopy 004002 884930
```

Before creating a file such as this be sure to confirm that running the command once gives you the desired results. Be sure to check the results using Transfer Equivalency.

Example output from dapmapcopy:

```
$ dapmapcopy
Copy mappings from school ID        > 004002
Copy the mappings to this school ID > 554987
=== Copying mappings from school 004002 to school 554987 ===
 == Unload mappings for school 004002 ==
 == Archive old mappings for school 554987 ==
 == Delete old mappings for school 554987 ==
 == Get the next mapping ID from dap-next-id-mst ==
 == The next mapping ID = MA092015 ==
 == Change mappings files to record oriented and rename to new ID ==
 == Change map and school ID on mappings ==
 == Change mappings files to stream oriented ==
 == Update next map-id in dap-next-id-mst ==
 == Load the new mappings for school 554987 ==
 == Reindex the mappings tables ==
=== DONE Copying mappings from school 004002 to school 554987 ===
```

dbbuild

This script creates all the database objects needed by the Degree Works software. It generates the table definitions from the dwschema.xml definition file, using the share.xml configuration file, which defines the relationship

between the various schema owners and their shared tables. It also loads all the necessary ancillary views and packages.

Options:

--banner	Generate banner views and packages (default).
--debug	Output debugging messages to stderr.
--help	Display the help screen.
--nobanner	Do not generate banner views and packages.
--pause	Pauses the script after each step.
--share	Specifies the sharing configuration file.
--usage	Display command syntax only.
--verbose	Display progress information.
--version	Display the version of this script.

The sharing configuration file must be configured for your particular sharing needs. It can be provided as the parameter to the --share option, or it will default to share.xml in the schema directory.

The --dbuser option can provide a database user that has the ability to create tables for all the schemas. If this option is omitted, then the value of the DB_LOGIN environment variable is used.

An experienced user can use the --pause option to step through specific parts of this command. At the beginning of each step, you will be prompted to either stop, continue, or skip just the next step.

The database, tablespaces, and users must be created prior to running this command. They are not created by the command.

FILES:

$DGWHOME/schema/dwschema.xml - is a file that contains the xml definition of all the required tables. This file is provided by Ellucian, and should not be modified by the client.

$DGWHOME/schema/share.xml - the default value for the file that defines the various schema owners and the tables they share. For multi-entity clients, this file must be configured and provided via the --share option.

dwsettings

This script is used to import settings into the shp_settings_mst, export settings into a file, delete settings from the shp_settings_mst, or to overwrite settings from the shp_settings_mst. Additional options allow settings to be encrypted during an import or decrypted on an export.

For an import, the input file can be a partial list of settings and can be an XML file or a properties file.

For an export, the output file is always created as an XML file.

For a delete, the input file must be a properties file.

For an overwrite, the input file can be a partial list of settings and can be an XML file or a properties file.

Options:

`--import`	Sets the mode to import; the database will be updated.
`--export`	Sets the mode to export; the data will be written to a file.
`--delete`	Deletes the settings from the database. The file specified should be a properties file format.
`--overwrite`	Causes import to overwrite existing data, which it will not normally do.
`--key`	Restricts the import and export to keys beginning with the keypattern.
`--spec`	Define the spect to delete a key or a group of keys
`--encrypt`	Causes import to encrypt specific entries, such as passwords. Without this keyword, the data file is assumed to be already encrypted
`--decrypt`	Causes export to decrypt specific entries, such as passwords. Without this keyword, the data is exported encrypted.

One and only one of `--import`, `--export`, or `--delete` may be used.
`--overwrite` and `--encrypt` can only be used in import mode.
`--decrypt` can only be used in export mode.
The `--key` flag cannot be used with multiple files.

Examples of how to run dwsettings:

```
dwsettings import infile.xml
dwsettings --import infile.xml
```

<< specify infile.xml as the input file, existing settings will not be overwritten >>

```
dwsettings import infile.properties
dwsettings --delete delete.properties
```

<< specify infile.properties as the input file >>

```
dwsettings import infile.xml overwrite
dwsettings --import --overwrite infile.xml
dwsettings --import --overwrite --encrypt infile.xml
dwsettings --import --encrypt infile.xml
dwsettings --import --overwrite delete.properties
```

<< specify that all settings should be updated and overwritten>>

```
dwsettings export outfile.xml
```

<< specify outfile.xml as the output file >>

```
dwsettings export outfile.xml classicConnector
dwsettings --export --decrypt outfile.xml
dwsettings export outfile.xml listofSpecificKeys
dwsettings --export --key listofSpecificKeys --decrypt outfile.xml
dwsettings --export --key listofSpecificKeys infile.xml
dwsettings --delete --key listofSpecificKeys
dwsettings --delete --spec test --key listofSpecificKeys
```

<< specify that all listofSpecificKeys entries are to be exported >>
When a key is specified a LIKE is performed; example:
LIKE shp_settings_key LIKE '%listofSpecificKeys%'#

```
dwsettings --delete outfile.properties
dwsettings --delete --key listofSpecificKeys
dwsettings --delete --spec test --key listofSpecificKeys
```

<< outfile.properties should be a list of settings to delete, one setting per line >>
Note: The key is case-sensitive.

getxmlaudit

Extract specified audit from the db and convert to an xml file. The file name created will be the audit-id with a .xml extension. The file will be placed in the current directory. The file will contain two xml commands.

```
$ getxmlaudit AA000123
```

This will create a file called AA000123.xml.

launchjob

This script launches a Transit job by contacting the Transit API. This script does not run the job itself but submits it to the Transit job queue. The job may run on any available executor, possibly on another server. The command must be run in an initialized Degree Works environment. More information about this script can be found in the *Transit Administration Guide*.

packdebug

The packdebug script can be used to collect debug files from the server when a user has enabled debugging from the Java applications running on the Java Application Servers. It reads debugging output from the specified log files by sessionId and creates a zip file with an encrypted key.

The script should be run in the target directory where log files exist, or should be provided the complete log file path in the parameters. If the input files do not exist, no files will be processed.

OPTIONS:

`--key <mykey>`	A user specified password for output zip file.
`--output <outputZipName>`	A user specified file name for output zip file.
`--session <sessionId>`	The debug sessionId provide in the UI when debug was enabled.

EXAMPLES:

```
packdebug --key myKey --output myOutputFile --session 2669bb29-781e-4be4-8a5d-
c348ffbd24bc scribe.log

packdebug --key myKey --output myOutputFile --session 2669bb29-781e-4be4-8a5d-
c348ffbd24bc /opt/apache-tomcat2/logs/scribe.log

packdebug --key myKey --output myOutputFile --session 2669bb29-781e-4be4-8a5d-
c348ffbd24bc /opt/apache-tomcat2/logs/@.log
```

profiledbg

Extract performance profiling data from a Degree Works classic debug file. This command is intended for use by the Degree Works development team, and should only be upon consultation with Ellucian.

SYNOPSIS

```
profiledbg [--verbose] [--output <file>] file [file2 ...]
```

DESCRIPTION

This command creates a data file to be used for profile analysis. It uses a file created by the classic Degree Works debugging tool logdebug as input. It matches up LDTime benchmark entries in the file and outputs a comma delimited file with the mark and duration for each transaction. This can be imported into a spreadsheet to create a profile table giving counts average and total durations, etc. To get a debug file to use with this command, export DWBENCHMARK=1 before launching the program being profiled. It can also be used on debug files created with the DWDEBUG environment variable set. If multiple debug files are given, then the output of each will be concatenated into the one output file.

This command may output a message to stderr if it cannot find a matching LDTime begin mark for the end mark it is processing. This is generally the result of a programming error of some sort. For example, exiting a routine without issuing a DEBUG_MODULE_END. It may or may not throw off results. The errant mark is simply discarded.

If no output file is provided, the script will output to a file named profiledbg.out. If you use a single dash ("-") as the output file name, the output will be sent to stdout.

The `--verbose` switch provides progress notification as the script is executing, including progress dots every 100 lines of file input.

FILE OUTPUT

The output file will be in the format:

```
Mark ID,0:0:1.234567890
```

The Mark ID column may have spaces or special characters, but it should not have commas, or that will impede importation into a spreadsheet. There are no guarantees as to the number of digits in the nanoseconds, although it will be 9 or less.

This file can be imported into Excel or another spreadsheet. A pivot table can then be constructed using the Mark as the row and, as columns, a count of the marks, an average of duration, max of duration, standard deviation of duration, sum of duration, and % of column total.

sharegen

The sharegen tool is intended for use in creating and maintaining the Degree Works database. It produces schema file(s) for the purpose of separating shared and unique tables for a Degree Works database that supports multiple institutions. As input, it takes an xml document describing the database structure (dwschema.xml), an xml document describing the sharing relationships between different institutions (share.xml), and the current database structure as divined from the native database. As output, it creates a set of sql ddl scripts to create and/or alter the database to fit the input specification. Output files can be generated in xml or sql format. In order to modify the database, sql files must be generated. The type and location of output files is defined in share.xml.

The sharegen script is called by the **dbbuild** script (it is never called directly). Before running dbbuild, any new schemas (users) defined by share.xml must be created in the database. The suggested procedure for setting up a schema/user named “dwschema” (for example) is to run the following commands using SQLPlus and substituting the mypassword and tablespace name dgw with a localized values.

1. create user *dwschema* identified by *mypassword* default tablespace *dgw* quota unlimited on *dgw* temporary tablespace temp;
2. grant connect to *dwschema*;
3. grant dba to *dwschema*;
4. Verify the user was created: select * from all_users order by username;

dwschema.xml

This file is hand maintained and contains the general structural information for a Degree Works database, such as dapdb, raddb, etc. It defines the tables, keys, and indexes for the database. It uses the Torque/DdlUtils XML database schema DTD for the definition, which can be found at http://db.apache.org/ddlutils/schema. The file is located in $DGWHOME/schema/dwschema.xml. Note that dwschema.xml is provided and maintained by Ellucian and should not be modified.

share.xml

This file is hand maintained and contains the configuration data defining the sharing relationships between client entities. It defines each entity, and groups of these entities, and the tables they share. This XML schema is unique to Ellucian. The default file used for a single school is located in $DGWHOME/schema/share.xml. An example file for multiple schools sharing some of the Degree Works tables located in $DGWHOME/schema/mepshareexample.xml An xsd schema file is used to validate the structure and content of share.xml. It is located in $DGWHOME/schema/dwshare.xsd.

More information on each of the important values in share.xml:

```
$ sharegen –-sourceSchema dwschema.xml –-shareInput share.xml
```

Depending on the configuration of share.xml this will produce one or more schema xml or sql files.

Usage

```
sharegen [<configfile>] --sourceSchema <sourceSchema>
[--shareInput <shareInput>] [--validateOnly <validateOnly>]
[--changeDatabase <changeDatabase>] [--createTables <createTables>]
[--dropTablesFirst <dropTablesFirst>] [--continueOnError <continueOnError>]
```

```
[--dburl <dburl>] [--tnslocation <tnslocation>] [--usetns <usetns>]
[--tnsname <tnsname>] [--username <username>] [--password <password>]

  --sourceSchema <sourceSchema>
        The source schema file to use. Defaults to dwschema.xml.

  [--shareInput <shareInput>]
        The schema for which to generate xml schema. Defaults to share.xml.

  [--validateOnly <validateOnly>]
        Set validateonly=1 to validate share xml and schema xml. (default: 0)

[--changeDatabase <changeDatabase>]
      Set 1 to change/modify database immediately after creating output files.
      (default: 0)

[--createTables <createTables>]
      Set 1 to specify that CREATE TABLE statements will be generated. Set to
      0 if ALTER TABLE statements should be generated for objects that already
      exist in the database. (default: 0)

[--dropTablesFirst <dropTablesFirst>]
      Set 1 to output DROP TABLE statements in sql when createTables=1. This
      has no effect when createTables=0. (default: 0)

[--continueOnError <continueOnError>]
      Set 1 to continue processing live database changes even if errors occur.
      (default: 0)

 [--dburl <dburl>]
        The schema for which to generate xml schema. A jdbc string is built
using
        values from $DB_LOGIN, $TWO_TASK and $DB_PORT_NBR system variables.

  [--tnslocation <tnslocation>]
        The location of tnsnames.ora file. By default supplied by .config file
or script.

  [--usetns <usetns>]
        Set 1 to use tnsname or 0 to use local database. (default: 0)

  [--tnsname <tnsname>]
        TNS name to use when usetns=1

  [--username <username>]
        The database username
  [--password <password>]
        The database password
```

Default Values

The sharegen script is not run directly; it is called by the dbbuild script. The sharegen script contains default values for dburl, sourceSchema and schemaInput, so those values do not normally need to be entered. When the dbbuild script is run, those values cannot be overridden on the command line. Alternatively a config file named sharegen.config can specify default values for input parameters.

shareinfo

The shareinfo command generates a list of database owners and associated table names for table names starting with DAP, RAD, SHP or UCX.

Format:

```
shareinfo
```

Example:

```
======== OWNER ==============  ========TABLE =========

DGWDBA                         DAP_APPLICNT_MST
DGWDBA                         DAP_AUDIT_DTL
DGWDBA                         DAP_AUDTREE_DTL
DGWDBA                         DAP_COLLEGE_DTL
DGWDBA                         DAP_EQV_CRS_MST
DGWDBA                         DAP_EXCEPT_DTL
… (and rest of associated tables and any additional users)
```

Degree Works Security Options

Security in Degree Works consists of logon authentication, service authorization, and user-class assignment. This document explains the options provided with Degree Works for controlling security. When reviewing this document, keep in mind whether or not a database outside of Degree Works already contains passwords for accessing services on your Web site. Think about granting access to Degree Works and services within Degree Works. If your site already has a method for granting and denying access to services then you may want to continue to use that mechanism for Degree Works. Use this document to help answer the following questions:

1. What types of users will be accessing Degree Works via the Web?
   - Students?
   - Advisors?
   - Applicants?
   - Athletic Department?
   - Registrar?
   - Financial Aid Office?

2. What Degree Works services on the Web should be accessible to these types of users?
   - Running audits?
   - Reviewing audits?
   - What-If audits?
   - Adding, modifying, deleting, or reviewing notes?
   - Adding or modifying exceptions?
   - Searching for students?
   - Petitions?
   - Student Planner?
   - GPA Calculator?

3. Which users need access to Scribe for writing degree requirements?

4. Which users need access to Transit for running batch audits and processes?

5. Which users need access to Controller for maintaining UCX codes and Shepherd Settings?

6. Which users need access to Transfer Equivalency administration?

7. When a logon to Degree Works is requested, where will the user's credentials be authenticated?
   - In Degree Works?
   - In your own central service?

These questions help determine how Degree Works is installed and which data should be sent via the Bridge for storage in Degree Works.

HTTPS/SSL

It is strongly recommended that all Java application servers (e.g. Tomcat) running Degree Works be configured with SSL certificates and to only allow HTTPS/SSL access to applications. Documentation for SSL certificate setup is widely available from each vendor or product website. Signed certificate providers provide additional documentation that relates their products and specific to many common platforms. Clients are responsible for setting this up as part of installation and configuration.

As a best practice, it is recommended that the Java application server be configured to not support weak SSL protocols such as SSLv2, SSLv3, TLS 1.0 & TLS 1.1. For example, for applications deployed on Tomcat, you should configure your <Connector> in server.xml with these two attributes:

```
sslProtocol="TLS"
sslEnabledProtocols="TLSv1.1+TLSv1.2"
```

For Transfer Equivalency Self-Service and Composer (applications with embedded Tomcat 8), you should add this environment variable to your deployment configuration to disable insecure protocols, leaving only TLS 1.1 and 1.2 enabled:

```
SERVER_SSL_CIPHERS=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WI
TH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,TLS_DHE_RSA_WIT
H_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_
WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE
_ECDSA_WITH_AES_128_CBC_SHA256
```

Authentication

Logon authentication determines if the user is who they say they are. The user typically enters an Access ID and Access Code (password), which is authenticated in a security database. The security database can either be the one supplied with Degree Works, one affiliated with the student system, or some institution wide central service such as CAS. Logon authentication occurs prior to granting access to Degree Works.

One of several options for single sign-on (SSO) may be employed when Degree Works is integrated with other systems. Degree Works supports SSO for Luminis, Self-Service Banner, CAS, and SAML 2.0. External access managers such as Oracle Access Manager are also supported.

The “changepassword” script may be used to change the password (shp_access_code on the shp_user_mst) from the command line. Refer to the *Special Scripts* section for more details on the “changepassword” script.

Banner Sites Only
The Banner extract programs generate default passwords for each student, advisor and staff member processed. The password will be loaded with one of two forms:

(1) Password generated from custom SQL
(2) 10-byte random password.

If custom SQL exists in integration.banner.extract.config for the appropriate keyword (PASSWORDSTU, PASSWORDADV or PASSWORDSTF) it will be used to read the Banner data and format it into a password (1 to 64-byte password) that will be loaded into the shp_access_code on the shp_user_mst for the individual. If the custom SQL is blank or the password generated from the custom SQL is blank a 10-byte random alpha-numeric password will be generated.

A ‘Change Password’ configuration flag exists in the UCX_CFG020 “WEBPARAMS” record. If this flag is set to ‘N’ then the shp_access_code on the shp_user_mst will NOT be changed by the Banner extracts, only added when the student, advisor or staff record is originally created.
Refer to the R171SHPU details in the *Banner Data Mapping for BIF Technical Guide*.

Multiple paths to Authentication

When a non-authenticated user attempts to access a secured Degree Works application, the user will normally be redirected to a login page. There are three possible scenarios for this:

1) Native Degree Works login
2) CAS single sign-on login
3) SAML single sign-on login.

With native Degree Works login, the user credentials can be validated against either the native Shepherd user database or an external LDAP server.

To configure the appropriate login scenario, change the Shep setting `core.security.authenticationType` to one of “**SHP**”, “**CAS**”, or “**SAML**”. The configuration requirements for each of these models are described below.

The authenticationType tells Degree Works where, if you are not already authenticated, to send you for authentication. If it is “SHP”, then you will get the built-in Degree Works login page. “CAS” will send you to a CAS server for login, and “SAML” will send you to a SAML server. When the type is set to CAS or SAML you, therefore, cannot use the built-in login pages with the typical Degree Works manager user. This authenticationType is used by all Degree Works applications unless you create separate shep settings for each application. See the *Multiple Authentication Entry Points* section below for information on how to do that.

As an alternative, you can configure an external access manager, such as Oracle Access Manager or Shibboleth, to handle authentication. In this scenario all requests are intercepted before they get to Degree Works, authentication is handled by the external access manager, and only authenticated requests are forwarded to Degree Works. See the *External Access Manager**Error! Reference source not found.*** section below for more configuration information on this option.

Degree Works Native Login

Degree Works applications contain a login screen that can be used to collect users’ credentials for access to the application. Each application is independent, and usually requires a separate login for each one. We recommend that you look at CAS to provide a single sign-on experience. To use the native login screen, set the Shep setting `core.security.authenticationType` to “SHP”.

The user’s credentials can be validated either using the Degree Works Shepherd user database that is populated via the bridge, and/or they can be validated using an external LDAP server. If both are configured, then a valid entry in either database will authenticate the user.

Shepherd User Database

Degree Works provides a basic user security database for logon authentication and service authorization. In this model, a shp-access-id and shp-access-code on the shp-user-mst is used for each student, advisor, faculty, staff, and administrator authentication.

This model requires that clients send the user-class to the Bridge in the SHPU record. For Banner clients this is handled by the Banner extract. For Colleague clients this is handled by the Colleague extract.

To enable this type of authentication you must set several Shpherd settings in Controller:

- Set the `core.security.authenticationType` to “SHP”. This will direct users to the native Degree Works login page and authenticate their credentials against either the shp-user-mst and/or and LDAP database, depending on the following settings.

- Setting `core.security.shp.authentication.enabled` to “true” causes the login credentials to be authenticated against the shp-user-mst.

- Setting `core.security.ldap.enable` to “true” causes the login credentials to be authenticated against an LDAP database. See the following section on **Error! Reference source not found.** for more configuration requirements. Either this setting or `core.security.shp.authentication.enabled` must be set to “true” when using “SHP” authentication. If both are set, valid credentials in either one will allow access to Degree Works.

- Setting `core.security.passwordEncoding.enabled` to “true” causes Controller and the Java bridges to encrypt the password. Note, however, that the Banner Extract as well as the RAD extract still follows the `Encrypt Password` flag in the UCX-CFG020 WEBPARAMS record in Controller.

- Setting `core.security.passwordCheck.sha1.enabled` to “true” causes the user’s password to be encoded with the SHA1 hash when comparing against the database password. If the `core.security.passwordEncoding.enabled` is set to “true”, so should this setting.

- Setting `core.security.passwordCheck.clearText.enabled` to “true” allows the software to check the clear text (unencoded) password against the database. You would do this if you have passwords that have not been encoded. For example, if you turn on password encoding and do not reload all the users’ passwords, then you will have some passwords stored in the database in unencoded form. You would need to set this to “true” in order for those passwords to work.

- The `core.security.shp.maxLoginAttempts` setting specifies how many invalid login attempts are allowed before further logins are ignored.

- The `core.security.shp.failLoginResetMinutes` specifies the amount of time in minutes after the maximum number of login attempts have been exceeded before the user is allowed to attempt another login.

Refer to the reference documentation on Shepherd Settings for more information about these settings.

LDAP User Database

LDAP is frequently used as a central repository for user information and as an authentication service. Degree Works can be configured to use an LDAP server instead of or supplementary to Degree Works native authentication (SHP database). LDAP authentication is enabled by changing the `core.security.ldap.enable` to “true”. The setting `core.security.authenticationType` must be set to “SHP”. Both LDAP and native SHP authentication can be enabled at the same time. In this case, valid credentials in either database will allow access.

No authorization (keys and services) information will be retrieved from LDAP. This is still controlled by Shepherd. An LDAP user must have an LDAP attribute that identifies their user records (shp_user_mst) in Degree Works. See *Locating Degree Works ID within LDAP*.

LDAP can be configured for authentication in several different ways. You should be generally familiar with LDAP before configuring Degree Works to use it. These instructions do not cover the normal set up and use of an LDAP server.

You must tell the authentication software the location of the LDAP server by setting `core.security.ldap.serverUrl`. For example, this might be something like "ldaps://ldap.myschool.edu:636/ou=users,dc=myschool,dc=edu". You should use the secure LDAP protocol for this.

An administrative user is required to find the user's record and read attributes in the LDAP database. You configure this user's dn in `core.security.ldap.adminDn` and their password in `core.security.ldap.adminPassword`. This user must have the ability to search for and return the user's distinguished name (dn) as well as to read any users attributes. The password is stored in encrypted form in the settings database.

Degree Works authenticates the user by binding to LDAP with the user's dn. It determines the dn in one of two possible ways. The first way is to use a pattern set in `core.security.ldap.userDnPattern`. The pattern should contain the token "{0}" that will be replaced by the user's Access ID (login name). An example would be "uid={0}". This is relative to the base given in the `core.security.ldap.serverUrl` setting. So, if a user with the id "bobstudent" logs in, and the serverUrl is set to "ldaps://ldap.myschool.edu:636", and the userDnPattern is set to "uid={0}", then the software would expect the user's unique dn to be "ou=users,dc=myschool,dc=edu,uid=bobstudent".

The second way to retrieve the user's dn is to do a search with an LDAP filter expression. This follows the format specified in RFC 4515. You should include a special token, "{0}", in your expression that will be replaced by the user's Access ID (login user name). For example, "(uid={0})". The scope of this search may be further limited by setting a search base in the setting `core.security.ldap.userSearchBase`. For more information about search filters see https://tools.ietf.org/html/rfc4515.

Once the distinguished name is found, the user will be authenticated with a direct bind to the LDAP server using the password provided by the user in the login process.

Locating Degree Works ID within LDAP

When authentication is successful, then Degree Works needs to find out how to locate or "map" the LDAP user to Degree Works internal database in order to determine their authorities. The username provided may not be the Degree Works ID, which is the ID from the Student Information System, and it may not be the primary Access ID for the user in the Shp user database. We map an attribute in the LDAP server to the *Alternate ID* in the user's shp_user_mst record via Controller. This should be bridged into Degree Works from the student system. You specify which LDAP attribute to use for this mapping in the setting `core.security.ldap.studentId.attribute`. It should be a unique attribute unless you are using the suffix, as described below.

Suppose that all Degree Works users have an attribute in LDAP named "employeeNumber" (an arbitrary example chosen from inetOrgPerson schema). You would need to populate that value into the Alternate ID field in the Shp User Record, usually using the bridge. Refer to the appropriate bridge document for your SIS for instructions on how to do this.

It is possible to have multiple attribute values for the user's Degree Works ID. In this case, you must append a suffix to the Degree Works ID value in the LDAP record. You should use the same suffix for every user, and you would define this suffix in the setting `core.security.ldap.studentId.suffix`. So for example, you may use the attribute named "externalId" for multiple systems, with Degree Works being just one of many, and each system may need a different ID value. You would define a suffix, for example "::DGW" and append this suffix to each ID entered in LDAP for this attribute. For example, bobstudent may have an externalId attribute with a value of "12345668::DGW". Degree Works will locate the value with the given suffix and remove the suffix before it looks up the students shp_user_mst record. If you use the suffix, every user must have this suffix appended to their attribute value.

CAS Single Sign-On

The Central Authentication Service (CAS) can be used to integrate Degree Works with portals and other Web applications. CAS provides an open, well-documented protocol and an open-source Java server component. CAS supports LDAP, Active Directory, and other data sources for single sign-on. More information about CAS is available at http://www.jasig.org/cas.

Functionality

CAS provides single sign-on to applications by issuing a one-time-use ticket to an end user. The ticket can be validated by a client application and is also used to retrieve the identity of the end user for internal use. When configured for CAS authentication, Degree Works checks for a CAS ticket. If a ticket is not present, the request is redirected to the CAS server. After CAS authentication takes place, the request comes back to Degree Works with an authentication ticket. The end user's Degree Works record is retrieved by linking a CAS attribute obtained during ticket validation.

CAS installation and setup

The CAS installation, setup, and basic configurations are outside the scope of this document. As a prerequisite to installing CAS, you need to be familiar with the CAS server, and should be able to modify CAS, xml and jsp configuration files. Information about installing CAS and the supported authentication mechanisms is available at http://www.ja-sig.org/wiki/display/CASUM/Home.

Configuring Degree Works for CAS

CAS supports SSO by issuing a one-use ticket to an end user. The ticket can be validated by a client application and is also used to retrieve the identity of the end user for internal use. Degree Works uses different IDs to internally identify each end user and to log the ID into CAS. For example, an end user usually logs into CAS using a CAS username, while Degree Works internally uses the rad_id. As part of the CAS configuration, you must set up or choose an attribute in your CAS backing data store that will map to the alternate ID field of the Shep user record. This alternate ID is normally populated by the nightly student extract process. Alternatively, the attribute value could map to the Shep access ID of the user.

In Banner, the source of the alternate ID it typically one of the following fields:

1. SPRIDEN_ID - a common choice at Banner sites to map to a Degree Works rad_id. Usually this mapping takes place via the `shp_user_mst.shp_access_id`.

2. UDCID - Degree Works must be configured to extract GOBUMAP_UDC_ID to the `shp_user_mst.shp_alt_id` field.

To configure the Degree Works, all configurations described in the following steps must be completed, including the configuration of the Shepherd Settings.

Step 1 - Configure the Degree Works Banner Extract

The Degree Works Banner Extract is configured to extract each user's `GOBUMAP_UDC_ID` and store it in Degree Works. Select access must be granted to the `GOBUMAP` table in Banner, for the Degree Works user (typically dwmgr). When the Banner extract is run, if a `GOBUMAP` record is found for the individual, the `GOBUMAP_UDC_ID` will be loaded into the `SHP_USER_MST.SHP_ALT_ID`. If a `GOBUMAP` entry is not found, then the individual's `SPRIDEN_ID` will be loaded into the `SHP_USER_MST.SHP_ALT_ID.`

Step 2 - Configure a CAS service for Degree Works

The CAS server needs to know that the Degree Works URL is protected for SSO. This is accomplished by configuring a CAS service for Degree Works.

Use the following steps to configure the CAS service that protects Degree Works.

1. Access your CAS server management page:

```
https://<CAS host>:<CAS port>/<CAS context path>/services/manage.html
```

2. Log in with a valid administrator user name and password (obtained from the CAS administrator).
3. Select the **Add New Service** tab.
4. Enter the following values:

   Name: `Your choice (e.g. "Degree Works appname")`

   Service URL: `https://<Degree Works  application host>:< Degree Works application port>/<Degree Works application context path/`

   Description: `Protecting Degree Works through CAS`

   Status: Select `Enabled` and `SSO Participant`.

   Attributes: Select `UDC_IDENTIFIER`

5. Click **Save Changes**. The CAS server will now allow Degree Works to make sign on requests.
6. Repeat steps 3 through 5 for each application, such the Controller, Composer, Responsive Dashboard and Scribe.

Step 3 - Export the CAS SSL certificate

The CAS server and the Degree Works Web server must run in the HTTPS/SSL mode. To enable Degree Works to make an HTTPS connection to your CAS server, export the SSL certificate used on your CAS server to a file (typically in the JKS keystore format) and make it accessible to the Degree Works Java application container (e.g. Tomcat or Weblogic). Consult the documentation from your certificate provider, for example VeriSign or Thawte, for more information on exporting SSL certificates to a file and converting between various certificate formats.

Step 4 - Configure Degree Works for CAS support

The following Shep settings must be configured to enable CAS support:

core.security.authenticationType - Should be set to “CAS”. It directs Degree Works to redirect unauthenticated requests to the CAS login page. The URL of the login page is configured in the setting described below.

core.security.cas.callbackUrl - This should be set to the URL of your Degree Works application, there should be one for each application with the appropriate specification value. It is used by CAS to callback to Degree Works after the user has logged in.

core.security.cas.idAttribute - This is the attribute configured in the CAS server in step 2 and holds the ID used to link the user to the Degree Works ID. For Banner clients this is typically set to UDC_IDENTIFIER.

core.security.cas.loginUrl - This is the URL of the CAS login page. Degree Works will direct all users to this page when they have not yet authenticated.

core.security.cas.serverUrlPrefix - The CAS URL used by Degree Works to validate the authentication ticket sent with a request.

core.security.cas.useSamlTicketValidator - This is a “true”/”false” value that tells Degree Works whether or not to use a SAML validation protocol with the CAS server when validating the ticket. If set to “true” it will use the SAML protocols for ticket validation. Otherwise, it will use native CAS protocols.

SAML Single Sign On

Degree Works can act as a service provider in a federated authentication environment that uses the SAML 2.0 protocols. In a federated system, a user can log in to one of potentially several different identity providers - services that can authenticate the user. The user is then able to access Degree Works without having to sign in again. A trust relationship is established between the identity provider and Degree Works. A SAML ecosystem is complex and powerful, and a complete explanation is beyond the scope of this document. A good overview can be found at www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf.

Configuring SAML

To enable SAML single sign-on, set `core.security.authenticationType` to “SAML”.

You must create the Service Provider metadata for each Degree Works application covered by SAML authentication. The metadata is used to communicate configuration data to the various identity providers in the system. A description of the metadata is beyond the scope of this document, but there are a few items that are particular to a Degree Works installation. You will be required to provide a Location attribute to the AssertionConsumerService element in the data. This should be in the form “scheme://host:port/somepath/context/saml/SSO/optional/elements”. For example, https://myschool.edu/degreeworks/dashboard/saml/SSO/samlPOST where “somepath” is “degreeworks” and “context” is “dashboard” – the context you have specified in your startup script for the application. A secure schema (e.g. https) is highly recommended here.

The metadata is stored in a SHP setting named `core.security.saml.metadata.xml.serviceProvider`. Since the URL in the metadata is unique to the application, you must configure one of these settings for each application, with a different *spec* value for each setting, corresponding to the application. Refer to the Shp settings documentation for the appropriate spec values.

The metadata for your identity provider is stored in the setting `core.security.saml.metadata.xml.identityProvider`. There are not special Degree Works modifications needed for this setting, and there only needs to be one with a “default” spec.

There are a number settings related to the security key store. The `core.security.saml.keystore.location` specifies the location on the server where the key store is located. It should begin with “file:”. For example: “file:/u01/jdk1.7.0_11/bin/keystore.jks”. The password associated with the key store is stored in `core.security.saml.keystore.password.` The key used to sign the SAML request should be entered in the setting `core.security.saml.keystore.keypair.signingKey`. This is usually the same value as the default key entered in `core.security.saml.keystore.defaultKey`. The `core.security.saml.keystore.keypair.password` contains the password for the key in the `core.security.saml.keystore.keypair.signingKey.`

Once the user has authenticated, an assertion is provided to Degree Works that identifies the student. An attribute in the assertion must contain the ID located on the SHP_ALT_ID or SHP_ACCESS_ID field in the SHP_USER_MST. The SHP_ALT_ID is normally loaded by the extracts from the student system, but may be maintained in Controller. The name of attribute can be anything, and is configured in the setting `core.security.saml.idAttribute`.

External Access Manager

An External Access Manager intercepts all requests to Degree Works, making sure they are authenticated. Under this scenario, Degree Works has no responsibility for ensuring that the requests have been authenticated. Examples of an external access manager include Oracle Access Manager and Shibboleth. When configuring the external authentication manager, you must ensure that all Degree Works endpoints are covered.

The external access manager asserts the user’s identity to Degree Works after authentication or SSO has taken place. The configuration allows the assertion to have a configurable name and to be delivered to Degree Works via an HTTP cookie or header.

Shepherd Settings configuration

`core.security.externalAccessManager.enable`

Enable external access manager. Again, it is important to take care never to leave this flag enabled (true) unless an external access manager is in place, configured and operational. If enabled, other security mechanisms (e.g. SHP, CAS) are disabled. In particular, the Shep setting `core.security.authenticationType` is ignored.

`core.security.externalAccessManager.assertionName`

The name of the external access assertion token. This name can be any value of your choosing except for "ASSERT_VALUE".

`core.security.externalAccessManager.assertionIsCookie`

If true, expect to find assertion value in a cookie. The name of the cookie should match the value in the setting `core.security.externalAccessManager.assertionName.` If false, it is expected to be in a header.

Self Service Banner Single Sign-On

The preferred method for single sign-on integration of Self-Service Banner with Degree Works is to use a standard central authentication service such as CAS or SAML. See the *Degree Works Banner Considerations Technical Guide* for more information on this feature.

Luminis Single Sign-On

The recommended method for single sign-on integration of Luminis with Degree Works is CAS. See the *Degree Works Banner Considerations Technical Guide* for more information on this feature.

Multiple Authentication Entry Points

When the authenticationType is set to CAS or SAML, the basic SHP authentication is automatically disabled. However, you may decide, for example, that you want the dashboard to use CAS or SAML authentication but you may want Scribe or Controller to use basic SHP security and the built-in login pages. To do that you can use Controller to create settings specific to each application.

For **Scribe**, create two new settings with these values:

Key	**Value**	**Specification**
core.security.authenticationType	SHP	scribe
core.security.shp.authentication.enabled	true	scribe

For the **Controller**, create two new settings with these values:

Key	**Value**	**Specification**
core.security.authenticationType	SHP	controller
core.security.shp.authentication.enabled	true	controller

Be sure to restart each application in Tomcat or WebLogic after making these changes in Controller. With these settings in place you would then only need to have a link to the dashboard in your portal and users of Scribe and Controller would use the built-in login.jsp pages.

Authentication is Persistent for a session

Once Authenticated, the user’s browser receives stateless tokens conforming to the JSON Web Token (JWT) standard. One is the “refresh” token which expires at the user’s maximum timeout and also identifies the increment timeout. The other token is the “access” token which expires at the user’s increment timeout and is recreated with each API call from the browser as long as the maximum timeout isn’t past. When a user logs out, token is cleared from the user’s browser and added to the TOKEN_REVOCATION_MST table. That table tracks revoked tokens to make sure they cannot be authenticated again.

This time limit has both an incremental and a maximum timeout. If the user continues to be active with their session, the expiration is extended by a configured increment. This is known as the timeout increment amount. There is also a maximum period, known as the timeout maximum, which is measured from the time the user signs on. Either or both of these values can be set to an unlimited time period.

The primary source for these values is the SHPCFG data. If the “TIMEINC” or “TIMEMAX” verbs are included in a rule the user qualifies for, then the timeout periods are set from these values. Otherwise, the values can come from the user’s SHP user record, or one of the groups to which the user belongs, or from the SHP settings `core.security.passport.timeoutIncrementDefault` and `core.security.passport.timeoutMaximumDefault`. Which one is used depends on the setting `core.security.passport.timeoutPrecedence` and `core.security.passport.timeoutPreference`. Refer to the Shp Settings technical documentation for more information about these settings and how they affect the session timeout values.

Access Control (Authorization)

Degree Works functionality is defined in terms of “Services”. Every service has a lock, with a corresponding key or keys. Keys can be organized and assigned into groups. The user’s keyring is a set of keys which ultimately controls what services are allowed. The user’s keyring is built during authentication. The user’s keyring is put together based on attributes of the user, and keys are assigned through rules in SHPCFG or through Controller. Keys can be added or removed (hence services can be allowed or denied). Keys can be referenced via a group, or individually.

Assigning keys with SHPCFG

The SHPCFG data reside in a SHP setting, `core.security.rules.shpcfg,` and can be used to globally assign keys and timeouts at logon.

SHPCFG contains a series of if-statements which assign keys to a user’s keyring based on the user’s assigned role and other user attributes.

Keys/Keyrings

Each User has a Keyring with one or more keys. These Keyrings are stored in SHPDB and give access to services. When users are authenticated at the time of logon, they acquire keys that are both explicitly and implicitly assigned.

Keys are needed to access “locked” Services.

Keys are defined in UCX_SHP078, so you may view the available Keys using Controller or review “List of Services and associated Keys” in the *Services* section of this document.

Groups are stored in the SHP_GROUP_MST and can be reviewed via Controller, or in the “List of standard Groups and Keys” in the *Groups* section of this document.

Explicit Assignment

Explicit keys are those assigned to an individual by id either through Controller or in SHPCFG. This method is inefficient for assignments in bulk, but very effective for granular, specific control.

Implicit Assignment

Implicit assignment of keys is accomplished through the use of authorization rules. It is very efficient for bulk assignments. This is accomplished by edits to SHPCFG, which contains the rules.

Maintaining SHPCFG

SHPCFG is maintained in Controller in the `core.security.rules.shpcfg` setting.

SHPCFG is a collection of if-statements which must adhere to specific rules and syntax. The if-statements in SHPCFG are in the following format:

```
if (expression) then
  commands
```

An expression consists of a Code and Value, for example:

```
if (DGWUSERCLASS = "REG") then
  addgroup = SRNREG    # Standard keys for the registrar's office users
  addkey   = RSAUDIT   # Web service for audit/articulation
```

Users who have been authenticated are assigned a User Class (DGWUSERCLASS) and are granted keys based on assignment of groups or specific keys to their User Class. In the above example, users with User Class of REG will have all keys from the SRNREG group plus the RSAUDIT key added to his keyring.

There are two types of **expression codes**; some are available to all users while others are only available for use with students. Valid expression codes for use with all users are:

DGWUSERCLASS	The USERCLASS assigned during authentication, defined in UCX_AUD012
DGWSHPACCID	The user's SHP_ACCESS_ID can be used to explicitly add or remove keys or groups
EVERYONE	Add or remove keys or groups to all users. This code is unique in that it does not require a value, e.g. **if (EVERYONE) then**

The following **expression codes** can be coded for students only. During the student extract process, if configured to do so, the following values will be stored in the SHP_USER_ATTRIB table which makes this data available to SHPCFG.

ACTIVETERM	RAD_STUDENT_MST.RAD_TERM
CATALOGYEAR	RAD_GOAL_DTL.RAD_CATALOG_YR
COLLEGE	RAD_GOALDATA_DTL.RAD_GOAL_VALUE where RAD_GOAL_CODE = COLLEGE
DEGREE	RAD_GOAL_DTL.RAD_DEGREE_CODE
DEGREESOURCE	RAD_GOAL_DTL.RAD_DEGREE_SRC
MAJOR	RAD_GOALDATA_DTL.RAD_GOAL_VALUE where RAD_GOAL_CODE = MAJOR
PROGRAM	RAD_GOALDATA_DTL.RAD_GOAL_VALUE where RAD_GOAL_CODE = PROGRAM
SCHOOL	RAD_GOAL_DTL.RAD_SCHOOL
STUDENTLEVEL	RAD_GOAL_DTL.RAD_STU_LEVEL
Custom codes	Custom codes, generated by UCX_BAN080 (Banner) or custom.client.properties (Colleague) can also be used as SHPCFG expressions. For more information on setting up custom codes and adding them to SHP_USER_ATTRIB, see the Considerations Guide for your Student Information System (SIS).

Valid **commands** in SHPCFG are as follows. Note that they are not case-sensitive:

addGroup	Adds a group to the user's keyring
remGroup	Removes a group from the user's keyring
addKey	Adds a specific key to the User's keyring
remKey	Removes a specific key from the User's keyring
deny	Denies access
timeInc	Sets a timeout increment for the User
timeMax	Sets a maximum login time for the User

Expression operators that can be used are:

=	Equal
<>	Not Equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Expressions can be combined with the conditional operators AND and OR. To ensure the results you expect, follow the best practice of using parentheses when combining expressions:

```
If (CODE1 = "VALUEA" or CODE2 = "VALUEB") then
If (CODE1 = "VALUEA" and CODE2 = "VALUEB") then
If (CODE1 = "VALUEA" and (CODE2 = "VALUEB" or CODE3 = "VALUEC")) then
```

Note that the use of ELSE is not supported, for example the following is **not valid**:

```
If (DGWUSERCLASS <> "STU") then
  addkey   = SUPPORT   # Support
Else                   # ELSE is not supported!
  addGroup = SRNSTU    # Standard keys for students
```

Instead, code the above logic as follows in separate rules:

```
If (DGWUSERCLASS <> "STU") then
  addkey   = SUPPORT   # Support

If (DGWUSERCLASS = "STU") then
  addGroup = SRNSTU    # Standard keys for students
```

In the next example, all students will be granted all keys in the SRNSTU group, but only students whose RAD_SCHOOL value = ‘UG’ will be granted the keys in the SEPSTUED group, except for the SEPPDELL key:

```
if (DGWUSERCLASS = "STU") then
addGroup = SRNSTU   # Standard keys for students

if (DGWUSERCLASS = "STU" and SCHOOL = "UG") then
addGroup = SEPSTUED # Planner Edit/Create
remkey   = SEPPDELL # Delete Plans
```

In the following example, the advisors are being assigned the SRNADV group. Advisors are also been given the SDSTUANY key to allow them to search on any student – we take away the SDSTUMY key so that advisors are not tied to a specific list of advisees.

If your advisors are tied to certain advisees then the SRNADV group will give you what you need. If you are bridging students with different TERM values and wish to see all students assigned to any advisor regardless of TERM value, set the UCX_CFG020 WEBPARAMS active term field to blanks to ignore the TERM field as part of the search criteria.

```
if (EVERYONE) then
  TIMEINC = 0055         #55 Minutes of idle time
  TIMEMAX = 0800         #Max 8 hrs before relogon needed
  remKey = SDPLANER,     #Remove old planner keys
  remKey = SDPLNMOD
  remKey = SDPLNVEW

#------------------------------------------------------------------------------
#-- DegreeWorks keys for students
#------------------------------------------------------------------------------
if (DGWUSERCLASS = "STU") then
  addGroup = SRNSTU
  addKey   = SD2SEPMOD # allow student to modify their own plan

#------------------------------------------------------------------------------
#-- DegreeWorks keys for applicants
#------------------------------------------------------------------------------
if (DGWUSERCLASS = "APP") then
  addGroup = SRNAPP

#------------------------------------------------------------------------------
#-- DegreeWorks keys for advisors
#------------------------------------------------------------------------------
if (DGWUSERCLASS = "ADV") then
  addGroup = SRNADV
  remKey   = SDSTUMY  # load my advisees only
  addKey   = SDSTUANY # allow search on all students

#------------------------------------------------------------------------------
#-- DegreeWorks keys for administrators
#------------------------------------------------------------------------------
if (DGWUSERCLASS = "REG") then
  addKey =   SDSOCMIL
  addGroup = SRNREG
```

The above example also illustrates use of the EVERYONE expression code. Here all users are assigned timeInc of 0055 and timeMax 0800. These time settings are in the format of hhmm, where timeInc is set to 55 minutes and timeMax is set to 8 hours. For more information on timeInc and timeMax, see the *User Session Timeout* section in this document.

While modifying SHPCFG, feel free to add comments which are preceded with a #. Comments can be added at the beginning of a line or after the assignment of a key or group, as illustrated above.

Running **shpparse** first is good as it will report any errors. Running daprestart (manually or via Transit) will also run shpparse but you will not see the parse errors. The shpparse script will pull SHPCFG from the SHP_SETTINGS_MST before attempting to parse it.

After modifying SHPCFG and doing a daprestart you need to restart the java applications in tomcat or weblogic in order for them to pick up the changes in the SHP_SETTINGS_MST.

Services

Each component of business functionality is a service. Services may be broad (an entire web page or more) or narrow (a button which does something useful). Services are locked and keys are needed to access them.

List of Services and associated Keys

TITLE	KEYS	APPLICATION
All services beginning with "PT" are standard PC Transit services		
Reports (DAP16, DAP22)	PTSDGWRE or PTSHPDGW	PC Transit
RAD Bridge (RAD11)	PTSRADPR or PTSHPRAD	PC Transit -- RAD only
Transit ADMIN jobs	PTSADMIN	PC Transit
Transit SCRxx jobs	PTSSCRIB	PC Transit
Transit Banner Bridge (RAD30)	PTSBANPR	PC Transit
Transit OPS Bridge (RAD20)	PTSOPSPR	PC Transit
Transit Colleague Bridge (CLG30)	PTSCLGPR	PC Transit
Transit AUDIT jobs	PTSAUDIT	PC Transit
All services beginning with "RS" are keys to access REST web services in degreeworks-services deployment		
Retreive audit/articulation	RSAUDIT	Web Service
Retrieve classes by student ID	RSCLASS	Web Service
Retrieve Mapping through a Web Service	RSMAPPNG	Web Service
Retrieve Plans through a Web Service	RSPLAN	Web Service
Retrieve SHP Settings	RSSETTNG	Web Service
Retrieve validation (UCX) tables	RSVALID	Web Service
Role for CourseInfo web services	RSCRSINF	Web Service
Run what-if audit	RSWHATIF	Web Service
Allows application access to Degree Works APIs	NOREFER	Web Service
All of these services are Dashboard and Responsive Dashboard services		
Financial Aid Audits tab	SDAIDAUD	Web
Review Financial Aid Audits	SDAIDREV	Web
Run Financial Aid Audits	SDAIDRUN	Web
Financial Aid Audit History	SDAIDHIS	Web
Delete Financial Aid Audits	SDAIDDEL	Web
Athletic Eligibility Audits tab	SDATHAUD	Web
Athletic Eligibility Audit History	SDATHHIS	Web
Review Athletic Eligibility Audits	SDATHREV	Web
Run Athletic Eligibility Audits	SDATHRUN	Web
Delete Athletic Eligibility Audits	SDATHDEL	Web
Admin tab	SDADMIN	Web
Audits – Worksheets tab	SDAUDIT	Web
Save as PDF option	SDAUDPDF	Web
Review Audit	SDAUDREV	Web
Run Audit	SDAUDRUN	Web
Delete Academic Audits	SDAUDDEL	Web
Department user	SDDEPART	Web
Exp Mgmt Exceptions Search	SDEMEXSR	Web

TITLE	KEYS	APPLICATION
Delete applied petitions	SDEMPEAD	Web
Except Mgmt Petitions Applied	SDEMPEAL	Web
Except Mgmt Petitions Approved	SDEMPEAV	Web
Fix petition status	SDEMPEFX	Web
Delete rejected petitions	SDEMPERD	Web
Except Mgmt Petitions Rejected	SDEMPERJ	Web
Except Mgmt Petitions Waiting	SDEMPEWA	Web
Exceptions	SDEXCEPT	Web
Add Exception	SDEXPADD	Web
Delete Exception	SDEXPDEL	Web
External Links	EXTLINKS	Responsive Dashboard
Allows visibility of encrypted Shep settings	SHENCRPT	Java Applications
Access to turn on and off debugging	DEBUG	Java Applications
Exeptions – Also Allow	EXPALLOW	Web
Exceptions – Apply Here	EXPAPPLY	Web
Exeptions – Change the Limit/ Remove Course	EXPCHANG	Web
Exceptions – Force Complete	EXPFORCE	Web
Exceptions – Substitute	EXPSUBST	Web
Exceptions – view details on worksheet	EXPVWDTL	Web
Exception Management Access	SDEXPMGT	Web
Student Search	SDFIND	Web
Student Search by ID	SDFINDID	Web
GPA Advice Calculator	SDGPAADV	Web
GPA Calculator	SDGPACLC	Web
GPA Graduation Calculator	SDGPAGRD	Web
GPA Term Calculator	SDGPATRM	Web
Audit History	SDHIST	Web
Look-ahead audit	SDLOKAHD	Web
Notes	SDNOTES	Web
Note Add	SDNTEADD	Web
Add free-form notes	SDNTECHG	Web
Delete Note	SDNTEDEL	Web
Modify Note	SDNTEMOD	Web
View Note	SDNTEVUE	Web
Run New Audit after saving note	SDNTERUN	Web
Petition Add	SDPETADD	Web
View status of all petitions	SDPETALS	Web
Petitions delete	SDPETDEL	Web
Petitions Modify	SDPETMOD	Web
View status of my petitions	SDPETMYS	Web
Petition View	SDPETVEW	Web
Show Refresh date/time (for Banner, also controls the refresh button)	SDREFRES	Web
Refresh Button – Colleague only	SDREFBTN	Web
Registrar/Any Student – for searching	SDSTUANY	Web
Student Services	SDSTUME	Web
My Advisees – for searching	SDSTUMY	Web
WEB30 Worksheet	SDWEB30	Web
WEB31 Worksheet	SDWEB31	Web
WEB32 Worksheet	SDWEB32	Web
WEB33 Worksheet	SDWEB33	Web
WEB34 Worksheet	SDWEB34	Web
WEB35 Worksheet	SDWEB35	Web
WEB36 Worksheet	SDWEB36	Web
WEB37 Worksheet	SDWEB37	Web
Financial Aid Report	SDWEB50	Web

TITLE	KEYS	APPLICATION
Aid and Academic Report	SDWEB51	Web
Athletic Eligibility Report	SDWEB55	Web
Athletic and Academic Report	SDWEB56	Web
What-if Audit	SDWHATIF	Web
What-If History tab	SDWIFHIS	Web
What-If Delete button on History tab	SDWIFDEL	Web
DegreeWorks Access	SDWORKS	Web
Diagnostics Report	SDXML30	Web
Class History report	SDXML31	Web
Student Data report	SDXML32	Web
Class Summary Report	SDXML33	Web
Freeze Audits	AUDFREEZ	Web
Audit Description	AUDDESCR	Web
What-if Freeze Audits	WIFFREEZ	Web
What-if Audit Description	WIFDESCR	Web
Any Student ID is allowed	ANYSTUID	Web
Advisor’s Advisees only are allowed	ADVISIDS	Web
Access to Support Features	SUPPORT	Web
All services beginning with "SEP" are standard Student Planner services		
Create New Plan	SEPPADD	Planner
Generate audits from within planner	SEPPAUD	Planner
Permission to "auto approve" a plan	SEPPAUTO	Planner
Create Block on plan	SEPPBLCK	Planner
Delete plans	SEPPDEL	Planner
Delete plans, limited	SEPPDELL	Planner
Planner tab	SEPPLAN	Planner
Lock a plan	SEPPLOCK	Planner
Edit a plan	SEPPMOD	Planner
Add plan term notes	SEPPNMAD	Planner
Delete plan term notes created by the current user	SEPPNMDL	Planner
Edit plan term notes created by the current user	SEPPNMED	Planner
Add plan requirement notes	SEPPNRAD	Planner
Delete plan requirement notes created by the current user	SEPPNRDL	Planner
Edit plan requirement notes created by the current user	SEPPNRED	Planner
Add plan notes	SEPPNTAD	Planner
Delete plan notes created by the current user	SEPPNTDL	Planner
Edit plan notes created by the current user	SEPPNTED	Planner
Add/Edit plan requirements	SEPPRQAD	Planner
Delete plan requirements	SEPPRQDL	Planner
Edit plan requirements	SEPPRQED	Planner
Override plan pre- and co-requisite checking	SEPPRQTO	Planner
Add/Edit plan terms	SEPPTADD	Planner
Delete plan terms	SEPPTDEL	Planner
Edit plan terms	SEPPTMOD	Planner
Create plan from a template	SEPPTEMP	Planner
Show the What-If button in Plans	SEPPWIF	Planner
Create New Template	SEPTADD	Planner
Delete templates	SEPTDEL	Planner
Edit templates	SEPTEDIT	Planner
Template Management functionality	SEPTMGMT	Planner
Add template term notes	SEPTNMAD	Planner

TITLE	KEYS	APPLICATION
Delete template term notes	SEPTNMDL	Planner
Edit template term notes	SEPTNMED	Planner
Add template requirement notes	SEPTNRAD	Planner
Delete template requirement notes	SEPTNRDL	Planner
Edit template requirement notes	SEPTNRED	Planner
Add/Edit template requirements	SEPTRQAD	Planner
Delete template requirements	SEPTRQDL	Planner
Edit template requirements	SEPTRQED	Planner
Add template notes	SEPTNTAD	Planner
Delete template notes	SEPTNTDL	Planner
Edit template notes	SEPTNTED	Planner
Change template term scheme	SEPTTRMS	Planner
Edit the critical indicator on plan and template requirements	SEPCRIT	Planner
Access to internal notes on plans	SEPINOTE	Planner
Select course choice requirement options on plans.	SEPPSEL	Planner
Edit plan notes created by other users	SEPPNTOW	Planner
Delete plan notes created by other users	SEPPNTGD	Planner
Edit plan notes that have been copied from a template	SEPPNTET	Planner
Delete plan notes that have been copied from a template	SEPPNTDT	Planner
Edit plan CHOICE requirement pointer	SEPPPTR	Planner
View the plan in Edit Mode	SEPPEDIT	Planner
View the plan in Calendar Mode	SEPVCAL	Planner
View the plan in Audit Mode	SEPVAUD	Planner
View the plan in Notes Mode	SEPVNOTE	Planner
Modify scope field in the Edit view	SEPSCOPE	Planner
All services below are for Transfer Equivalency		
TreqAdmin audit and API articulation and audits	SDTREQER	Transfer Equivalency
Transfer Equivalency Admin - access	SRNTREQ	Transfer Equivalency
All services below are for Transfer Equivalency Self-Service application		
Transfer Equivalency Self-Service access.	DWTESELF	Transfer Equivalency Self-Service
Transfer Equivalency Self-Service admin	DWTEADMN	Transfer Equivalency Self-Service
All services below enable functionality in the Transfer Finder application		
Transfer Finder Categories from audit	TFCATGRY	Transfer Finder
Transfer Finder Tab	TFFINDER	Transfer Finder
Transfer Finder Don’t Display Acknowledge Message	TFNOACK	Transfer Finder
Transfer Finder Transfer Audit	TFTRAUDT	Transfer Finder
All services below enable functionality in the Scribe application		
Key to access Scribe web application	SCRIBE	Scribe
Key to access parse API	SCRPARSE	Scribe
Access to modify all block types	SCRBLALL	Scribe
Access to modify ATHLETE blocks	SCRBLATH	Scribe
Access to modify AWARD blocks	SCRBLAWR	Scribe
Access to modify COLLEGE blocks	SCRBLCOL	Scribe
Access to modify CONC blocks	SCRBLCON	Scribe

TITLE	KEYS	APPLICATION
Access to modify DEGREE blocks	SCRBLDEG	Scribe
Access to modify ID blocks	SCRBLID	Scribe
Access to modify LIBL blocks	SCRBLLIB	Scribe
Access to modify MAJOR blocks	SCRBLMAJ	Scribe
Access to modify MINOR blocks	SCRBLMIN	Scribe
Access to modify OTHER blocks	SCRBLOTH	Scribe
Access to modify PROGRAM blocks	SCRBLPRG	Scribe
Access to modify REQUISITE blocks	SCRBLREQ	Scribe
Access to modify SCHOOL blocks	SCRBLSCH	Scribe
Access to modify SPEC blocks	SCRBLSPC	Scribe
All services below enable functionality in the Composer application		
Key to access Composer application	COMPOSER	Composer
All services below enable functionality in the Controller application		
Application Access	CONTROL	Controller
Access to Users tab	CTLUSERS	Controller
Access to Add Users	CTLUSRAD	Controller
Access to Modify Users	CTLUSRMD	Controller
Access to Delete Users	CTLUSRDL	Controller
Access to Groups tab	CTLGROUP	Controller
Access to Add Groups	CTLGRPAD	Controller
Access to Modify Groups	CTLGRPMD	Controller
Access to Delete Groups	CTLGRPDL	Controller
Access to Configuration tab	CTLCONF	Controller
Access to Add Shepherd Settings	CTLSETAD	Controller
Access to Modify Shepherd Settings	CTLSETMD	Controller
Access to Delete Shepherd Settings	CTLSETDL	Controller
Access to Add UCX Entries	CTLUCXAD	Controller
Access to Modify UCX Entries	CTLUCXMD	Controller
Access to Delete UCX Entries	CTLUCXDL	Controller
Access to UCX Bulk Operations	CTLUCXBK	Controller
All services below enable functionality in the Transit application		
Transit - Application Access	TRANSIT	Transit
Transit - Run All Jobs	TRANALL	Transit
Transit - Access to Run Jobs tab	TRANRUN	Transit
Transit - Access to SQL select criteria	TRANSQL	Transit
Transit - Access to delete jobs	TRANDEL	Transit
Transit - Access to Upload Artifacts	TRANART	Transit
Access to run ADMIN jobs and scripts	TRADMIN	Transit
Access to run AUD01	TRAUD01	Transit
Access to run AUD02	TRAUD02	Transit
Access to run BAN62 (Banner clients)	TRBAN62	Transit
Access to run the Student Extract (Colleague clients)	TRCLG30	Transit
Access to run the Advisor Extract (Colleague clients)	TRCLG31	Transit
Access to run the Staff Extract (Colleague clients)	TRCLG32	Transit
Access to run the Colleague Course Extract	TRCLG33	Transit
Access to run the Colleague Validation Extract (UCX)	TRCLG34	Transit
Access to run the Colleague Transfer School Extract (ETS)	TRCLG35	Transit
Access to run the Colleague Equivalency	TRCLG36	Transit

TITLE	KEYS	APPLICATION
Extract		
Access to run the Colleague Transfer Equivalency Extract (Mappings)	TRCLG37	Transit
Access to load the *.client.properties files (Colleague clients)	TRCLG38	Transit
Access to run the DAP16 processor	TRDAP16	Transit
Access to run DAP21	TRDAP21	Transit
Access to run DAP22	TRDAP22	Transit
Access to run DAP27	TRDAP27	Transit
Access to run DAP28	TRDAP28	Transit
Access to run DAP54	TRDAP54	Transit
Access to run DAP58	TRDAP58	Transit
Access to run DAP59	TRDAP59	Transit
Access to run the Radbridge Processor (RAD clients only)	TRRAD11	Transit
Access to run the Student Extract (Banner clients)	TRRAD30	Transit
Access to run the Advisor Extract (Banner clients)	TRRAD31	Transit
Access to run the Applicant Extract (Banner clients)	TRRAD32	Transit
Access to run the Staff Extract (Banner clients)	TRRAD33	Transit
Access to run the Banner Course Extract	TRRAD34	Transit
Access to run the Banner Curriculum Rules Extract	TRRAD35	Transit
Access to run the Banner Validation Extract (UCX)	TRRAD36	Transit
Access to run the Banner Transfer School Extract (ETS)	TRRAD37	Transit
Access to run the Banner Equivalencies Extract	TRRAD38	Transit
Access to run the Banner Transfer Equivalency Extract (Mappings)	TRRAD39	Transit
Access to run SCR02	TRSCR02	Transit
Access to run SCR05	TRSCR05	Transit
Access to run SCR06	TRSCR06	Transit
Access to run SCR07	TRSCR07	Transit
Access to run SCR08	TRSCR08	Transit
Access to run SCR09	TRSCR09	Transit
Access to run SCR10	TRSCR10	Transit
Access to run SCR11	TRSCR11	Transit
Access to run SCR91 (test for Banner Prerequisite)	TRSCR91	Transit
Access to run SCR92 (test for Banner Prerequisite)	TRSCR92	Transit
Access to run SCR93	TRSCR93	Transit
Access to run SCR94	TRSCR94	Transit
Access to run SCR95	TRSCR95	Transit
Access to run UCX01	TRUCX01	Transit

Groups

A User Class will typically have a Group of Keys assigned. These groups are stored in SHPDB and can be viewed and modified using Controller. A user will inherit the group keys from their user-class, which will be combined with other keys they may acquire from SHPCFG, or those assigned through Controller.

The Keys assigned to a Group can be modified via Controller. See the *Controller Administrative Guide* for more information.

List of standard Groups and associated Keys

GROUP	KEYS
CONTROL	CONTROL, CTLUSERS, CTLUSRAD, CTLUSRMD, CTLUSRDL, CTLGROUP, CTLGRPAD, CTLGRPMD, CTLGRPDL, CTLCONF, CTLSETAD, CTLSETMD, CTLSETDL, CTLUCXAD, CTLUCXMD, CTLUCXDL, CTLUCXBK
SCRIBAEA	SCRIBE, SCRPARSE, SCRBLATH
SCRIBAID	SCRIBE, SCRPARSE, SCRBLAWR
SCRIBREG	SCRIBE, SCRPARSE, SCRBLALL
SEPADV	RSCRSINF, RSPLAN, RSSETTNG, SEPCRIT, SEPINOTE, SEPPADD, SEPPAUD, SEPPBLCK, SEPPDEL, SEPPEDIT, SEPPLOCK, SEPPMOD, SEPPNMAD, SEPPNMDL, SEPPNMED, SEPPNRAD, SEPPNRDL, SEPPNRED, SEPPNTAD, SEPPNTDL, SEPPNTED, SEPPPTR, SEPPRQAD, SEPPRQDL, SEPPRQED, SEPPSEL, SEPPTADD, SEPPTDEL, SEPPTEMP, SEPPTMOD, SEPPWIF, SEPVAUD, SEPVCAL, SEPPLAN, SEPSCOPE, SEPVNOTE
SEPREG	RSCRSINF, RSPLAN, RSSETTNG, SEPCRIT, SEPINOTE, SEPPADD, SEPPAUD, SEPPAUTO, SEPPBLCK, SEPPDEL, SEPPEDIT, SEPPLOCK, SEPPMOD, SEPPNMAD, SEPPNMDL, SEPPNMED, SEPPNRAD, SEPPNRDL, SEPPNRED, SEPPNTAD, SEPPNTDL, SEPPNTDT, SEPPNTED, SEPPNTET, SEPPNTGD, SEPPNTOW, SEPPPTR, SEPPRQAD, SEPPRQDL, SEPPRQED, SEPPRQTO, SEPPSEL, SEPPTADD, SEPPTDEL, SEPPTEMP, SEPPTMOD, SEPPWIF, SEPSCOPE, SEPTADD, SEPTDEL, SEPTEDIT, SEPTMGMT, SEPTNMAD, SEPTNMDL, SEPTNMED, SEPTNRAD, SEPTNRDL, SEPTNRED, SEPTNTAD, SEPTNTDL, SEPTNTED, SEPTRQAD, SEPTRQDL, SEPTRQED, SEPTTRMS, SEPVCAL, SEPPLAN, SEPVAUD, SEPVNOTE
SEPSTUED	RSCRSINF, RSPLAN, RSSETTNG, SEPPADD, SEPPAUD, SEPPDELL, SEPPMOD, SEPPNMAD, SEPPNMDL, SEPPNMED, SEPPNRAD, SEPPNRDL, SEPPNRED, SEPPNTAD, SEPPNTDL, SEPPNTED, SEPPRQAD, SEPPRQDL, SEPPRQED, SEPPTADD, SEPPTDEL, SEPPTEMP, SEPPTMOD, SEPPWIF, SEPVCAL, SEPVNOTE, SEPPEDIT, SEPPLAN, SEPVAUD
SEPSTUVW	RSCRSINF, RSPLAN, RSSETTNG, SEPPAUD, SEPVCAL, SEPVNOTE, SEPPLAN, SEPPSEL, SEPVAUD
SRNADV	ADVISIDS, EXPALLOW, EXPAPPLY, EXPCHANG, EXPCHANGE, EXPFORCE, SDAUDPDF, SDAUDRUN, SDEXCEPT, SDEXPADD, SDEXPDEL, SDFIND, SDGPAADV, SDGPACLC, SDGPAGRD, SDGPATRM, SDLOKAHD, SDNOTES, SDNTEADD, SDNTECHG, SDNTEDEL, SDNTEMOD, SDNTERUN, SDNTEVUE, SDPLNAUD, SDPLNDEL, SDSEP, SDSEPMOD, SDSTUANY, SDWEB31, SDWEB36, SDWHATIF, SDWIFDEL, SDWIFHIS, SDWORKS, SDXML31, WIFDESCR, WIFFREEZ, SDAUDREV, EXTLINKS
SRNADVX	ADVISIDS, EXPALLOW, EXPAPPLY, EXPCHANG, EXPCHANGE, EXPFORCE, SDAUDPDF, SDAUDRUN, SDFIND, SDGPAADV, SDGPACLC, SDGPAGRD, SDGPATRM, SDLOKAHD, SDNOTES, SDNTEADD, SDNTECHG, SDNTEDEL, SDNTEMOD, SDNTERUN, SDNTEVUE, SDPETADD, SDPETMOD, SDPETMYS, SDPETVEW, SDPLNAUD, SDPLNDEL, SDSEP, SDSEPMOD, SDSTUANY, SDWEB31 , SDWEB36, SDWHATIF, SDWIFDEL, SDWIFHIS, SDWORKS, SDXML31, SEPETDEL, WIFDESCR, WIFFREEZ, SDAUDREV, EXTLINKS
SRNAID	ANYSTUID, AUDDESCR, AUDFREEZ, AWARD, SDAIDDEL, SDAIDHIS, SDAIDREV, SDAIDRUN, SDAUDPDF, SDFIND, SDSTUANY, SDWEB50, SDWEB51, SDWEB52, SDWORKS,SDAIDAUD, EXTLINKS
SRNAPP	SDAUDPDF, SDLOKAHD, SDSTUME, SDWEB31, SDWHATIF, SDWORKS, SDXML31, SDAUDREV, EXTLINKS

SRNATHL	ANYSTUID, ATHLETE, AUDDESCR, AUDFREEZ, SDATHDEL, SDATHHIS, SDATHREV, SDATHRUN, SDAUDPDF, SDFIND, SDSTUANY, SDWEB55, SDWEB56, SDWORKS, SDXML30, SDXML33, SDATHAUD, EXTLINKS
SRNREG	ANYBLOCK, ANYSTUID, AUDDESCR, AUDFREEZ, EXPALLOW, EXPAPPLY, EXPCHANG, EXPCHANGE, EXPFORCE, PSDGWSHP, PTSADMIN, PTSAUDIT, PTSDGWRE, PTSRADPR, PTSSCRIB, SCBULK, SDADMIN, SDATHAUD, SDATHDEL, SDATHHIS, SDATHREV, SDATHRUN, SDAUDDEL, SDAUDPDF, SDAUDREV, SDAUDRUN, SDEMEXSR, SDEMPEAD, SDEMPEAL, SDEMPEAV, SDEMPEFX, SDEMPERD, SDEMPERJ, SDEMPEWA, SDEXCEPT, SDEXPADD, SDEXPDEL, SDEXPMGT, SDFIND, SDGPAADV, SDGPACLC, SDGPAGRD, SDGPATRM, SDHIST, SDLOKAHD, SDNOTES, SDNTEADD, SDNTECHG, SDNTEDEL, SDNTEMOD, SDNTERUN, SDNTEVUE, SDPLNAUD, SDPLNDEL, SDSEP, SDSEPAPP, SDSEPMOD, SDSTUANY, SDTMP, SDWEB30, SDWEB31, SDWEB32, SDWEB34, SDWEB35, SDWEB36, SDWEB37, SDWEB55, SDWEB56, SDWHATIF, SDWIFDEL, SDWIFHIS, SDWORKS, SDXML30, SDXML31, SDXML32, SDXML33, SUPPORT, WIFDESCR, WIFFREEZ,PSDGWSTD, SDWEB33, EXTLINKS
SRNSTU	SDAUDPDF, SDGPAADV, SDGPACLC, SDGPAGRD, SDGPATRM, SDLOKAHD, SDPLNVEW, SDSEP, SDSTUME, SDWEB31, SDWEB36, SDWHATIF, SDWORKS, SDXML31, SDAUDREV, EXTLINKS
TFADV	TFCATGRY, TFNOACK, TFTRAUDT, TFFINDER
TFREG	TFCATGRY, TFNOACK, TFTRAUDT, TFFINDER
TFSTU	TFCATGRY, TFTRAUDT, TFFINDER
TRANBAN	TRBAN62, TRRAD30, TRRAD31, TRRAD32, TRRAD33, TRRAD34, TRRAD35, TRRAD36, TRRAD37, TRRAD38, TRRAD39, TRSCR91, TRSCR92, TRSCR93, TRSCR94, TRSCR95
TRANCLG	TRCLG30, TRCLG31, TRCLG33, TRCLG34, TRCLG36, TRCLG37, TRCLG38, TRCLG39, TRCLG40
TRANREG	TRANSIT, TRANRUN, TRANSQL, TRANDEL, TRADMIN, TRAUD01, TRAUD02, TRDAP16, TRDAP21, TRDAP22, TRDAP27, TRDAP28, TRDAP54, TRDAP58, TRDAP59, TRRAD11, TRSCR02, TRSCR05, TRSCR06, TRSCR07, TRSCR08, TRSCR09, TRSCR10, TRSCR11, TRUCX01

Users

Authenticated users may request services based on their user-class assignment. User-class assignment associates the user with similar users for the purpose of controlling read-write access to notes and access to menu options in Degree Works. User-class is sent to Degree Works on the SHPU record when each user is loaded via the bridge. For Banner clients this is handled by the Banner extracts and their associated configuration tables. See the *Banner Considerations Guide* for additional information.

User Class

The valid user class codes are stored in UCX_AUD012. The user class determines which shp_group (collection of keys) is granted to the user, thereby determining which services the user can access.

The user-class is stored in the SHP_USER_MST in DAPDB. A user can have only one user class.

The following set of groups is normally assigned in SHPCFG based on the user-class:

User Class	Description	Group (shp_group_mst)
ADV	Advisor	SRNADV, SEPADV, TFADV

ADVX	Advisor without exceptions	SRNADVX, SEPADV
AID	Financial Aid Office	SRNAID, SCRIBAID
APP	Applicant	SRNAPP, SEPSTUVW or SEPSTUED, TFSTU
ATHL	Athletic Department	SRNATHL, SCRIBAEA
REG	Registrar	SRNREG, SEPREG, TFREG, SCRIBREG
STU	Student	SRNSTU, SEPSTUVW or SEPSTUED, TFSTU

Creating a new User Class

Degree Works uses the concept of a User Class to identify a user as having certain privileges.

1. Use Controller to create the new user-class in UCX_AUD012 (code can be up to 4 bytes long).
   Use the Copy function in Controller to copy the REG record and make changes.
2. Modify SHPCFG to assign or remove whatever keys you want to this user-class:

```
if (DGWUSERCLASS = "ABCD") then
  AddKey = SCRIBE   # Scribe
  RemKey = SDFIND   # disallow Find button
```

Important notes

1. Remember to do a daprestart after modifying SHPCFG.
2. You don't have to create a Shepherd group for your new user-class as you will be controlling access using SHPCFG.
3. After creating the user-class you can then bridge users with the new user-class.
   Ellucian does not recommend creating new user-classes. It is best to use the user-classes provided.

Granting User Access to Degree Works

Granting access to Scribe

Scribe is used to maintain degree requirements. Typically, only a few staff members in the Registrar's office are granted access to Scribe.
Users need the SCRIBE and SCRPARSE keys to access Scribe. Additionally, users will need the keys for the block types they have access to modify. Most users can be given SCRBLALL, which grants access to modify all block types. Users with limited access should be given the keys for the appropriate block type(s):

SCRBLATH – ATHLETE blocks
SCRBLAWR – AWARD blocks
SCRBLCOL – COLLEGE blocks
SCRBLCON – CONC blocks
SCRBLDEG – DEGREE blocks
SCRBLID – ID blocks
SCRBLLIB – LIBL blocks
SCRBLMAJ – MAJOR blocks
SCRBLMIN – MINOR blocks
SCRBLOTH – OTHER blocks
SCRBLPRG – PROGRAM blocks

SCRBLREQ – REQUISITE blocks
SCRBLSCH – SCHOOL blocks
SCRBLSPC – SPEC blocks

Granting access to Transfer Equivalency

Transfer Equivalency is used to process transfer equivalence and articulation. There are two components – Transfer Equivalency Self Service and Transfer Equivalency Admin.

Transfer Equivalency Self Service allows transfer prospects to map courses between their old school and yours.

The DWTESELF key is required to access Transfer Equivalency Self Service.

Transfer Equivalency Admin is an administrative tool to manage mappings and transfers. Typically, only a few staff members in the Registrar's office are granted access to Transfer Equivalency. The SRNTREQ key is required to access Transfer Equivalency Admin.

Granting access to Controller

Controller is used to maintain the codes and configuration settings that control the behavior of Degree Works. Typically, only a few staff members in the Registrar's or Information Technology office are granted access to Controller.

The CONTROL group should be given to these users to full access in Controller. However, individual keys can instead be added as needed for those who should be given limited access.

Granting access to Transit

Transit users are those typically in the Registrar's Office and the IT team. The TRANREG group should be given to these users to access the full list of reports and processors. However, individual keys can instead be added as needed for those who should be given limited access.

Banner schools:

In addition to the TRANREG group Banner schools should also give their users the TRANBAN group to gain access to the Banner-specific processors.

Colleague schools:

In addition to the TRANREG group Colleague schools should also give their users the TRANCLG group to gain access to the Colleague-specific processors.

Granting access to the Dashboard

Degree Works on the Web does not require installation of software on your PC other than your standard Web Browser (IE, Firefox or Safari). Degree Works on the Web provides access to many different services intended for use by various user classes. Typically, all students, advisors, faculty, deans, department heads, and Registrar's staff are granted access to the Dashboard.

Access to the specific services is granted via specific keys. The keys are granted to administrators, staff, faculty, advisors, students and Registrar based on the user class assigned in SHPCFG.

Anyone accessing Degree Works Dashboard must have the “SDWORKS” key.

Granting access to Web Notes

Besides the keys listed above to control access to web notes, a special combination of keys controls whether a user can modify any student’s notes or only their own notes. The combination of keys SDSTUME and SDNTEMOD restricts a student to modifying only the notes that they created themselves. Advisors and REG need to be given SDNTEMOD only (without SDSTUME) in order to update any student's notes.

Granting access to see Advisees’ records

Non-student users can be granted access to student records using a variety of keys. The three main categories of such users are:

Registrar – access to any student’s records

Advisor – access to any advisee records

Department head – access to records of students in one or more departments

However, some advisors may need to be setup so that their advisees are loaded once the advisor connects but the advisor can then choose to open up any student on campus. Mixing these categories is possible using the right combination of keys.

These are the primary keys that correspond to the three categories above:

SDSTUANY	user can access any student’s records
SDSTUMY	user can access advisees (user’s ID is on rad_goalData_dtl of students)
SDDEPART	user can access students in specified departments/majors (advisee filters on the user’s shp_user_mst)

Additional keys can be given to these three users if the user is allowed to access any student. Usually if you give one of these keys you also give the other.

SDFINDID	The ID search field on the main page is enabled
SDFIND	The Find button on the main page is enabled

Example setup scenarios:

Registrar user – access to any student

SDSTUANY

SDFINDID

SDFIND

Advisor user – advisees only

SDSTUMY

Advisor user – advisees are loaded but user can then search on any student

SDSTUMY

SDFINDID

SDFIND

Department head – access to Political Science department only

SDDEPART

Department head (senior) – access to Business department but can also access any student's record

SDDEPART

SDFINDID

SDFIND

Advisor / Dept head - access to advisees but can also access students in the Chemistry department

SDSTUMY

SDDEPART

Advisor / Dept head (senior) - access to advisees, students in the History dept and also any student

SDSTUMY

SDDEPART

SDFINDID

SDFIND

Users with the **SDSTUMY** key will see their advisees loaded in the Name drop-down list as soon as they connect to Degree Works.

Users with the **SDDEPART** key will see the students in their departments loaded in the Name drop-down list as soon as they connect to Degree Works.

Users with both the **SDSTUMY** and the **SDDEPART** key will see the advisees and the departmental students combined together, listed alphabetically, in the Name drop-down list.

These advisor and department head users that also have the SDFIND and SDFINDID keys can then choose to search on any student. However, once the target student(s) is found that student(s) will be loaded in to the Name drop-down list replacing the advisee or departmental students.

If the advisor or department head user then wishes to reacquire their list of advisees or departmental students the user must reconnect to Degree Works.

For added extra security, additional keys are needed that have no effect on the user interface.

ANYSTUID – give to users who are allowed to view/edit any student's record; department heads also need this key

ADVISIDS – give to users who are only allowed to view/edit their advisee records

Here is the complete list of keys for non-student access:

SDSTUANY

SDSTUMY

SDDEPART

SDFINDID

SDFIND

ANYSTUID

ADVISIDS

For setting up department heads please review the *Additional Advisee Filtering* section.

Database Privileges

We recommend that you give the Degree Works database user account DBA privileges while you are processing an update, to avoid any complications. However, once the update is complete, you may choose to restrict the account to the following minimum required privileges:

```
grant create table to dwschema;
grant unlimited tablespace to dwschema;
grant create session on dwschema;
alter user dwschema quota unlimited on dgw;
alter user dwschema quota unlimited on pseudotemp;
grant SELECT_CATALOG_ROLE to dwschema;
```

If your user account is something other than `dwschema`, replace it in the commands listed above.

Encrypted Data

The following data is stored in an encrypted form in the database.

Shepherd User Passwords

For users defined in the native Shepherd user database, the passwords can be optionally stored in an encrypted format. This can be enabled in Controller in the UCX CFG020 record with a key of WEBPARAMS. Set the “Encrypt Password” field to “Y”. If this field is modified, then all passwords must be reset. The encryption is a one-way digest using the SHA-1 algorithm. The password cannot be decrypted.

Shepherd Settings

There are several settings configured in the database that contain sensitive information, such as passwords. These settings are encrypted as they are maintained using Controller. The settings that are encrypted are listed in the *Degree Works Configuration Technical Guide*. The encryption is a 2-way (symmetric) encryption using a 128-bit AES block cypher. This means that the clear text unencoded value can be viewed in Controller. In order to view and edit these values, the user must be assigned an access key of SHENCRPT. Without this key, the encrypted entries will not be displayed in Controller. Before editing any encrypted entry, you must first enter the encryption key. This is stored in the `core.shpSetting.encryptionKey` setting. This entry is not encrypted and can only be seen or edited only by someone with the SHENCRPT key.

If any password is modified, the applications that rely on that setting should be restarted.

System Administration

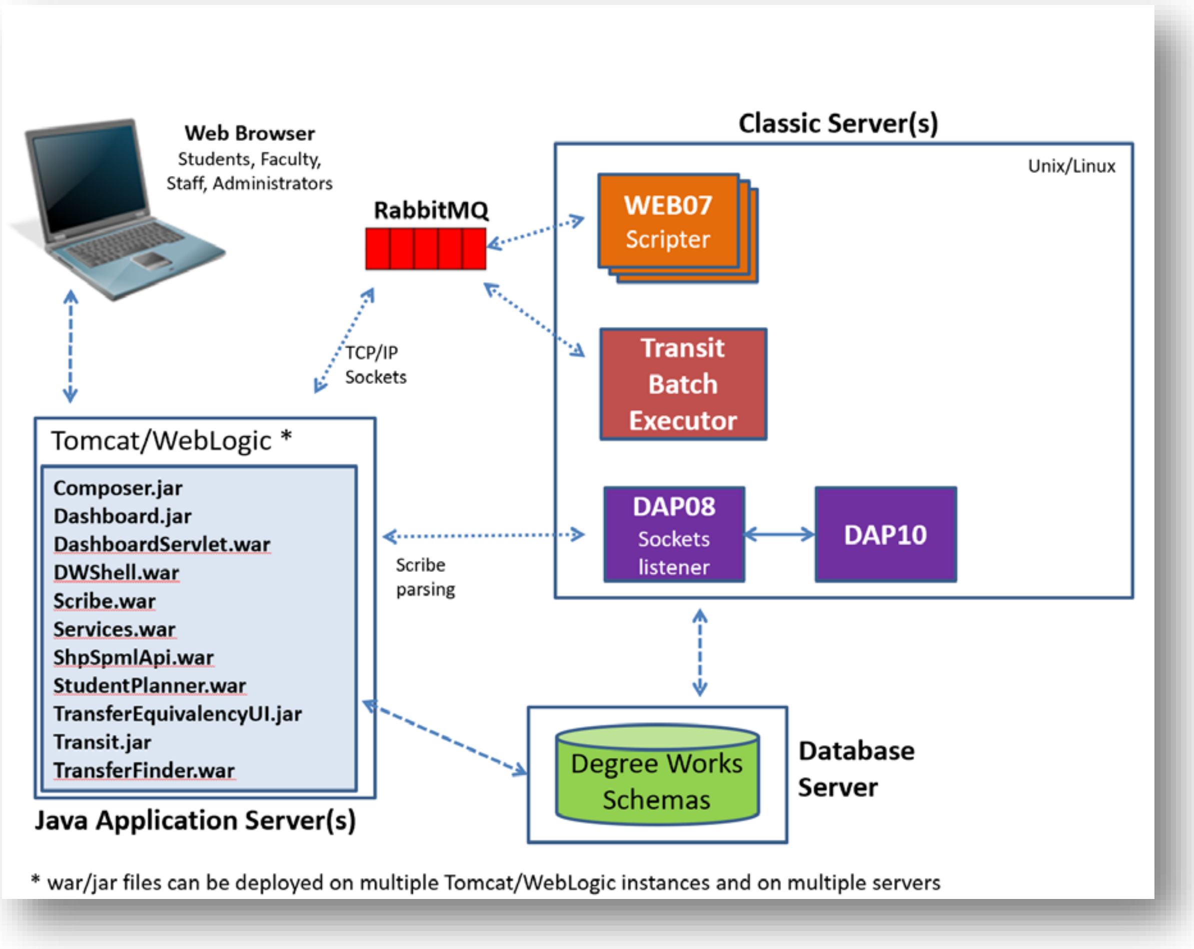
Communication with Degree Works exists in three ways:

Over the Web: Users talk to Degree Works using their browser

Batch jobs: A small pool of staff run maintenance jobs on the classic server.

System upkeep: On a nightly or weekly basis certain commands are run to keep Degree Works running

Degree Works Flow Diagram



Degree Works Web Applications

The following is a step-by-step description of the operation of a Degree Works Web session, and the fulfillment of a service request.

The Java Application Server houses either Tomcat or Weblogic, which serves the role as the HTTPD Web Server and usually runs on its own machine.

Request-Response Flow through the Java Application Server and Classic Server

1. When the user clicks a button in their browser a request is sent to the Java application server.
2. The Java application server passes the request to the dashboard.
3. The dashboard passes the request over a RabbitMQ message queue to the web07 daemons on the classic server.
4. The next free web07 daemon takes the request from the queue and then calls the scripter subroutine to process the request.
5. The scripter handles the request and then passes the completed script back through the RabbitMQ message queue.
6. The dashboard reads the message queue response and hands it off to the Java application server.
7. The Java application server obtains the output from the dashboard servlet and passes it back to the browser.

Request-Response Flow through the Java Application Server

The newer Degree Works applications, such as Scribe, are run through the Java Application Server and mostly do not flow through the classic server. The user's browser is communicating directly to the Java application server.

1. When the user clicks a button in their browser a request is sent to the Java application server.
2. The Java application server may communicate directly with the database or it may send a request to the classic server as explained above.

Several places exist where a timeout may occur while waiting for a response:

Java App Server: On the Java Application Server, Tomcat or WebLogic might timeout waiting for a response from the Degree Works classic server.

Request-Response Flow through Transit

Please refer to the Request-Response Flow through Transit section in the Transit Administration Guide.

RabbitMQ

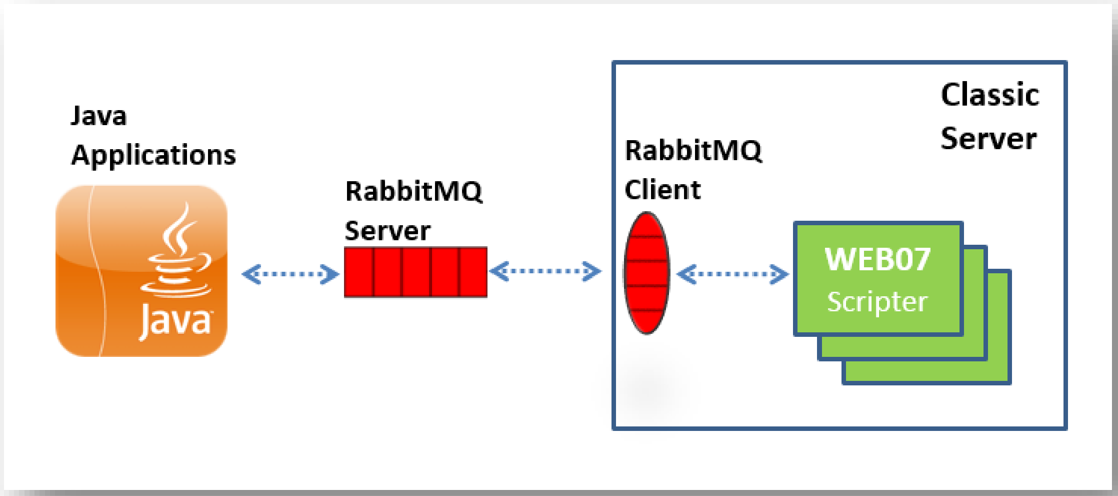
Degree Works uses RabbitMQ, an open source message brokering software that implements the Advanced Message Queuing Protocol (AMQP), to communicate between processes running on the same or on different servers. For information on installing and using RabbitMQ, please visit http://www.rabbitmq.com.

Two RabbitMQ software components must be installed for this communication to function:

RabbitMQ server: installed on any single Unix or Windows machine; must be running at all times

RabbitMQ client: installed on the classic server; the Degree Works software makes calls into the RabbitMQ C library to communicate with the server

The RabbitMQ software allows applications to communicate as shown in this diagram. The Java applications communicate with each other via the RabbitMQ Server and also communicate with the classic daemons via the RabbitMQ Server and with the classic daemons using the RabbitMQ Client to read from and write to the server.



The following settings are required for configuring RabbitMQ in an environment. The bolded settings must be unique for each environment.

classicConnector.amqp.channelCacheSize

classicConnector.amqp.exchange

classicConnector.amqp.timeout

core.amqp.broadcast.heartbeatSeconds

core.amqp.exchange.shpSettings

core.amqp.exchange.ucx

core.amqp.broker.host

core.amqp.broker.port

core.amqp.password

core.amqp.username

core.amqp.virtualHost

Information about the settings required for RabbitMQ can be found in the Shepherd Settings documentation.

On the machine where RabbitMQ server is installed you can use the **rabbitmqctl** tool to monitor the queues and verify everything is fully operational. Helpful rabbitmqctl commands include:

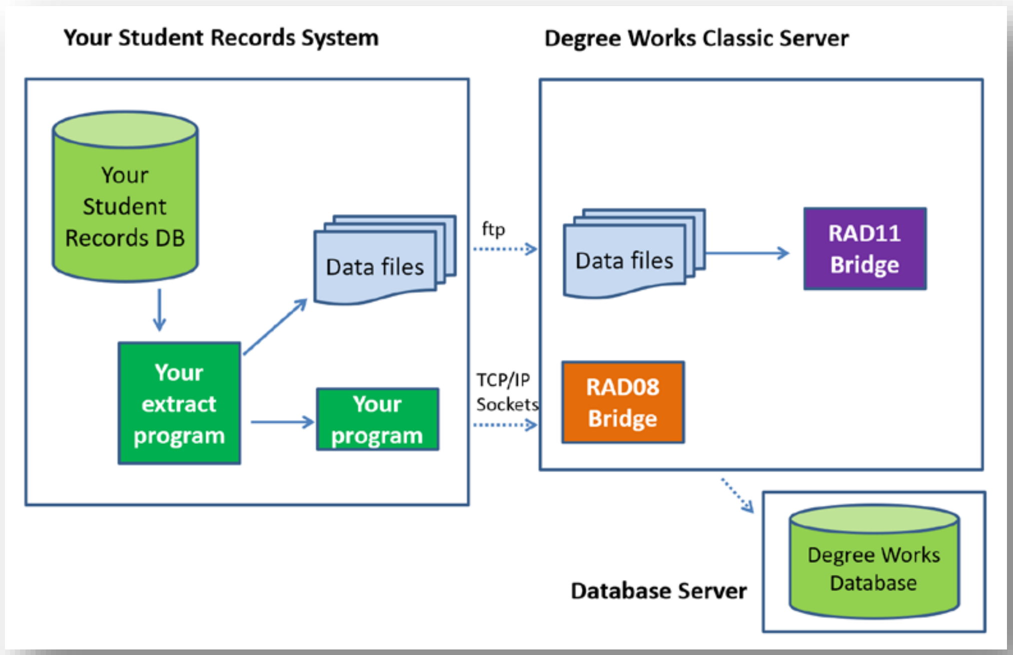
```
rabbitmqctl stop_app
rabbitmqctl start_app
rabbitmqctl status
rabbitmqctl report
rabbitmqctl list_queues -p /
```

You may also use “service rabbitmq-server status” to get some valuable information; the stop and start options works also as well as restart.

As always, you may need to use “sudo” to run these commands if you are not the root user on that machine.

The rabbitmqctl tool documentation is located here: https://www.rabbitmq.com/man/rabbitmqctl.1.man.html

Degree Works and Your Student Data



This diagram provides a step-by-step description of the operation of bridging your records to Degree Works. This flow is used by non-Banner, non-Colleague schools that write their own extract program and use RAD11 to bridge data into Degree Works.

Although this diagram shows your Student System living on a different machine than that where Degree Works resides, both systems may be on the same machine.

Banner schools should refer to the *Banner Considerations Technical Guide*.
Colleague schools should refer to the *Colleague Considerations Technical Guide*.

Request-Response Flow

Bridge Method #1 – batch loading of student and other data on a nightly basis

1. Your Extract Program reads from your Student Records System and writes the data to one or more data files.
2. The data files are FTP'd to the Degree Works system.
3. The RAD11 batch program is run pointing to the specific BIF file to be loaded.
4. RAD11 removes all data from the RAD database for a given ID and inserts the new, bridged data.

Bridge Method #2 – used for dynamically sending data for one student

1. Your program pushes data to the RAD08 Bridge Listener. The program may be asked to send this data because of change to student data within or outside of the context of a user accessing Degree Works.
2. The RAD08 Bridge Listener receives the student data and writes it to the Degree Works database.
3. After RAD08 returns a FINISHED message, your program may proceed with sending a run audit request to Degree Works to be sure the latest audit for the student reflects the data changes.

Maintaining Degree Works

Degree Works software requires that certain system management tasks be performed on a regular basis. Some critical tasks must be restarted after a system failure. Other tasks need to be done on a regular daily, weekly, monthly, or yearly basis. And still other tasks must be performed as needed.

This section is a checklist of the system management tasks associated with the Degree Works software package. Please make sure your system manager has a copy of this document.

Restarting Applications

There are times when you will need to restart the Degree Works applications. You can restart the daemons running in classic by running webrestart and daprestart in Transit or by running those commands directly on the classic server's Unix console. To restart the Java applications you need to restart them in Tomcat or WebLogic.

When you make changes to **UCX** records via Controller or if you bridge in new UCX values, you do not need to restart any of the applications. Messages are broadcast over RabbitMQ to notify each of the applications of a change to the UCX. The software recognizes the changes and acts appropriately when new requests are received.

When you make changes to **Shepherd Settings** via Controller, save some exceptions, you do not need to restart any of the applications. Messages are broadcast over RabbitMQ to notify each of the applications of a change to the settings. The software recognizes the changes and acts appropriately when new requests are received. However, when you make changes to these settings you will need to restart the classic and Java applications:

articulation.*
classicConnector.amqp*
core.amqp.*
core.apiClient.*
core.security.authenticationType
core.security.cas.*
core.security.externalAccessManager.*
core.security.ldap.*
core.security.passport.*
core.security.password*
core.security.saml.*
core.security.shp*

When you make changes to **Shpscripts** via Composer you do not need to restart the classic daemons. The web07 daemons recognize when any shpscript has changed and will reread the shpscript from the database when a new request is received from the dashboard.

When you make changes to **properties** via Composer you do not need to restart the Java applications. The properties will be refreshed by the applications after about 20 seconds. However; a user currently logged into the application may need to sign out and sign on again before the changes will be visible.

Cron setup for Degree Works

Clients can configure cron to schedule reports or processes to run on a daily basis. This is most simply done by scheduling a script to run in cron, where the script runs your daily processes.

As the Degree Works administrative user, save your daily maintenance script on your classic server and configure cron to run it. Use the *crontab -e* command to create or update cron.

```
$ crontab –e
# Run daily extract and maintenance jobs at 2:00 am on weekdays
0 2 * * 2-6 /home/dwadmin/cron/NightlyMaintenance.sh
```

In the example daily maintenance script below, a file named dw_maint_yymmdd.log will be created to contain messages from each process that is to be run. You must create the directory to contain these log files, e.g. /home/dwadmin/cron/logs.

Be sure to modify the path to the dwenv and dwenv.config scripts, so that the Degree Works environment is sourced and all of the scripts and processors can be located.

If you wish to run your student extract job via cron, you will need to run *launchjob* and provide a Transit parameter file in JSON format. For more information please see the *launchjob* section in the *Transit Administration Guide* and either the *Banner Considerations*, *Colleague Considerations* or, for other schools, the *Bridge Inface Format Technical Guide*.

```
#!/usr/bin/ksh
#
# Run this script daily via cron

# Source the Degree Works environment
. /dworks/app/scripts/dwenv /dworks/dwenv.config

# create log file and name with date
export LOGFILE="/home/dwamin/cron/logs/`/bin/date +dw_maint_%y%m%d.log`"

echo "Starting daily extracts"                     >$LOGFILE
date >>$LOGFILE

echo "Running student extract via transit"      >>$LOGFILE
launchjob $ADMIN_HOME/myjobs/rad30Stu.json      >>$LOGFILE 2>&1
echo "Daily extracts are complete"              >>$LOGFILE

# remove all output older than 7 days
rmoldfiles $ADMIN_HOME/dgwspool 7             >> $LOGFILE
rmoldfiles $ADMIN_HOME/logdebug 7             >> $LOGFILE
rmoldfiles $ADMIN_HOME/jobdata  7             >> $LOGFILE


echo "Restarting dap daemons"                 >> $LOGFILE
daprestart                                     >> $LOGFILE
echo "Restarting web daemons"                 >> $LOGFILE
webrestart                                     >> $LOGFILE

# email the logfile results
mailx -s DW_extract_results admin@mySchool.edu < $LOGFILE
```

After System Failure

It is recommended that these tasks be accomplished after a system failure. It is suggested that you add appropriate symlinks in your /etc/rc3.d directory (or wherever is appropriate for your operating system) to the dw* scripts in the /etc/init.d directory so that the Degree Works daemon processes startup when your machine is booted. Only create symlinks for the daemons you want started. For example, if you are not using the Banner prerequisite daemons then do not create a symlink do the dw*.preq script in /etc/init.d.

You will need to execute the .profile in the same shell.

Launch other regularly scheduled Degree Works jobs.

Some Degree Works jobs are scheduled to run on a regular basis and should be launched if they are not in the job queue after recovery from a system failure. Your users may have other such jobs (the Bridge program), or you may have regular system management jobs (the backup) that need to be launched after the machine is restarted.

OS change or recompiling in 64-bit

When you want to create a new Degree Works environment on another machine you may lose your audit history. When the source machine has a different OS from the target machine the historic audits created on the source machine and stored in the database cannot be accessed on the target machine. For example, if you are setting up a new production environment on Linux and your old production machine was Sun Solaris then those audits that were created on Sun Solaris cannot be viewed in the new environment on Linux. These historic audits are essentially lost. This same issue occurs if audits were created in one environment compiled in 32-bit and an attempt is made to access the audit from a 64-bit environment – even if the OS is the same in both environments. If you are considering creating a new 64-bit production environment or simply want to now recompile your current environment in 64-bit mode you need to be aware that historic audits will be lost. (The same is true if you are switching from 32-bit to 64-bit.)

Unlike with audits, your Scribe blocks are safe when moving to a new OS or are recompiling in 64-bit mode. The only wrinkle here is that you need to delete the contents of your admin/daptrees directory in the new environment (or before recompiling in 64-bit mode) and run a dap16all.

Daily Tasks

1. Check disk space for free space and fragmentation.

2. Run *webrestart* and *daprestart* as part of the nightly job. Also run *preqrestart* and *radrestart* if you are using them. If you are always wanting CPA data to be generated for new audits run *resstop* before running the bridge/extract and *resstart* when it is done.

3. Run student extract. See the *Cron setup* section above.

4. Other daily jobs.

    a. Some Degree Works jobs are scheduled to run on a regular basis and should be launched if they are not in the job queue. Your users may have other such jobs (the Bridge program), or you may have regular system management jobs (the backup) that need to be launched.

5. Check execution reports for Degree Works jobs.

    Examine the .log files in the admin/logdebug directory for the daemons. Use Transit to examine the jobs that were launched.

7. A daily partial backup (data only) of the Production account is important for safety and recovery.

If you would like advice on when to schedule partial or full backups, please contact Ellucian.

Monthly Tasks

Copy test data from databases in Production to TEST environment using a database tool.

Note: You may not want to copy all audits (and associated CPA data) from Production to TEST, as that data is not needed in TEST and it is a lot of data to copy. You should consider excluding these tables from your copy process: dap_audit_dtl, dap_audtree_dtl, dap_result_dtl, dap_resclass_dtl, dap_resnoncr_dtl.

The following steps need to be executed after copying your data into TEST:

- Copy the files from daptrees in Production to TEST, overwriting those in TEST.

If you don’t want to keep the production notes and exceptions in TEST you can remove the contents of these tables: dap_note_txt_dtl, dap_note_dtl and dap_except_dtl

Remove old rad_log_dtl records using the **dapdelradlogs** script:

```
   $ dapdelradlogs 20101231
```

All records created prior to the date specified are deleted. In this example records created before December 31, 2010 are deleted. This script should be run about once a month to help clean up the Degree Works tablespace.

Semi-Yearly/Yearly Tasks

Update the “current” term used as the default.

Change the Current Term in UCX-CFG020 WEBPARAMS.

This should be done whenever the “bridged” term changes. This term is used as the default.

Patching Code between Releases

Sometimes you may encounter problems and require an immediate fix from the ActionLine. When you are provided with a new file with the fix you should archive your existing file using the “archive” script.

For example, if the ActionLine gives you a new version of dap43s.c you should first archive your old file by doing one of the following:

```
$ cd c
$ archive dap43s.c
```

Or

```
$ archive $DGWHOME/src/c/dap43s.c
```

The archive script will copy the file into the *$DGWHOME/archive* directory keeping the same directory structure (*archive/src/c* in this case). It is important to archive your existing version in case the new version you receive has bigger problems and you need to revert back to your archived version.

At any time you can view your archived files by doing the following:
```
$ ll –R $DGWHOME/archive
```

The archive script cannot be not used to archive files under *$LOCAL_HOME or $ADMIN_HOME* – it can only be used to archive files under *$DGWHOME*.

As Needed Tasks

Miscellaneous

1) Process Degree Works updates

Degree Works updates are NOT optional. If you do not process the updates then your software becomes increasingly outdated. At some point, Ellucian will stop supporting old versions of Degree Works at your site if you have received an update but not processed it.

2) Add Degree Works users

When personnel turnover takes place, you may need to add a new user for the Degree Works system. Instructions for adding new users are in this document. Consult that section for the details of how to add a Degree Works user. Ellucian strongly recommends that the user be added first into the TEST account and then, after testing and training, be added into the Production account.

3) Transfer blocks between two different environments

Transferring blocks between environments consists of three steps:
- Use *dapblockunloa*d to unload tables relating to Scribe blocks
- Use *dapblockload* to load the tables relating to Scribe blocks
- Run *DAP16* to reparse your new set of blocks

Use *dapblockunload* to unload tables relating to Scribe blocks
You can specify that all blocks be unloaded, just the RA blocks or just the RB blocks (the RB blocks are those generated from student plans). The *dapblockunload* script takes in a parameter of R, RA or RB with RA being the default if no parameter is specified.

The contents of the **dap-req-block** table are unloaded along with the appropriate **dap-next-id-mst** records.

Note: You cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.

The tables are unloaded to a tar file in the admin/datac directory. If a tar file name is not specified then a file with the current date will be used. For example, **reqblocks20160317.tar**.
- The block columns are unloaded to R*.fields files
- The block text is unloaded to R*.text files.

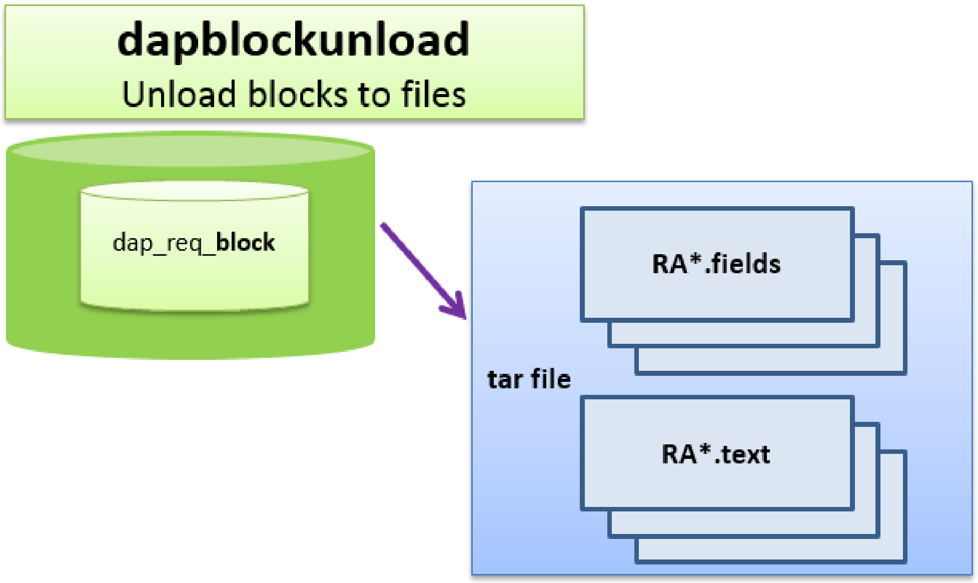
The dap-next-id-mst records are unloaded to DWNXTMST.dmp and DWNEXTMST_RB_.dmp files.
All of the files are then combined into one big tar file.

Examples of how to run **dapblockunload**:

```
$ dapblockunload       # unloads all RA blocks to tar file of current
date
$ dapblockunload R     # unloads all blocks to tar file of current date
$ dapblockunload RA    # unloads all RA blocks to tar file of current
date
$ dapblockunload RB    # unloads all RB blocks to tar file of current
date
$ dapblockunload R myfile.tar # unloads blocks to specified tar file
```

Using **dapblockunload** does not remove the blocks from the current environment; they are merely copied to files.

Copy the files to the new environment
C Copy the **.tar** file created by **dapblockunload** into the **admin/datac** directory in the environment in which you want to load the requirements. If the environment is on another machine you will need to FTP the files using BINARY as the transfer type.



Use *dapblockload* to load the tables relating to Scribe blocks from a tar file into the database

You can specify that all blocks be loaded, just the RA blocks or just the RB blocks (the RB blocks are those generated from student plans).

- The first parameter is either R, RA or RB (RA is the default if no parameter is specified).
- The second parameter is the name of the tar file; a tar file name is required.

The current contents of **dap-req-block** and the R dap-next-id-mst record are copied to archive/datac as a tar file called **reqblockarchive.tar**. The contents of dap-req-block, dap-req-crs-dtl and dap-req-link-dtl are deleted along with the RA and/or RB dap-next-id-mst records. However, if RA is specified as the parameter only the RA blocks are deleted. If RB is specified then only the RB blocks are deleted.

The R*.fields and R*.text files from the tar file are then processed to create insert and update sql commands. The entire set of insert sql statements are first run to create the **dap-req-block** records with empty CLOB/text fields. The update sql statements are then run to create the requirement text data. There will be one insert statement for each line of text in each block so expect a lot of database updates to occur and expect this to take at least several minutes to run.

The appropriate RA or RB datac files are then imported.

Examples of how to run **dapblockunload** to load a **tar** file:

```
      $ dapblockload R  myfile.tar # load blocks from tar file in admin/datac
      $ dapblockload RA myfile.tar # load blocks from tar file in admin/datac
      $ dapblockload RB myfile.tar # load blocks from tar file in admin/datac
```

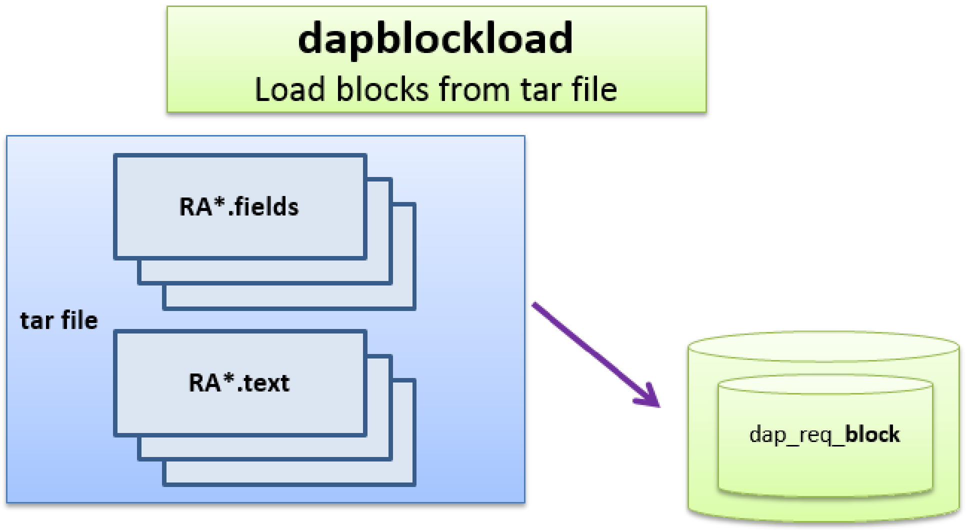
Since the dapblockload output is so long you should consider redirecting to a file:

```
      $ dapblockload RA myfile.tar > my.out 2>&1
```

If you have tee on your system you might consider using it to get the output in the file and to your screen:

```
      $ dapblockload RA myfile.tar 2>&1 | tee my.out
```

(Do *which tee* to confirm you have it installed.)
Once the load finishes you can then examine the contents of my.out to check for errors.



Run DAP16 to reparse your new set of blocks
Once the new blocks are loaded you must use Transit to run DAP16 to reparse all of the blocks. Be sure to review the DAP16 report to examine any errors found in the parsing process. Fix all errors found. You may run DAP16 from Transit or may run the dap16all script from the command line to reparse all of your blocks.

5) Transfer mappings between two different environments

Use *dapmapunload* to unload tables relating to Transfer Equivalency mappings.

The tables unloaded are dap-mapping-dtl, dap-map-cond-dtl, dap-title-dtl and the M dap-next-id-mst record. These tables are unloaded to files in the datac directory called DWMAPDTL, DWCNDDTL, DWTTLDTL, DWNXTMMST, respectively. Using dapmapunload does not remove the blocks from the current environment; they are merely copied to files.

Copy the files to the new environment.
Copy the three files just extracted into the datac directory in the environment in which you wish to load the mappings. If the environment is on another machine you will need to FTP the files using BINARY as the transfer type.

Use *dapmapload* to load the tables relating to Transfer Equivalency mappings.
The current contents of the dap-mapping-dtl, dap-map-cond-dtl, dap-title-dtl and the M dap-next-id-mst record are copied to archive/datac. The contents of dap-map-dtl, dap-map-cond-dtl, and dap-title-dtl are deleted along with the M dap-next-id-mst record. The datac files are then imported.

Note: You cannot use the load/unload scripts to copy between two environments on two different versions of Degree Works.

6) Transfer UCX records between two different environments

Note: do not use these scripts to copy the UCX tables between environments that are on different versions of the Degree Works software. For example, if PROD is on the 4.1.4 release and TEST is on the 4.1.7 release you should not use these scripts. Some of the tables have entries that are specific to the software version, such as UCX-SYS935 and others. You can instead use the bulk options feature in Controller to load/unload a table at a time.

Use *dapucxunload* to unload the UCX tables
All UCX tables are unloaded and placed in the datac directory in a file called DWUCXALL.tar.gz. Using dapucxunload does not remove the UCX records from the current environment, they are merely copied out.

Copy the file to the new environment
Copy the file just extracted into the datac directory in the environment in which you wish to load the UCX tables. If the environment is on another machine you will need to FTP the file using BINARY as the transfer type.

Use *dapucxload* to load the UCX tables
The current contents of the UCX records are copied to archive/datac. The contents of the UCX are deleted. The contents of the DWUCXALL file are then imported.

7) Delete old student data from your Degree Works database

After bridging students into Degree Works for many years, you may wish to remove inactive student data from the Degree Works database. You can run the script **deletestu** to delete students by bridge date. To run this script, issue the command

```
deletestu YYYYMMDD
```

where YYYYMMDD will be used to select students who have been bridged on or before this date. The script will create a file of the selected student ID's and store it in the $LOCAL_HOME/sql directory as **deletestu.ids**. The "bannerextract deleteid" process will then be run on the deletestu.ids file, removing these students from the Degree Works database.

If needed, you can edit the deletestu.ids file before the delete process takes place. To do so, respond with "N" when prompted "Continue?" after the script displays the number of students to be deleted. This will terminate deletestu, but you may then edit deletestu.ids to modify the student

ID's to be deleted. After modifying the file, *cd* out of the *local/sql directory* (exiting local/sql is important!) and issue the command

```
bannerextract deleteid deletestu.ids
```

Note: Only Banner schools can use the *deletestu* script.

8) Cloning database from TEST to PRODUCTION

When you are ready to go live with Degree Works you may want to clone the Degree Works database from your test environment to your production environment. You will need to review certain Shepherd settings (shp_settings_mst table). These settings should be different between environments but others may need to be different also based on your setup.

```
core.security.cas.callbackUrl
core.classicUrl.serverDomain
classicConnector.dap08.port
classicConnector.serverNameOrIp
```

If you can't get into Controller to modify these settings you can use the **shpsettingsset** script to change the value. For example:

$ shpsettings**set** classicConnector.dap08.port 1934

You may also use **shpsettingsshow** to view the settings. Simply specify any part of the setting:

$ shpsettings**show** classicConnector

Please note: do not use these scripts to copy the UCX tables between environments that are on different versions of the Degree Works software. For example, if PROD is on the 4.1.4 release and TEST is on the 4.1.6 release you should not use these scripts. Some of the tables have entries that are specific to the software version, such as UCX-SYS935 and others. You can instead use tableunload/tableload or the bulk options feature in Controller to load/unload a table at a time.

Monitoring Service Access

daphits

The daphits command displays the number of times a particular Degree Works service has been accessed since a given date.

dapreset

The dapreset command resets the access count and date for the Degree Works services.

You can also examine the logdebug/web.log file to see what and when certain requests occurred. This log file is cleared out whenever the web jobs are restarted.

Also review the webanalyze script in the *Performance* section; this script can be very helpful in understanding access.

Maintain Email notification configuration

In Controller go to the Configuration tab and search for "email" to find the email settings.

You may optionally set this email address value to give the address to be used when a user clicks Reply. If this value is not set, the To address as used as the reply-to address.

```
core.notification.fromEmail
```

You may optinally set a CC email address that is used for all of the processors listed above. This might be useful for CC'ing the registrar or an office assistant.

```
  core.notification.ccEmail
```

Ensure that the SMTP_MAIL_SERVER variable is defined in your environment. If it is not, you may set it here in *dwenv.config*. If this is not set to a correctly configured mail server, e-mail notifications will not work.

# Set this to your machine's mail server if not already set; eg: mailhost.mymachine.edu

```
export SMTP_MAIL_SERVER=mailhost.mymachine.com
```

Database Credentials Changes

When you change the database user or password for the configured Degree Works user (see the DB_LOGIN environment variable) or the SIS database user (see the DB_LOGIN_BANNER or DB_LOGIN_CLG environment variable) you must configure the new credentials in 3 different areas.

In all the configurations, an attempt has been made to hide the database passwords as much as possible. That is, they do not appear unencrypted in files, or in a user's environment.

Important

This does not prevent a user who is logged onto the server from seeing the plain password. It is still critical to secure access to the server to only those individuals who would be authorized to have full access to the database.

Classic Applications

The database connections for the classic software are configured in the file dwenv.config in the $DGWBASE directory. This file, which is sourced in during the logon process, sets the DB_LOGIN environment variable. This variable contains the user name (schema) and connection. For example:

```
DB_LOGIN="dgwmgr@dwdevl"
export DB_LOGIN
```

In the above example, "dgwmgr" is the user name and "dwdevl" is the connection name.

The database password is configured using the setdbpasswords command. For example,

```
setdbpasswords --password mydwpassword --sispassword mysispassword
```

In the above example, "mydwpassword" is the Degree Works database password associated with the DB_LOGIN user, and "mysispassword" is the password associated with the SIS password.

Java Tomcat Applications

For the applications which are deployed in a Tomcat instance, the connection information is stored in the server.xml file. It will look something like the following:

```
<Resource auth="Container"
   name="DwProdDatasource"
   url="jdbc:oracle:thin:@dbhost.myschool.edu:1521/dwprod"
   username="dgwmgr"
   password="ENC(Smf/7X6r2Eqw2c2YIkCIJw==)"
   defaultAutoCommit="true"
   driverClassName="oracle.jdbc.OracleDriver"
   factory="org.apache.commons.dbcp.BasicDataSourceFactory"
   maxActive="30" maxIdle="10" maxWait="1000"
   testOnBorrow="true"
   testWhileIdle="true"
   timeBetweenEvictionRunsMillis="1800000"
   type="javax.sql.DataSource"
   validationQuery="select * from dual"
   validationQueryTimeout="1"/>
```

In the above, the "username" field is the user being used by the application to connect to the database. The "password" is an encrypted string enclosed in "ENC()". To get the encrypted string, you should login to the classic environment that uses this database (has the same user) and issue the following command:

```
Showdbpasswords
```

You must have already run the setdbpasswords command, as described above, in that environment. It will produce the following output (example):

```
Substitute the following for the DW password in the server.xml datasource and
jdbc.properties files:


ENC(Smf/7X6r2Eqw2c2YIkCIJw==)
```

Copy the last line into the server.xml file.

Java Applications on the Classic Server

There are a few Java applications that are run on the classic server. These require that the username and password be configured in a jdbc.properties file in the $ADMIN_HOME/common directory. Here is an example (abridged):

```
dw.jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
dw.jdbc.url=jdbc:oracle:thin:@dbhost.myschool.edu:1521/dwprod
dw.jdbc.username=dgwmgr
dw.jdbc.password=ENC(Smf/7X6r2Eqw2c2YIkCIJw==)
```

This uses the same password string described in the above Tomcat section.

Java Self-contained Web Applications

These Java applications, such as Composer, Gateway, and Transfer Equivalency Self-service, are not deployed via a Tomcat instance. Instead, they are started with a shell script. Here is an example (abridged):

```
export
SPRING_DATASOURCE_URL="jdbc:oracle:thin:@dbhost.myschool.edu:1521/dwprod"
export SPRING_DATASOURCE_USERNAME="dgwmgr"
export SPRING_DATASOURCE_PASSWORD="ENC(Smf/7X6r2Eqw2c2YIkCIJw==)"
export DEPLOY_LOCATION="/u01/dw/webapps"

...

/usr/java/latest/bin/java -jar $DEPLOY_LOCATION/treqss.jar \
> /usr/tomcat/default/logs/startup-treqss.log 2>&1 &
```

The SPRING_DATASOURCE_PASSWORD variable should contain the same encrypted string as was described in the above Java Tomcat Applications section.

Customizing Degree Works source code

If you modify any of the standard Degree Works C or Pro*C code you need to place your changes under the **$LOCAL_HOME** directory in one or more of the following directories:
../local/src/include
../local/src/c
../local/src/ec

When a build occurs, the script will find and use the version found in these directories and ignore the version stored under $DGWHOME/src. Subsequent Degree Works updates will place new versions in $DGWHOME and will not overwrite your custom changes in $LOCAL_HOME. Of course, you will need to integrate any new changes in each release into your custom versions and rebuild.

Degree Works Standing Daemons

The following daemons should always be running on the classic server for the Degree Works software to function properly:

dap08	Scribe sockets listener daemon
dap10	Scribe daemon
web07	Dashboard daemons
transitexecutor.jar	Transit Batch Executor

Optionally you may choose to run these daemons:

dap25	Create CPA results
rad08	Dynamic Bridge (non-Banner and non-Colleague)
dap61	Banner pre-requisite checker
dap62	Banner pre-requisite descriptions

The following jobs can be controlled by using the following scripts which can be entered at the system prompt. The restart scripts can also be run via Transit. However, if you have multiple classic servers running the restarts will only occur on the machines where the Transit Batch Executor is running. If it is running on multiple machines the restart request could go to any of the machines.

When you have multiple classic servers it is important to know which deamons should be or should not be run on each server:

dap08/dap10	run on only one classic server
dap25	run on only one classic server
web07	run on one or more servers as needed
rad08	run on one or more servers as needed
dap61/dap62	run on one or more servers as needed
transitexecutor	run on one or more servers as needed

webstart

The webstart script uses the dwinit.web script located in $DGWHOME/initd.

The webstart script starts up a utl01 process and tells is to startup a series of web07 child processes. You control the number of web07 processes that are started by setting this variable in $DGWBASE/dwenv.config:

```
DW_WEB07_COUNT
```

The web07 count is usually a high number like 50 or 100; you will need to test to determine what works best for your system.

webrestart

Runs **webstop** followed by **webstart**.

webstop

This stops the utl01 process with scope=web causing the child web07 processes to also stop.

dapstart

Please see the notes above for **webstart**. The difference is that **dwinit.dap** is used and the variables in dwenv.config that you need to configure are these:

DW_DAP08_COUNT

DW_DAP10_COUNT

The dap08 count must be 1 and the dap10 count is usually 1 but can be higher. However, with such a small set of users having access to Scribe there is no need to set this higher than 1.

daprestart

Runs **dapstop** followed by **dapstart**. Please also see the special notes under **webrestart**.

dapstop

This stops the utl01 process with scope=dap causing the child dap08 and dap10 processes to also stop.

radstart

Please see the notes above for **webstart**. The difference is that **dwinit.rad** is used and the variable in dwenv.config that you need to configure is this:

DW_RAD08_COUNT

This should be a low number if you don't send Degree Works many simultaneous dynamic bridge requests your system. This can be higher if you have many simultaneous pushes of data. Most schools do not use rad08.

radrestart

Runs **radstop** followed by **radstart**.

radstop

This stops the utl01 process with scope=rad causing the child rad08 processes to also stop.

resstart

Please see the notes above for **webstart**. The difference is that **dwinit.res** is used and the variable in dwenv.config that you need to configure is this:

DW_DAP25_COUNT

This should be a low number if you don't want Degree Works to spent a lot of resources creating CPA data for new audits. This can be higher if you can devote sufficient resources to create CPA data.

resrestart

Runs **resstop** followed by **resstart**.

resstop

This stops the utl01 process with scope=res causing the child dap25 processes to also stop.

tbestart

This script starts the transitexecutor.jar running the Transit Batch Executor. This is needed to allow Transit to launch jobs. This is also needed for the launchjob script to function.

tberestart

Runs **tbestop** followed by **tbestart**. Unlike with daprestart and webrestart, you do not need to run tberestart every night in your cron script.

tbestop

This stops the transitexecutor.jar.

preqstart

Please see the notes above for **webstart**. The difference is that **dwinit.preq** is used and the variables in dwenv.config that you need to configure are these:

DW_DAP61_COUNT

DW_DAP62_COUNT

The dap61 and dap62 count can be higher if you have many requests coming from Banner. Running preqshow can show if the queue is getting backed up due to not having enough daemons running.

preqrestart

Runs **preqstop** followed by **preqstart**.

preqstop

This stops the utl01 process with scope=preq causing the child dap61 and dap62 processes to also stop.

Check on Running Jobs

There are some scripts to display the specific daemon processes:

dapshow

This shows the running dap08 and dap10 processes and the parent utl01 process with scope=dap. Also shown is the status of the message queue used to communicate between the dap daemons.

```
$ dapshow
OWNER    PID   PPID  STARTED  CPUTIME COMMAND
======== ===== ===== ======== =======
==========================================
dwadmin   9785     1 Mar 7    00:00:00 /dw/dwprod/app/bin/utl01x
db=dwadmin_prod@ scope=dap

dwadmin   9786  9785 Mar 7    00:00:00 dap08x –p7701 db=dwadmin_prod@
dwadmin   9787  9785 Mar 7    00:00:00 dap10x db=dwadmin_prod@

===== Degree Works Message Queue: Key: 603 = 0x25b =====
MsgQ Key   MsgId      Owner      Perms    #Bytes       #Msgs
0x0000025b 31555591   dwadmin    666        0             0
```

webshow

This shows the running web07 processes and the parent utl01 process with scope=web. Also shown is the status of the message queue used to communicate between the web daemons.

```
$ webshow
OWNER    PID   PPID  STARTED  CPUTIME COMMAND
======== ===== ===== ======== =======
==========================================
dwadmin   8785     1 Mar 7    00:00:00 /dw/dwprod/app/bin/utl01x
db=dwadmin_prod@ scope=web

dwadmin   8830  8785 Mar 7    00:00:00 web07x db=dwadmin_prod@
dwadmin   8831  8785 Mar 7    00:00:00 web07x db=dwadmin_prod@
dwadmin   8832  8785 Mar 7    00:00:00 web07x db=dwadmin_prod@

===== Degree Works Web Message Queue: Key: 603 = 0x25b =====
MsgQ Key   MsgId      Owner      Perms    #Bytes       #Msgs
0x0000025b 31555591   dwadmin    666        0             0
```

radshow

This shows the running rad08 processes and the parent utl01 process with scope=rad. Note that the number of rad08 processes running will always be one more than the number specified in DW_RAD08_COUNT. This is because utl01 always spawns a single rad08 and this rad08 parent then spawns the number of children specified by DW_RAD08_COUNT. In the example below the DW_RAD08_COUNT was 3. Note that the -3 is what is passed to the rad08 parent using the –c parameter.

```
$ radshow
```

```
OWNER    PID   PPID  STARTED  CPUTIME COMMAND
======== ===== ===== ======== =======
==========================================
dwadmin    856     1 11:33    00:00:00 /dw/dwprod/app/bin/utl01x
db=dwadmin_prod@ scope=rad

dwadmin    858   856 11:33    00:00:00 rad08x db=dwadmin_prod@ -p8001 -c3
dwadmin    859   858 11:33    00:00:00 rad08x db=dwadmin_prod@ -p8001 -c3
dwadmin    860   858 11:33    00:00:00 rad08x db=dwadmin_prod@ -p8001 -c3
dwadmin    861   858 11:33    00:00:00 rad08x db=dwadmin_prod@ -p8001 -c3
```

dwadmin   861   858 11:33   00:00:00 rad08x db=dwadmin_prod@ -p8001 -c3

Degree Works Troubleshooting

The following is an outline of troubleshooting topics. Some are performed by the client, others by the vendor after receiving information from the client. The purpose of the outline is to inform you of the issues involved in troubleshooting problems.

I. Troubleshooting a problem

A. Get information
- 1. Can the problem be reproduced?
- 2. In which environment does the problem occur?
  - a. If one account why is it not a problem in the other account?
  - b. Is the software or data different?
- 3. When did the problem start? After an update? After an electrical storm?
- 4. Does this happen for every ID or just this one?
- 5. Is this problem urgent?
- 6. Obtain a debug file

B. Attempt to reproduce the problem
- 1. Attempt in test account
- 2. Attempt in live account

C. Focus on one problem at a time

II. Use available tools

A. Browser
- 1. Do a View-Source to see what is being sent
- 2. Check for localized scripts or web files
- 3. Check for older versions

B. SQL
- 1. Use database tools to examine the database
- 2. Use database tools to modify the database

III. Get a debug file

A. Classic environment – web07
- 1. Use *debugon* to turn on debug.
- 2. Run *webrestart*
- 3. Run a test in the dashboard.
- 4. Examine web07.nnnn.xml debug file

A. Classic environment – dap10
- 1. Use *debugon* to turn on debug.
- 2. Run *daprestart*
- 3. Run a test in Scribe – parse or save.
- 4. Examine dap10.nnnn.xml debug file

C. Java Applications
- 1. Ensure your user has the DEBUG Shepherd key

2. Enable debugging from the Debug page in the application
3. Note your Debug Tag
4. Examine the log file on the application server, searching for your Debug Tag.

Debugon and Debugoff

The debugon alias makes it easier for you to turn on debugging when requested by Ellucian. Issuing "debugon" works as well as "debugon frog" - the logdebug files created will have "frog" in their names so that you can easily find the ones that you created. Note that you must do a webrestart or daprestart within five minutes of issuing the debugon command; this helps prevent you from leaving debugging turned on accidentally.

The debugoff alias helps to easily turn debugging off again:

```
$ debugon frog
$ webrestart
 (do some testing on the dashboard)
$ cd logdebug
$ ll *frog.xml - these will be your debug files

$ env | grep DEBUG
DWDEBUG=1
DW_LOGDEBUG_DIR=/dw/dwprod/admin/logdebug
DW_LOGDEBUG_PID=frog
```

Relaying Information to Support

If it is possible to reproduce the problem take special care to document the steps that you are taking. Enclose these steps when conveying the information to support. Determine if the problem encountered is pervasive. Does the error or problem encountered occur in all instances of running the particular program or performing the specific task. When a specific error message is relayed from an application it is recommended to include this message. Capturing a screen shot of the error message displayed is often helpful when analyzing the problem. Often times an error or message appears in the standard list of the job that was running. Reviewing the standard list can reveal if there was a warning message or an abort. Copying the section of the log that contained the error or warning message and relaying this information to the support staff is also helpful in getting the problem resolved in a timely manner.

Using Available Tools

View Source

If there is a problem regarding the display of information from the Web reports it is recommended that you first view the source of the display. This is done by “right clicking” on the page in question and selecting View Source. A text document will display showing the HTML or XML of the page. Reviewing this information can often times reveal the specific location of the problem.

Special Reports

You should also get used to using the **Diagnostics Report** on the Worksheets tab. Although it might be hard to read at first there is a lot of valuable information contained within as to why your audit turned out the way it did.

You should get used to using the **Student Data Report** to examine what is stored in the Degree Works database for the student.

Webtest

The ***webtest*** command is often very helpful in identifying a problem with security and access to the Degree Works applications. At the host command prompt on the classic server, type *webtest*. You will be prompted to enter a User ID and User Password. The user's key ring can then be displayed. This allows you to view the keys and services that are available to the particular user. The ***webtest*** command also allows you to select a particular service to verify if the user has access. This is accomplished using the service query option that appears at the end of the webtest listing of keys. Also available via webtest is the ability to process or view a script.

Weblogon

The ***weblogon*** command is often very helpful in identifying a problem with security and access to the Degree Works applications. You can run it from the command line or run it from Transit's ADMIN reports option. Simply supply the user's logon ID and their list of keys will appear in the log file for you to examine.

GETXMLAUDIT

Very often while trouble shooting a problem support will request that the XML audit be sent. In order to extract the xml audit form the database ***getxmlaudit*** is used. At the command prompt type *getxmlaudit* along with the audit ID number. This will invoke a script that extracts the audit tree from the dap-audtree-dtl and then will call dapext to create the xml file in the current directory. The file should be transferred to your PC as ASCII.

```
$ getxmlaudit AA000123
```

Version information

Use the ***dgwversion*** script to gather information about what versions of the Degree Works source code is currently in place.

```
$ dgwversions > verions.txt
```

Audit debug

Use the ***dap22dbg*** script to run and tar up lots of good information on the audit for this student.

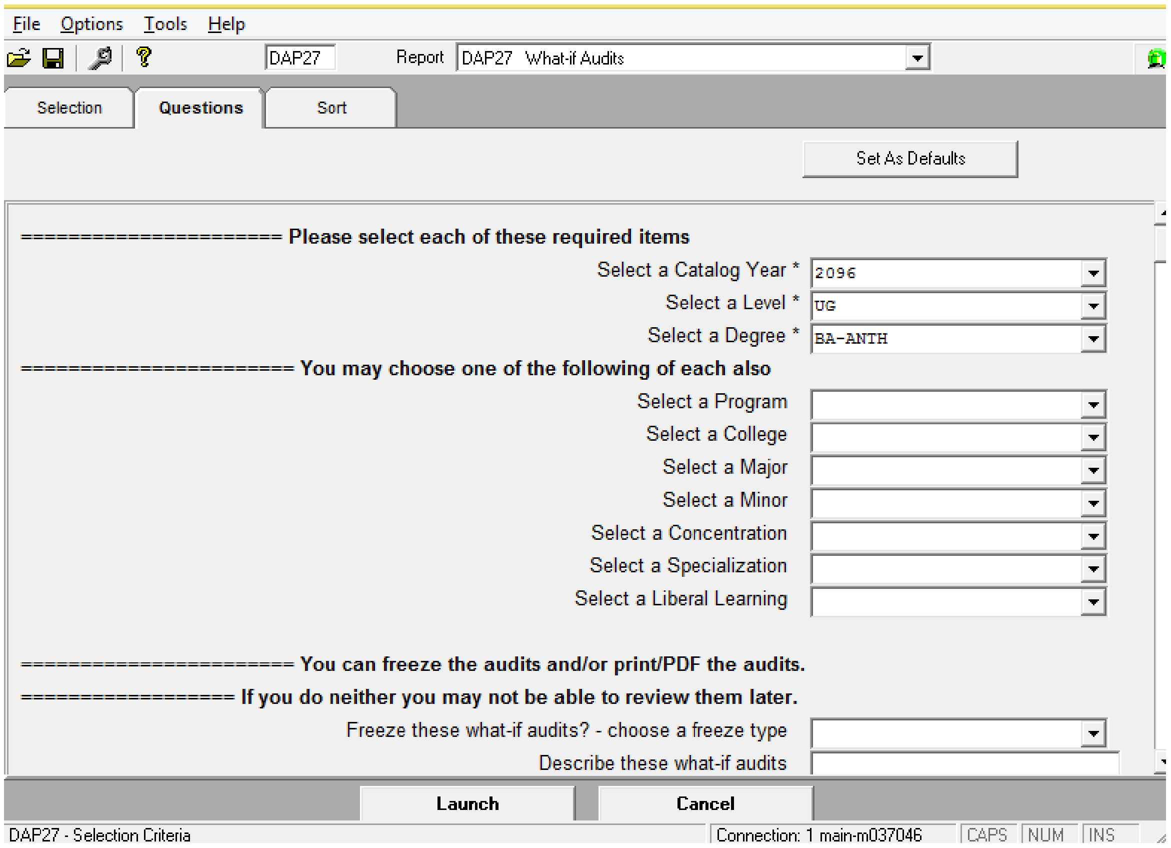
```
$ dap22dbg 123456 mytarfile
```

Send the tar file created to Ellucian for analysis.

You can also run dap22dbg from Transit using the ADMIN report option. Simply supply a student ID and launch the job and a tar file will be created for you to download (as binary) and send to the Action Line on your open case.

Troubleshooting from the PC Applications

Turning on diagnostics mode using Transit tells the classic server software to send process information to a debug file in the logdebug directory. It will be dap10.xxxx.xml with the x's being the process ID of the DAP10 daemon running. The application also creates a file in the Tmp directory on the PC describing each request to and each response from the classic server.



Report: DAP27 What-if Audits

====================== Please select each of these required items

Select a Catalog Year *: 2096

Select a Level *: UG

Select a Degree *: BA-ANTH

======================= You may choose one of the following of each also

Select a Program

Select a College

Select a Major

Select a Minor

Select a Concentration

Select a Specialization

Select a Liberal Learning

======================= You can freeze the audits and/or print/PDF the audits.

================== If you do neither you may not be able to review them later.

Freeze these what-if audits? - choose a freeze type

Describe these what-if audits

DAP27 - Selection Criteria

Connection: 1 main-m037046

The dap10 xml debug file may be examined to determine why you are unable to save your changes or why the database cannot be opened.

In most cases this debug file should be sent to the Ellucian support team. Transfer the file to your PC and attach the file to an email sent to support.

Troubleshooting using the Web Interface

You should review the logdebug/web.log file and use webanalyze to get a summary of the contents of the live web.log file or an older log file.

fixdebug and debugshow

You may have to run the *fixdebug* script to allow the xml file to be used against the Logdebug xsl. Since the web07 process is probably still running the xml file will be missing the end Logdebug and some end Module tags. The *fixdebug* script will add whatever is needed so that it becomes a properly ended xml document.

```
        $ fixdebug web07.1234.xml
```

The debugshow script can be used to pull out the xml for a particular module.

```
          $ debugshow dap41 web07.1234.xml
```

You can redirect the output to a file or let it display on your screen.

Troubleshooting Java Applications

Debug logging can be enabled from the Dashboard Servlet (including Student Planner and Transfer Finder), Scribe, Composer and Transfer Equivalency Self-Service user interfaces. If a user has the DEBUG Shepherd key, they will be able to access the Debug page in the application and enable debug for their session. For additional information on enabling debug, see the *Composer Administration Guide*, Responsive *Dashboard Administration Guide*, *Scribe Administration Guide*, *Student Educational Planner Administration Guide*, *Transfer Equivalency Self-Service Administration Guide*, *Transfer Finder Administration Guide, Transit Administration Guide* or *Web Interface User Guide*.

Backup issues

Backing up the system, especially the data, is a standard procedure performed by the data processing department. It is recommended that the following issues are addressed by any backup procedure:

1. Establish backup schedule, with either partial or full backups
   - Daily
   - Weekly
   - Monthly

2. Verify each backup for completion

3. Develop an archive methodology
   - frequency of recycling (reuse of backup media)
   - location of media (offsite, onsite)

Load Balancing

Load balancing is the technique of creating multiple physical and/or virtual servers that run the same application. It may be done for performance reasons – allowing you to spread the load to different network segments. It may also be easier to respond to changing demand levels by adding or removing servers, thereby optimizing your server resources. It may be done for reliability reasons, allowing failing systems to be removed from service while not disrupting overall responsiveness. Many factors go into architecting a load balanced system, and it is not in the scope of this document to cover them all. Instead, this section discusses the unique aspects of the Degree Works set of applications as they apply to load balancing.

Degree Works began as a monolithic application that was intended to be run on one server. As it has evolved, it incorporated newer technologies that lent themselves to load balancing. It is now possible to load balance most of the Degree Works applications, with some caveats.

The load balancing capabilities of any piece of application code depends on the technology used in its construction. There are three main categories that are useful in this discussion. The first is the applications that run daemon processes on a Unix system. Since this is the oldest code in our suite, we refer to this as our classic applications. The second category are java applications that are deploy in an application container, such as Tomcat or Weblogic. Finally, there are Java applications that run standalone, such as Composer or Transfer Equivalency Self-service. These are all discussed in more detail below.

The nature of the application should also factor into the load balancing design. Administrative applications such as Controller, Scribe and Composer are used by a limited number of users, even at large institutions. It may not be desirable to load balance these at all, depending on the objectives you are trying to meet.

Classic Load Balancing

The classic Degree Works server, which runs the web daemons (started via webstart) use a RabbitMQ queue to communicate with the Dashboard servlet and other applications. As such, it can be load balanced with a few caveats.

There are several configuration and localization files still on the classic server. Any changes to these files must be kept in sync between all the load balanced servers. Examples of these include dwenv.config. Generally, any time you need to modify a file on one of the classic servers, you must make those changes to all other servers as well, or copy the files to all the other servers.

Each server will have its own web.log file, so if you use the webanalyze or webstats tools, you will need to do that on each server, and then combine the results. The webanalyze that you might run from Transit will only report from the server that the Transit Batch Executor is running.

Containerized Java Application Load Balancing

Generally, load balancing Java applications deployed in Tomcat or Weblogic requires no special handling. However, these applications require secure (https) connections, so if you plan to terminate SSL at the load balancer, you must forward the appropriate request headers to our applications so that they know the original request was secure. This includes “x-forwarded-for”, “x-forwarded-by”, and “x-forwarded-proto”. How you configure this depends on the load you are using. Please refer to documentation specific to your technology. On the Tomcat side, you must configure a valve to interpret these headers. Use the following configuration:

```
<Valve className="net.hedtech.degreeworks.tomcat.DwRemoteIpValve"
       remoteIpHeader="x-forwarded-for"
       remoteIpProxiesHeader="x-forwarded-by"
       protocolHeader="x-forwarded-proto"
       protocolHeaderHttpsValue="https(,https)*"/>
```

This uses a special version of the RemoteIpValve that has been created by Ellucian and is based on the Tomcat RemoteIpValve. In order for this valve to work, you must place the Degree Works utilities.jar file in the Tomcat lib directory. This jar file can be found in the classic server app/java directory.

The attributes in the configuration are the same as the Tomcat RemoteIpValve, except that the `protocolHeaderHttpsValue` accepts a regular expression instead of an exact matching string. You may need to configure a value for `internalProxies` depending on your configuration. Refer to your Tomcat configuration guide for an explanation on how to do this.

Some of our Java applications are accessed through our Gateway application (UserAPI). The Gateway serves as a forwarding proxy for the Dashboard, Student Planner, and Transfer Finder applications. When load balancing any of these applications, make sure that there is a Gateway application also running on that server, and point it to the localhost URL for those applications. The load balancer would then point to the Gateway for those applications. See notes below regarding the Gateway application.

Standalone Java Application Load Balancing

Load balancing the standalone Java applications such as Composer and Transfer Equivalency Self-service is rather straightforward, as generally nothing special is required. If your load balancing is terminating the SSL connection, you must add a few additional environment variables to your startup script:

```
export SERVER_USE_FORWARD_HEADERS="true"
export SERVER_TOMCAT_PROTOCOL_HEADER="x-forwarded-proto"
export SERVER_TOMCAT_REMOTE_IP_HEADER="x-forwarded-for"
export SERVER_TOMCAT_REMOTE_IP_PROXIES_HEADER="x-forwarded-by"
```

The Gateway application is itself a proxy application for the Dashboard, and by extension, the Student Planner and Transfer Finder. It should be placed on each server that has these applications. It also requires the above additional environment variables if SSL is to be terminated at the load balancer. Your path URLs to these applications should then reference localhost. Here is an example:

```
export INTERNAL_BASE_DASHBOARD="http://localhost:8080"
export INTERNAL_BASE_PLANNER="http://localhost:8080"
export INTERNAL_BASE_FINDER="http://localhost:8080"
export PATH_DASHBOARD="/dashboard/"
export PATH_PLANNER="/planner/"
export PATH_FINDER="/finder/"
```

This example assumes that the context for the applications is at the root.

System Performance

This section provides information about system performance management, configuration options to manage performance, and troubleshooting guidelines.

Getting Started

The best time to begin thinking about performance is before performance issues are reported. You should become familiar with the tools used to monitor performance and collect performance metrics from your systems at different times of the day and year. Find out how the system performs during periods of light and heavy usage. When issues occur, you can compare current metrics against these baselines and that will help you identify the components that are causing the issue.

Troubleshooting

Define the problem by gathering data about when the issue occurred and the task the user was trying to accomplish when the issue occurred. Usually users think of "poor performance" as "poor response time". Check if that is the case, or if the user meant that a batch job takes too long to run.

Try and determine the programs that were running when the issue occurred. Find out which users report the issue, and if the issue is constant or intermittent. If response time is defined as the time from clicking Run Audit to seeing something returned, then find out that duration. Ask your users to keep a log of the service, ID, time-of-day, and response time whenever the issue occurs.
Ask the user to call you immediately when the issue occurs. You can run some simple diagnostics to gather system-wide information.

If you suspect an issue with the Degree Works software, call Ellucian and send us the information you have gathered. Ellucian will work with you to further analyze the problem and, if needed, will ask you to call your computer hardware vendor.

System Management and Performance

In many ways the performance of Degree Works is linked to your overall system performance. If your computer is overloaded for other tasks, then it may be overloaded for Degree Works processes as well. Ellucian and your computer hardware vendor can help you analyze if you have the right hardware for the tasks you want to perform.

Sufficient disk space and database management often help improve software performance.

Check the available disk space on the system, regularly. Check the degree of fragmentation as well, if you have only small clusters of disk space, you may need to perform some disk space maintenance.

Configuring the Database Server

There are several best practices and techniques that DBAs can use to optimize an Oracle database for best performance. A complete discussion of those is beyond the scope of this document, this document focuses on those items that apply to Degree Works.

The following configuration parameter is the most important configuration parameter specific to Degree Works:

```
cursor_sharing=FORCE
```

If this parameter is not set, system performance will be adversely affected.

The database character set can be US7ASCII, WE8ISO8859P1 or WE8MSWIN1252. **UTF8 is currently not supported.**

Some additional initialization parameters might help with performance of the Degree Works databases, but optimum values will depend on the RDBMS version and database size; consult your Oracle documentation. Oracle is largely self tuning, so care should be taken when setting any non-default values.

The DBA can refer to Oracle performance documentation for more information.

Java Database Pooling Configuration

Java programs obtain a database connection from a pool of open connections. If the pool is too small, then performance may suffer, and the application may even stop working entirely. The default size is adequate for a test account but will need to be increased for production. For those applications deployed into Weblogic, the pools size is controlled on the console page used to set up the database. For Tomcat deployed applications, the pool size is controlled in server.xml where the database resource is configured. The most important value to adjust is the `maxActive` value. The default is 8, but you will want a larger number for production. Start with 100 and adjust after monitoring. More information about these settings can be found by searching the internet for "Apache commons dbcp BasicDatasource settings".

For microservice applications that start via a script (e.g. Gateway), you can adjust this value by exporting the `dw_datasource_maxActive` environment variable. Example

```
export dw_datasource_maxActive=100
```

The same attributes you can change in the Tomcat server.xml file can be set in these startup scripts by exporting environment variables with names beginning with "`dw_datasource_`" plus the attribute name (e.g. `export dw_datasource_maxIdle=50`".

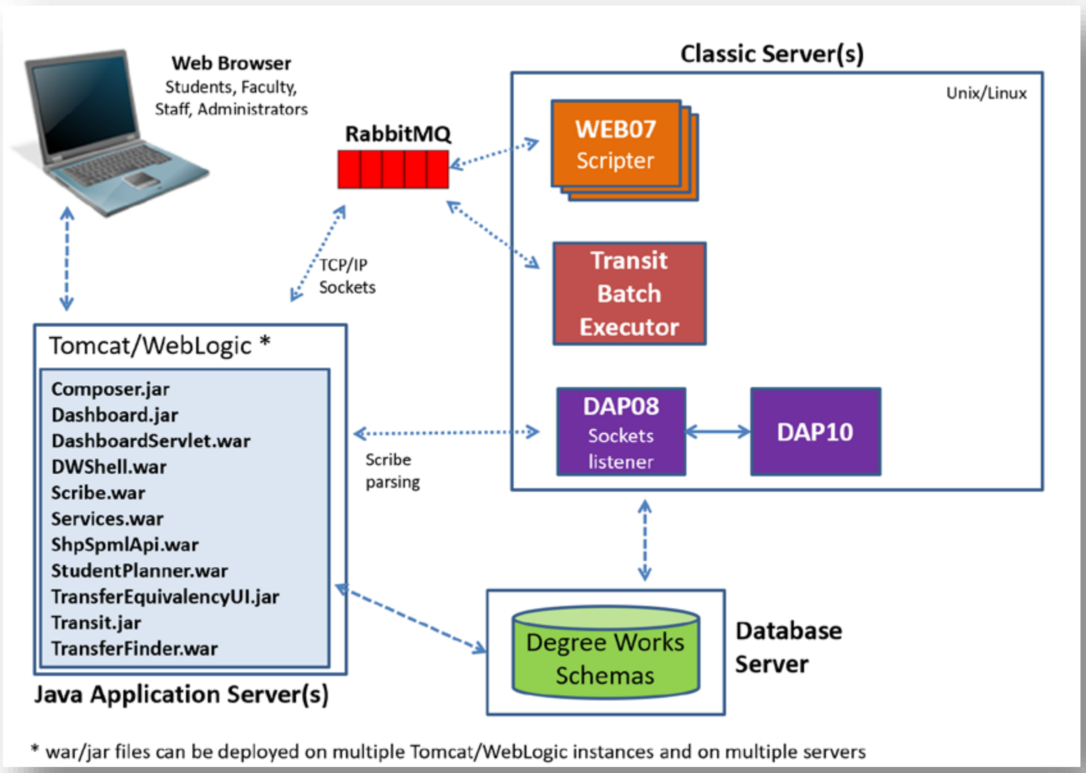
Monitoring the Database Pool

The database pool exposes metrics via the Java Management Extensions (JMX). There are many tools that can monitor these metrics. JConsole is a free application that comes with Java that will display these values. For Tomcat deployed apps, these metrics can be viewed under the MBean Catalina/DataSource/javax.sql.DataSource. For microservice apps, they can be viewed under net.hedtech.degreeworks/JmxBasicDataSource/datasource. There is a getState operation that will report all the relevant values for the datasource.

Classic Web Performance

Most users will use the Classic Degree Works Web interface. This includes the student portal, audit worksheets, and what-if audits, among other things.

Degree Works Classic Web Architecture



Almost all requests coming from the browser go through the dashboard servlet. This servlet passes the request on to the web07 daemon on the classic server via a message queue. Web07 reads the request and validates it. Most of the requests result in the processing of a script (i.e. Web page). In addition, many scripts invoke business logic based on embedded tags. Web07 composes the page and sends the page back to the dashboard servlet, which in turn sends it back to the user's browser.

There are a number of potential choke points in this process. You need to know how to monitor the system and determine where a performance issue originates from.

Useful Tools for Analyzing Web Performance

webanalyze

Webanalyze is primarily a post-mortem tool to check to see how the system performed during the period covered by a `web.log` file. You can run webanalyze against your current `web.log` file or against a saved file. When webanalyze is run against an active `web.log`, it also displays information about the number of web07 daemons that are still running compared to the number you initiated. However, this number should always match since utl01 creates a new web07 process if one dies. The average time for all web07 requests is also displayed, allowing you to see if performance is getting better or worse over the day's activities.

You can e-mail the results of webanalyze, a useful feature when setting it up in cron to run on a daily or hourly basis. A quick check of the webanalyze results in an e-mail is a simple way to monitor system performance.

You can also run webanalyze by navigating to ADMIN in Transit and selecting webanalyze. The webanalyze results are also available on the Web under the **Admin** tab.

webtime

Webtime is a tool that provides information about the number of transactions.You can run webtime against your current `web.log` file or an older, saved `web.log` file.

```
$ webtime
$ webtime myold.web.log
```

The output provides the time it took for each request and the slowest, fastest and average times. See the sections that follow for more information.

webstats

This is a tool to produce statistics on the performance of the web daemons by analyzing the web.log file. Its primary purpose is to produce a data file containing a time series of web server metrics. This includes the periodic measurements of the average transaction duration and maximum transaction count for web07. The comma-delimited file can be input into a spreadsheet or other statistical program for further analysis and graphing.

The tool can be run without any parameters to analyze your web.log file. However, you can specify the file as a parameter if you have saved it under a different name. For example:

```
    webstats web.log.20140301
```

A summary report is printed to the screen:

```
    Web log file: web.log.20140301 (3844198 lines)
    Generated data file: webstats.csv

    Number of web07s started = 50

    SUMMARY:
    Process    Total       Avg   Max
    WEB07    2003274    0.2605    50
```

It displays the number of daemons originally started at the beginning of the report. It then gives, for each of the servers, the total number of transactions processed (Total), the average duration for a transaction (Avg), and the Maximum number of concurrent transaction (Max) count. The Max count is the maximum number of concurrent transactions occurring at the same time that were found for the server. For example, the summary above shows that, at some point covered by the web.log file, there were 235 WEB08 processes processing transactions.

The data file is called webstats.csv, and contains a series of snapshots at regular 1 minute intervals. Each snapshot contains a timestamp, an elapsed time, and for web07, an average duration of its transactions and the maximum concurrent transaction (Max) count in that period. The average and maximum are calculated for a 5 minute period surrounding the snapshot time. For example, the webstats.csv file might contain the following entries:

```
"01:54:00",840,0.148393,50
"01:54:10",850,0.147676,50
"01:54:20",860,0.145578,50
"01:54:30",870,0.144776,50
"01:54:40",880,0.149042,50
```

At 1:54 (row 1, 1st column), 840 seconds since the start of the log (2nd column), the average duration for a web07 transaction was 0.148393 seconds (3rd column) and there was at most 50 web07 daemons running in that time period (4th column), which lasted from 1:54:00 to 1:54:10.

The numbers in the file are moving counts and averages. That means that a record is output every so often, at the value of the interval, by default 1 minute. The average and counts, however, are taken from transactions that occur in a window around that point. The window can be, and by default (5 minutes) is, larger than the interval. So, while we may be outputting records every minute, it will include, by default, the average duration of transactions occurring 2.5 minutes before to 2.5 minutes after that point.

The interval between snapshots can be changed using the --interval option, and the size of the period can be changed with the --window option. Both take parameters in seconds. The name of the data file can be changed with the --datafile option.

```
webstats --interval=120 --window=600 --datafile mydata.txt
```

This would produce a snapshot every 2 minutes, and each snapshot would cover a 10 minute period centered on the snapshot. The first line of the file contains a heading entry.

The starting point for the first entry is calculated to put it on an even multiple of the interval and to include a full window of transactions, and the last entry is also calculated to contain only a full window. This means that there will be some transactions at the beginning and end that will not be included in the data. This is to prevent outlying data points at the beginning and end.

The following figures are examples of some graphs that can be produced from the file:

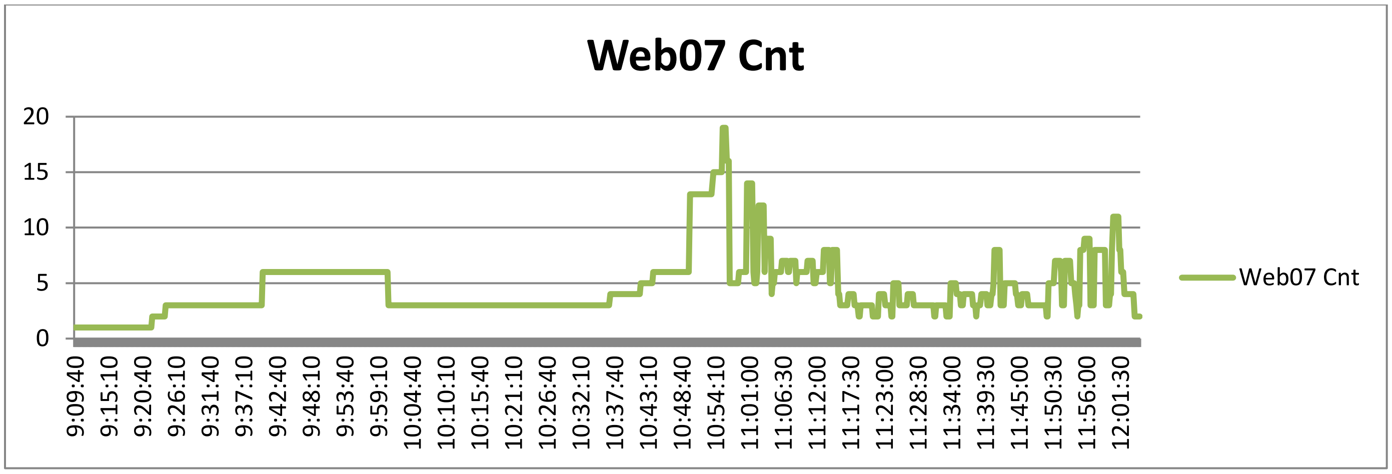


Figure 1 Server Counts by Time

The above graph shows the count values (Cnt heading) from the data file plotted against the timestamp (Time). The graph below shows the average transaction duration (Avg heading) plotted against the timestamp.

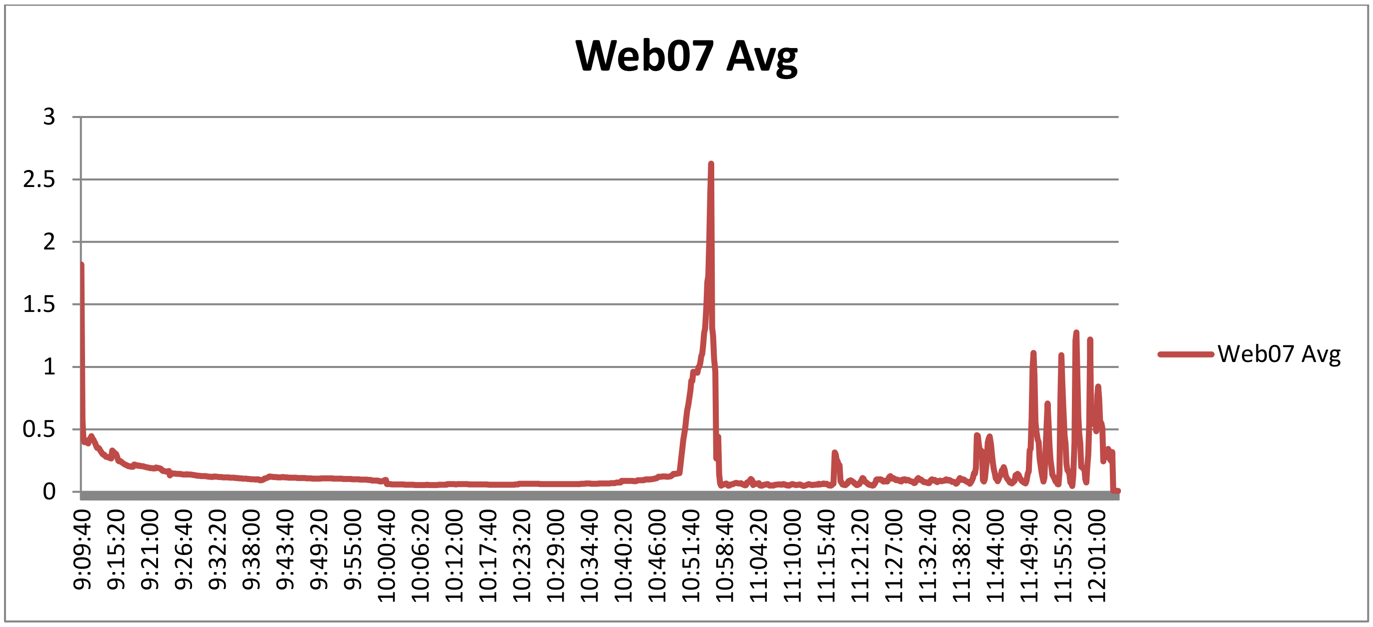


Figure 2 Average Transaction Time by Time

You can see in the examples above that there was a period where web07 transaction time spiked dramatically. You should check the cpu usage during this time. Also look at the counts for web07. If they are approaching the configured maximums then you may want to increase that, so long as you have cpu overhead.

top and vmstat

Top, or a similar tool, is provided on most UNIX systems. It monitors the overall performance of the machine and lists the processes that are using the most CPU resources. Another common

UNIX tool is vmstat, which will also provide the total statistics on CPU usage. When the Classic application server's CPU is fully utilized, configuration changes in the number of Degree Works servers will not improve performance. However, if the CPU is not fully utilized, you may be able to increase performance by increasing one or more of the Degree Works web server daemon counts.

Configuring the Degree Works Web Server Daemons

The primary configurations for the Degree Works daemons are the instance counts for the various daemons. There is only one daemon that can be configured: web07. The number of these is controlled by this environment variables in dwenv.config:

DW_WEB07_COUNT

There is no formula for setting this configuration. Too few servers will choke off responses to requests. Too many will use up system resources (memory, process space).

You can use the webtime script to analyze a web.log script to determine how many requests were handled during the time period covered by the log. There are no exact guidelines for the number of web07 daemons. Try different values for this parameter and check if it has any impact on performance.

If you have under-configured these settings, your response times will suffer, yet neither the Degree Works classic server, the application server, nor the database server will be taxed.

The web performance is significantly affected by a refresh and/or audit. Normally, a student will simply view audits that have previously been calculated, generally during the last student extract batch run. However, there are settings that can be enabled which would cause new refreshes and audits to be generated. These are set in the UCX-CFG020 REFRESH record. Be careful with these settings since they have a very significant effect on performance. For similar reasons, what-if audits can also have an impact on performance.

There are also several operating system parameters that may limit the number of servers that can be run. The `ulimit` command lists the important ones:

```
$ ulimit -aS
time(cpu-seconds)    unlimited
file(blocks)         unlimited
coredump(blocks)     0
data(kbytes)         unlimited
stack(kbytes)        10240
lockedmem(kbytes)    32
memory(kbytes)       unlimited
nofiles(descriptors) 4096
processes            77824
$ ulimit -aH
time(cpu-seconds)    unlimited
file(blocks)         unlimited
coredump(blocks)     unlimited
data(kbytes)         unlimited
stack(kbytes)        unlimited
lockedmem(kbytes)    32
memory(kbytes)       unlimited
```

```
nofiles(descriptors) 4096
processes            77824
```

The first example details the soft limits (`ulimit –aS`), which can be changed for any particular session, and the second example (`ulimit –aH`) details the hard limits, which can only be changed by the system manager. The commands may be slightly different on your system, as may the means to set the limits and the defaults. The limits of particular concern are the `nofiles` and `processes` values. The `nofiles` value limits the number of open files, and the `processes` value limits the number of running processes. As these limits were meant primarily to prevent "runaway" processes from consuming the entire machine, there is little negative impact in making them very large. If they are too small you may see error messages in your log files, such as "Too many open files" or "Too many processes". You may also notice that you do not have as many servers running as you configured. On most systems, a `nofiles` setting of 4096 and `processes` setting of 77824 should be sufficient.

Since Degree Works uses message queues significantly, the operating system (kernel) parameters concerning message queues may also need to be adjusted. The recommended values in the *Degree Works Pre-Installation Checklist* should be sufficient for most systems. If they are too small, you may see messages such as "Out of space on device" or "Failed to write to message queue".

Batch Processing Performance

The primary configuration used to adjust batch performance is the transit.*.workerCount settings. If the batch job is running on the classic server, setting this variable to more than the CPU count can decrease the overall time to completion. You will need to test to determine the optimal setting for your environment for each job.

When running the resstart to build CPA results the DW_DAP25_COUNT variable from dwenv.config must be 1 since only a single dap25 parent can be created. However, the DAP25_NEW_AUDIT_MAX and the DAP25_AUDITS_PER_CHILD controls the maximum number new audits that will be processed together and how many audits will be assigned to each spawned dap25 child process. The number of child processes spawned is based on these two settings. For example, if the max is set to 1,000 and the per-child is 100 then 10 child processes will be spawned; each child will handle 100 audits. You may need to alter these settings if you are finding that the building of CPA results is taking too long.

Another factor that can significantly affect the building of CPA results is the flag settings in the UCX-CFG020 RESULTS record. Each flag set to "Y" will increase the time it takes to process a student. You should only turn on those flags for data that you will be using.

Custom Indexes

When you create your own indexes into the Degree Works database tables you cannot create indexes with names longer than 21 characters as they are not supported.