This is a new edition of the *Customizing Colleague Web API* manual. This edition replaces the previous edition dated July 28, 2014 and is updated for Release IN013553-Colleague Web API 1.6.

**Note:** For Colleague Web API 1.6 to be fully functional, you must also install SU013560-485, SU013559-4.1.0, and SU013589-1805.

### Primary Changes Made

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughout the manual</td>
<td></td>
<td>• Updated the release level for Colleague Web API to version 1.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the release level for Colleague SDK for .NET to version 1.5</td>
</tr>
<tr>
<td><strong>Known Issues</strong></td>
<td></td>
<td>Added issue: Machine Key error on the Colleague Connection Settings page during development</td>
</tr>
</tbody>
</table>
Contents

Getting Started .............................................................................................................. 9
  What this manual covers/How it is organized .............................................................. 9
  Where to find more information .................................................................................. 10

Overview ......................................................................................................................... 11
  Requirements ............................................................................................................... 11
  Getting source code .................................................................................................... 12
  Other resources .......................................................................................................... 12
  Known issues .............................................................................................................. 13

Architecture ..................................................................................................................... 14
  Overview of Colleague Web API architecture ............................................................ 14
  Ellucian-specific architecture ...................................................................................... 15

Best practices ................................................................................................................. 18

Preparing for Development .......................................................................................... 19
  Configuring the Local IIS & Visual Studio ................................................................. 19
  1) Turn on IIS ............................................................................................................ 20
  2) Install Visual Studio ............................................................................................... 22
  3) Install ASP.NET MVC4 .......................................................................................... 22
  4) Install Colleague SDK for .NET version 1.5 ......................................................... 22
  5) Test Visual Studio setup ....................................................................................... 23

Configuring the development server ............................................................................. 28
  Install Colleague Web API using the Installer ......................................................... 28

Suggested source code structure .................................................................................. 29
Setting up source control ........................................................................................ 30
Setting up source control using Git............................................................................. 31

Getting the source code .......................................................................................... 34

Unpacking and preparing the source code ............................................................ 35
PowerShell script....................................................................................................... 35
Updating your ellucian.license file in the solution....................................................... 35
Configure the solution’s connection parameters ....................................................... 35
Unpacking and preparing the source into Git........................................................... 36
Set startup project in Visual Studio............................................................................. 39
Next steps with using Git............................................................................................ 39
   Set up Git on a server .......................................................................................... 40
   Using the Git Visual Studio SCC plugin ............................................................... 41

Making Customizations .......................................................................................... 42

Navigating the solution............................................................................................ 43
API controllers ........................................................................................................... 43
Modules .................................................................................................................... 43
   DTO - Data Transfer Objects ............................................................................. 43
   Domain – Entities and services......................................................................... 44
   Data - DataContracts, transactions, and repositories ..................................... 45
   Coordination – Coordination services ............................................................... 45

Routing .................................................................................................................... 46
Use common URI conventions ................................................................................... 46
RouteConfig.cs ........................................................................................................... 46
Controllers .................................................................................................................. 47
   Determining the route......................................................................................... 47
      RouteConfig entry .......................................................................................... 47

Ellucian.Colleague.Api.Client .................................................................................. 48

High-level view of Colleague Web API ................................................................... 49
How to use the Colleague Data Reader ................................................................. 50

BulkReadRecord<T> .......................................................................................... 53

Example: Providing an array of IDs: ............................................................... 53
Example: Providing a query string: ................................................................. 53
Example: Extracting all rows from the table: ............................................... 53

ReadRecord<T> .................................................................................................. 53

Select ................................................................................................................... 54

How to Use the Colleague Transaction Invoker .................................................. 56

Understanding data repositories ........................................................................ 60

Logging errors .................................................................................................... 61
Caching data ........................................................................................................ 62

Understanding the domain .................................................................................. 64

Working with Data Transfer Objects (DTOs) ...................................................... 65

JSON serialization ............................................................................................ 65
AutoMapper and adapters .................................................................................. 66

Understanding API controllers .......................................................................... 67

Understanding program evaluation ................................................................... 69

ProgramEvaluationService ............................................................................... 69
ProgramEvaluator ........................................................................................... 70
Group ................................................................................................................ 72
Customizing program evaluation ..................................................................... 73
Rules .................................................................................................................. 74

Custom Code Samples ....................................................................................... 75

How to read additional data in a repository ....................................................... 75
How to update additional data in a repository ................................................... 80
How to add new data to a data transfer object .................................................. 85
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>87</td>
</tr>
<tr>
<td>Test-first development</td>
<td>88</td>
</tr>
<tr>
<td>Test data repositories</td>
<td>91</td>
</tr>
<tr>
<td>Mocking</td>
<td>92</td>
</tr>
<tr>
<td>Test harness</td>
<td>94</td>
</tr>
<tr>
<td>Debugging</td>
<td>96</td>
</tr>
<tr>
<td>Debugging during development: Colleague Studio</td>
<td>96</td>
</tr>
<tr>
<td>Debugging during development: Visual Studio</td>
<td>96</td>
</tr>
<tr>
<td>Debugging after deployment</td>
<td>105</td>
</tr>
<tr>
<td>Packaging and Deployment</td>
<td>113</td>
</tr>
<tr>
<td>Overview of packaging and deployment methods</td>
<td>113</td>
</tr>
<tr>
<td>Method 1: Import application using IIS Manager</td>
<td>113</td>
</tr>
<tr>
<td>Method 2: Remote deploy using Web Deploy</td>
<td>114</td>
</tr>
<tr>
<td>Suggested build packaging, build storage, and deployment workflow</td>
<td>114</td>
</tr>
<tr>
<td>Debug builds</td>
<td>117</td>
</tr>
<tr>
<td>Release builds</td>
<td>118</td>
</tr>
<tr>
<td>XML settings transformations setup</td>
<td>118</td>
</tr>
<tr>
<td>Release transforms</td>
<td>121</td>
</tr>
<tr>
<td>Creating the web deployment package using Visual Studio</td>
<td>122</td>
</tr>
<tr>
<td>Debug builds – Visual Studio 2010</td>
<td>122</td>
</tr>
<tr>
<td>Debug builds – Visual Studio 2012</td>
<td>123</td>
</tr>
<tr>
<td>Release builds – Visual Studio 2010</td>
<td>124</td>
</tr>
<tr>
<td>Release builds – Visual Studio 2012</td>
<td>125</td>
</tr>
<tr>
<td>Deploying: Import application method</td>
<td>126</td>
</tr>
<tr>
<td>Deploying: Remote Microsoft deploy method</td>
<td>130</td>
</tr>
</tbody>
</table>
Integrating Your Customizations with a New Release ...................... 132

General process ....................................................................................................... 132
Specific items requiring careful merging ............................................................... 134
Updating the Ellucian branch using Git ............................................................... 135
Integrating your customizations with a new release using Git ................................. 136

Procedures Library ................................................................................................. 142

Installing the InstallShield version of Colleague Web API ........................................ 142
Adding your ellucian.license to the solution ............................................................. 143
Configuring the solution’s Colleague connection parameters ............................................... 146
Shared dependent assembly updates ............................................................................. 149
Getting Started

This manual is intended for developers at your institution who will be making customizations to the Colleague® Web API.

What this manual covers/How it is organized

This manual describes the Colleague Web API design, and tells you how to prepare for, install, configure, and customize the Colleague Web API for your needs.

Table 1 lists the parts in this manual.

Table 1: Parts in this manual

<table>
<thead>
<tr>
<th>Part</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started</td>
<td>Provides overview information including requirements, where to obtain the source code, and related sources of information.</td>
</tr>
<tr>
<td>Architecture</td>
<td>Provides information on the Colleague Web API architecture including Ellucian-specific architecture, best practices, and coding conventions.</td>
</tr>
<tr>
<td>Preparing for Development</td>
<td>Provides detailed information about unpacking the source code, configuring the solution for development, and using source control.</td>
</tr>
<tr>
<td>Making Customizations</td>
<td>Discusses important concepts you need to know when making customizations and includes some recommended “best practices” for making your customizations.</td>
</tr>
<tr>
<td>Testing</td>
<td>Discusses how to unit test code using mocking libraries and test data repositories.</td>
</tr>
<tr>
<td>Debugging</td>
<td>Describes how to debug code changes using techniques such as the unit test repository or a web debugging proxy.</td>
</tr>
<tr>
<td>Packaging and Deployment</td>
<td>Provides guidelines and processes for packaging and deploying your customized version of the Colleague Web API to various web server environments.</td>
</tr>
<tr>
<td>Integrating Your Customizations with a New Release</td>
<td>Provides guidelines and information for how to handle integrating your current customizations with new Ellucian releases.</td>
</tr>
<tr>
<td>Procedure Library</td>
<td>Provides a collection of various procedures that other sections of this manual ask you to perform.</td>
</tr>
</tbody>
</table>
Where to find more information

This manual provides procedures for customizing the Colleague Web API. The following sources of information can provide additional assistance.

Table 2: Additional sources of information

<table>
<thead>
<tr>
<th>Topic</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed information about installing and configuring the Colleague Web API.</td>
<td>The <a href="#">Setting Up Colleague Web API</a> manual.</td>
</tr>
<tr>
<td>Information about installing Colleague software updates, SA Valet, the Colleague daemon, DMI Listeners, and Colleague application server security.</td>
<td>The <a href="#">Installation Procedures for Colleague Release 18</a> manual.</td>
</tr>
<tr>
<td></td>
<td>The <a href="#">Managing Colleague Software Environments</a> manual.</td>
</tr>
</tbody>
</table>
Overview

The Colleague Web API (Application Programming Interface) provides a RESTful, web-based interface for other Ellucian and third-party solutions that need to communicate with the Colleague ERP system.

All of the API services (commonly referred to as endpoints) utilize a consistent communication style, making it easy for external applications to communicate with Colleague. This also greatly decreases the time it takes to build new applications, as those applications can use existing APIs instead of needing to define new ones.

The Colleague Web API is based on REST, an architectural style based on stateless communications and the HTTP protocol. RESTful applications use HTTP requests to post data, read data, and delete data.¹

Requirements

Developer PC requirements

- Microsoft Visual Studio 2010 Professional, Premium, or Ultimate editions with Service Pack 1, or Microsoft Visual Studio 2012 Professional, Premium, or Ultimate editions

  Note: Microsoft Visual Studio Express is not sufficient to perform code-level customization work on the source code for Colleague Web API.

- ASP.NET MVC4 SDK
- .NET Framework 4.0
- Colleague SDK for .NET Version 1.5
- Colleague Studio 2.9 or later
- Visual Studio extensions:
  - NuGet Package Manager
  - Colleague Data Contract Generator (which is installed by Colleague SDK for .NET)
  - Internet Information Services (IIS) 7 or later

¹ For a brief introduction to REST, you can find online tutorials such as Dr. M. Elkstein’s tutorial at http://rest.elkstein.org/.
Recommended

Microsoft Visual Studio Productivity Power Tools

Developer customizing the API needs knowledge of:

- Microsoft Visual Studio
- Object-oriented programming
- C# programming language
- ASP.NET Web API
- Colleague Envision development
- Unit testing using Moq

Getting source code

The source code for the Colleague Web API is available from SA Valet in the form of an installer download package. When you download the source code, SA Valet will download only the source for modules you have licensed. If you are a developer who does not have access to SA Valet, then work with the Colleague Administrator who has access to SA Valet to obtain the source code.

More information on getting and setting up the source code can be found in the Preparing for Development section.

Other resources

For an introduction to ASP.NET MVC, Routing, Controllers, Linq, Moq:

*Pro ASP.NET MVC Framework*, A. Freeman, S. Sanderson, Apress (for MVC 3 or greater)

To understand how to build Colleague Transactions, help is available on the Transaction forms:

Colleague Studio Help

Microsoft tutorials, such as:


An introduction to Domain Driven Design:

Known issues

web.config permissions

When deploying the web deployment package (using either method discussed in the Packaging and Deployment section) Microsoft Web Deploy sets the permissions of the ColleagueApi\web.config file as read-only for all users. This is normally not an issue; however, the configuration settings page accesses the web.config file on saving the Web API settings to initiate an application pool recycle in IIS. On saving, the Web API settings are correctly saved, but you will receive an error stating that access was denied to the web.config file, meaning that you will have to manually recycle the application pool for the new settings to take effect.

**Resolution:** Change the file permissions for the ColleagueApi\web.config file to be “full control” for the Network Service user.

Machine Key error on the Colleague Connection Settings page during development

While debugging/running the Colleague Web API solution locally on a development PC and when deploying to a development test server it is normal to see the following error on the Colleague Connection Settings page of the Colleague Web API.

**Resolution:** This error is show during development because the web.config file within the Ellucian.Colleague.Api project contains a static machine key. The static machine key is necessary during development as the decryption of the Shared Secret, stored in the settings.config file, is dependent on it. A separate error will be displayed if the encrypted Shared Secret cannot be read and that error would need to be corrected before being able to debug; however, the presence of the Machine Key error as shown above will not affect the software during development debugging. Release builds that are loaded on a test or production web server must heed this error and take the corrective action noted in the Setting Up Colleague Web API manual under the “Generate Machine Keys for Security” section.
Overview of Colleague Web API architecture

The Colleague Web API is built using the principles of Domain Driven Design, N-layered architecture, and RESTful API architecture. The API as a whole may also be referred to as the “Business Layer” of an application because it is in the API layer that any business rules are enforced. In response to a request for data, the business layer retrieves data from the database, parses it into domain objects, and then returns the data to the requestor in a serialized form. Several components within this business layer work together to complete the task.

Domain-Driven Design and the Colleague Web API

Domain-Driven Design (DDD) is an approach for designing software where the business domain is complex. One of the tenets of DDD is that the domain model is written in common language that is understandable by all stakeholders.

The N-Layered approach covered in the relevant Microsoft publication is one of many approaches to implementing software using DDD. By using an N-layered architecture, a separation of concerns is maintained. The left side of the Ellucian-Specific Architecture diagram below shows an example of the levels of an API designed using this type of architecture.

- The Service Layer contains web services that deliver the Data Transfer Objects (DTOs) to consumers.
- The Application Layer contains coordinating code and the plumbing for application tasks. This layer should not contain domain logic or execute domain tasks. This layer is responsible for providing granular services, executing workflows for long-running processes, persisting data in a database, and data conversion (DTO Adapters).
- The Domain Layer is responsible for representing business processes and domain rules. This layer separates the domain logic from the infrastructure details.
- The Data Access Layer is responsible for providing data access to entities and persistence when needed.

---

Ellucian-specific architecture

The layers of the Colleague Web API follow a very similar pattern. The diagram below shows how the conceptual API described above maps to the Colleague Web API.

The next graphic shows the mapping of each item in the Visual Studio solution. Each highlighted item below maps to one of the API layers on the right side of the diagram above. You are seeing the Base module below, but there is a set of projects in each module that mirror the Base module. The only difference is that in each module, the project names will contain the module name instead of “Base”. For example, Ellucian.Colleague.Coordination.Student will be the Coordination project in the Student module.

The highlighted items in the figure below map to the Coordination layer in the right side of the diagram above. The highlighted Data project below maps to the Data layer in the right side of the diagram above. The same follows for Domain, Service Client, and Web Service. The items that are not highlighted are related to each of these layers conceptually, and will be explained in this document.
The highlighted items above show where each of these layers resides in the Visual Studio solution. The solution is organized by folders, and within the folders are projects. The section below briefly explains each layer and some of the components embedded within each layer.

**Data**

The *Data* project contains facilities to access the Colleague database specifically, and segregates the database-specific logic away from the rest of the business logic. This layer includes the DataContracts and Transactions, which provide the interface for retrieving data from Colleague. It also includes the Repositories, which contain the logic for reforming the data retrieved from Colleague into Domain Entity objects.

**Domain**

The *domain* contains all of the entities and business logic that defines the behavior of the system. The domain entities define the data types, required fields, and field interdependencies in the system. Domain entities may also have methods that enforce operations on the entity. In some cases, where business rules span multiple entities, domain services are used to implement those rules.
Coordination

The *Coordination* project contains logic to manage the retrieval of data from the repositories and managing the state of domain entities. For example, when data needs to be gathered from multiple repositories to compose a response to an API request, a coordination service is used.

DTOs

A Data Transfer Object (*DTO*) is a lightweight object returned by, or passed into, the web API. DTOs are frequently derived from domain entities, typically as a subset of the related domain entity, and do not have any attached business logic.

Web Service

The *Web Service* defines the endpoints available to external consumers via the components in the Ellucian.Colleague.Api project. The *RouteConfig.cs* and the methods in the Controllers folder respond to received requests.

Service Client

The *service client* provides a .Net C# implementation of a client capable of consuming data from the Colleague Web API. In addition to the *ColleagueApiClient.cs* and *ColleagueApiClient.Base.cs*, there is a *ColleagueApiClient.module.cs* for each licensed module. For example, if you have the Student module licensed, you will see a *ColleagueApiClient.Student.cs* in the list.
**Best practices**

RESTful architecture is followed for all endpoints. Hence, the API endpoints are designed to be predictable, stateless, HTTP-based, and platform and language-independent:

- Operations are self-contained, and each request carries with it all the information that the server needs in order to complete it.

- Requests may be GET, PUT, POST, or DELETE. GET, DELETE, and PUT are assumed to be idempotent requests; executing the same request in sequence results in the same state as a single request. POST is not idempotent; executing the same POST request in sequence is not guaranteed to have the same result.

- If a new endpoint is added to the API, a corresponding entry should also be added to the API client.

**Maintain a robust set of unit tests**

- A test-first protocol should be followed. The developer writes the test before implementing any new feature. Then the code may be added and tests run to confirm that the code added actually addressed the feature. Running the entire unit test repository after each change also verifies that other functionality was not disturbed by the added logic. You can read more about Testing in a later section of this document.

- It is extremely important to have excellent test coverage in the repository, domain, and coordination layers of the API. Many difficulties can be avoided by having a thorough battery of tests that cover every property and method in the domain, and every method in the repository.

**Conduct thorough analysis before modifying DTOs**

- Do not add DTO properties unless absolutely necessary. Delivered DTO properties must not be changed or deleted. Always add new properties to the end of the DTO.

- When considering a modification to DTOs, ask yourself the following questions:

**Q:** Is the data you want to add to the DTO something you want to expose for everyone consuming the endpoint, or is this new data specific to something you are doing?

This will determine whether or not you should add to the existing DTO and affect the existing endpoint, or if you create a new endpoint (and maybe a new DTO) to show this data.

**Q:** Are you trying to change the “contents” put into an existing DTO attribute?

If so, this will affect everyone consuming the DTO and should not be done. Instead, you need to put the new contents into a new attribute of the same DTO or in another DTO.
Preparing for Development

This section describes setup steps that you must perform, and how to access and make source code available for customization. As you prepare for your custom development, first review the following information.

### Preparing Tasks

<table>
<thead>
<tr>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring Local IIS &amp; Visual Studio</td>
</tr>
<tr>
<td>Configuring the Development Server</td>
</tr>
<tr>
<td>Suggested Source Code Structure</td>
</tr>
<tr>
<td>Setting Up Source Control</td>
</tr>
<tr>
<td>Getting the Source Code</td>
</tr>
<tr>
<td>Unpacking and Preparing the Source Code</td>
</tr>
</tbody>
</table>

### Configuring the Local IIS & Visual Studio

Use the steps below to configure the Local IIS and your Visual Studio solution.

1) **Turn on IIS**

2) **Install Visual Studio**

3) **Install ASP.NET MVC4**

4) **Install Colleague SDK for .NET Version 1.5**

5) **Test Visual Studio Setup**
1) Turn on IIS

1. From the Start menu, use the Control Panel, Programs menu to access Programs and Features.

2. Select the Turn Windows features on or off link in the left navigation panel.
3. Under the **Internet Information Services** nodes, make sure the following items (at a minimum) are selected:

- Internet Information Services
- FTP Server
- Web Management Tools
  - IIS 6 Management Compatibility
    - IIS 6 Management Console
    - IIS 6 Scripting Tools
    - IIS 6 MMC Compatibility
    - IIS Metadata and IIS 6 configuration compatibility
  - IIS Management Console
  - IIS Management Scripts and Tools
  - IIS Management Service
- World Wide Web Services
  - Application Development Features
    - .NET Extensibility
    - ASP
    - ASP.NET
    - CGI
    - ISAPI Extensions
    - ISAPI Filters
    - Server-Side Includes
  - Common HTTP Features
    - Default Document
    - Directory Browsing
    - HTTP Errors
    - HTTP Redirection
    - Static Content
    - WebDAV Publishing
  - Health and Diagnostics
    - Custom Logging
    - HTTP Logging
    - Logging Tools
    - ODBC Logging
    - Request Monitor
    - Tracing
  - Performance Features
    - Dynamic Content Compression
    - Static Content Compression
  - Security
    - Basic Authentication
    - Client Certificate Mapping Authentication
    - Digest Authentication
    - IIS Client Certificate Mapping Authentication
    - IP Security
    - Request Filtering
    - URL Authorization
    - Windows Authentication

4. Select **OK**.

5. Restart your computer so that the settings take effect.
2) Install Visual Studio

You may use Visual Studio 2010 Service Pack 1 (SP1) Professional, Premium, or Ultimate editions, or Visual Studio 2012 Professional, Premium, or Ultimate editions.

If you are using an installation of Visual Studio 2010 that has not yet been upgraded to SP1, you can download the service pack at the following link:


3) Install ASP.NET MVC4

ASP.NET MVC 4 is a framework for building scalable, standards-based web applications using well-established design patterns, and utilizing ASP.NET and the .NET Framework.

Be sure that you have installed APS.NET MVC4. The download page is located at the following link:

http://www.asp.net/mvc/mvc4

4) Install Colleague SDK for .NET version 1.5

Colleague Software Development Kit (SDK) for .NET makes it easier to build .NET applications and services that interact with Colleague. Using the SDK, you will be able to build .NET solutions that can retrieve data from Colleague and run Colleague Transactions. The SDK includes a Visual Studio Extension: the Colleague Data Contract Generator, which is a tool for generating data contracts required for interactions with Colleague. This extension currently supports Visual Studio 2010, 2012, and 2013. Code samples are also included.

Colleague SDK for .NET is available using SA Valet.
5) Test Visual Studio setup

To test your PC setup for Visual Studio, create a test MVC 4 project.

2. The first New Project dialog box from the Create New project wizard is displayed.

- Select the **ASP.NET MVC 4 Web Application** template.
- Make sure **.NET Framework 4** is selected.
- Under **Other Project types**, expand either Visual Basic or Visual C#, and then select **Web**.
- Under **Visual Studio installed templates**, select **ASP.NET MVC 4 Web Application**.
- In the **Name** field, enter **TestMVC4Application**.
- In the **Location** field, enter a name for the project folder.
- Press **OK**.
3. The second New Project dialog box is displayed.
   - Make sure **Internet Application** is highlighted.
   - The View engine field should be set to **Razor**.
   - Press **OK**.
4. Right-click on the new project (*TestMvc4Application*). Select **Properties**, and then choose the **Web** tab to view the server properties.

- By default the **Use Visual Studio Development Server** option is selected. Change this to the **Use Local IIS Web Server** option.
- Make a note of the project URL that defaults. Save this change.
- You will be asked to configure the virtual directory in IIS. Select **Yes**. This will add the project into IIS as a DefaultWebSite.
5. Build and run the project.

- Press **F6** to build the project.
- Press **Ctrl+F5** to run the project. This will open the project in a browser. If the project launches, then the test is successful.
Configuring the development server

Install Colleague Web API using the Installer

Later portions of this section and future sections (Packaging and Deployment) will require a web server installation of the binary version of the software (using the installer) on your development web server.

By using the Colleague Web API installer to create the website on your development server, you are:

- Provisioning all of the required IIS components.
- Providing a deployment target for your customized solution.
- Creating the license file needed by the runtime software, which you will copy into your custom solution in a future step of this section.

In the Procedures Library section, see Installing the Installer Version of Colleague Web API for all of the necessary setup steps.

This process will also be used when you deploy your customized solution to test and eventually to production, which will be discussed further in the Packaging and Deployment section of this guide.
**Suggested source code structure**

The following directory structure should be used with the solution’s source code. This structure helps to separate the solution source code and non-solution items, such as your builds of the solution and source control files. This structure also takes into account future updates of the software that will require the solution source to be separate from other items. The base directory (highlighted below) in this structure would be the base directory when you add this to source control.

> **Note:** Create the base directory as close to the root of the drive as possible to help avoid exceeding the maximum path and filename lengths allowed in Windows. The suggested structure below is similar to what Ellucian developers use on their development PCs, so the maximum path and filename lengths should not be a problem when using this structure.

```
Ellucian.Colleague.Api
  builds
  source
```

The Ellucian.Colleague.Api directory will serve as the base directory for all source and any other items related to your customization efforts.

The `source` directory will be the destination to which the source code will be unpacked later in this section.

The `builds` directory is used later when packaging and deploying the customized solution to keep track of the builds that have been deployed to your environments.
Setting up source control

Ellucian strongly recommends that you use a source control system while making and maintaining your customizations of this solution. There are hundreds of source files included in this solution, and keeping track of which files your institution adds or modifies is much easier when tracking those changes in a source control system. Whether your institution has one developer or ten, a source control system is imperative to your success and the quality of your customized solution. Furthermore, as Ellucian releases new versions of the source code, source control will aid you as you merge your customizations with Ellucian changes and enhancements.

At Ellucian, most of the .NET development teams use Microsoft Team Foundation Server (TFS) as the source control system. TFS tightly couples with Visual Studio, but also has associated costs for the software itself. While TFS is best for working with Visual Studio solutions, there are several other popular source control systems that work well and have various levels of integration with Visual Studio. This guide will provide basic instructions on setting up Git⁴, an open source distributed version control system with a large community of users and support.

Because it is distributed, Git allows you to create a repository on your development PC first, and then move the repository to a server-hosted repository, which is a best practice for source control. Git repositories are directories on disk, so making a backup is as easy as backing up a directory. This ease of use helps you to get the source code into source control with little effort in setting up source control. This is one of the core reasons Ellucian recommends the use of Git. However, you can use any source control system you like or already have in place.

☞ Note: Please take the time to learn about and set up a source control system. It will benefit you and your fellow developers while making customizations, and will aid Ellucian support and development staff if you need assistance with the customized solution. Knowing what has changed is an important first step in troubleshooting an issue in the software.

---

⁴ More information on Git can be found at [www.git-scm.com](http://www.git-scm.com).
Setting up source control using Git

Use the following steps to install Git on your development PC and to create the initial repository, which will be populated with the source code in one of the upcoming sections.

All developers must install Git for Windows on their PC and configure some basic Git properties prior to using Git.

1. Install Git by downloading it from the Git website (www.git-scm.com). Stick with the recommended settings as you run the Git InstallShield. Git is primarily a set of commands, but the Git InstallShield for Windows will also install some convenient tools that make using Git on Windows much easier.

2. Configure your username and email address. Each time a developer commits a change to Git, the change is tagged with the user and email address. Once Git is installed, open Git Bash and use the following commands to setup your username and email:
   
   ```
   $ git config --global user.name "Your name"
   $ git config --global user.email your_email@school.edu
   ```

The developer unpacking the source code will need to perform the following steps.

1. Create a base directory for all Git repositories on the root drive, such as C:\Git.

2. Open Git Bash.

   **Note:** You can right-click on the Git directory you created in Step 1 and select **Git Bash**. The Git shell extensions were installed as a part of the Git installation.

   Windows experts: Git bash is a Unix-style bash shell; it does a good job of converting Windows paths to UNIX paths, but all commands are UNIX commands, not Windows commands. For example, you need to use `ls` instead of `dir` to list a directory's contents.
3. Make sure the current location is set to the base directory you created in Step 1 (the status line should show the path: “/C/Git”). Use the `cd` command to change to that directory if necessary.

4. Initialize a new repository by typing the following command:
   $ git init Ellucian.Colleague.Api

   This directory is created if it did not already exist.

5. Clone the Github gitignore repository to your local repo home. By default, Git will include every file, and you do not want that. Git uses a file named `.gitignore` in the root of the repository to specify what to ignore. Instead of creating that manually, you can obtain a version maintained by the Git community via Github.

   To clone the gitignore Github repository to your local machine, type the following command:
   $ git clone https://github.com/github/gitignore.git

   This creates a directory at the path C:\Git\gitingnore that will contain various `.gitignore` files.

6. Now that you have the `.gitignore` files, you need to copy the correct one to the repository created in Step 4.

   To copy the VisualStudio.gitignore file to your repository and rename it `.gitignore`, type the following:
   cp ./gitignore/VisualStudio.gitignore ./Ellucian.Colleague.Api/.gitignore
7. The .gitignore file you copied in the previous step is configured to ignore debugging symbols (.pdb files). Ellucian suggests that you check these in, as they aid in debugging the solution. Modify the following line of the .gitignore file that you copied to the repository using a text editor by commenting out the "*.pdb" line (line # 28 at time of writing):

Change: "*.pdb" to "#*.pdb"

8. Now, back in the Git Bash window, move into repository directory:

$ cd ./Ellucian.Colleague.Api

9. You are now finished with the basic structure of the repository. It now needs to be committed as a change. Type the following commands to add all of the repository's files to Git's staging area and commit them as a change:

$ git add .
$ git commit -m "Initial repository setup"

At this point, you have set up the initial structure of the repository and have configured which files (via the .gitignore file) that the repository will track. The next step is to obtain the source code and add it to this repository. There are also other useful Git topics including setting up a Git server, moving the repository you just created to that server, and other topics that will make working with Git easier.
Getting the source code

1. Download the source code zip files using SA Valet 2.8 or later. For more information, see the “Viewing and Downloading Installer Releases” section of the Updating Colleague Software manual.

2. Save the files on the development PC where the source code will be unpacked (see the next section).

3. The items pulled down from SA Valet will be contained in a version number folder (such as 1.6.0.0) inside a folder called Colleague Web API.

   From this folder, you can see the installation .exe file, a PowerShell script file for unpacking and preparing source code, a readme file, and the zip folders related to the modules you have purchased. If, for instance, you did not purchase Student Planning, then you would not see the Ellucian.Colleague.Api_Planning.zip file.

4. Do not attempt to directly unzip any items in this archive. Instructions for unpacking and preparing the solution are described below.
Unpacking and preparing the source code

Note: If you are using Git, skip to the Unpacking and Preparing the Source into Git topic in this section.

PowerShell script

The source code download from SA Valet includes a PowerShell script (EllucianColleagueWebSourceSetup.ps1) that will correctly unpack the zip archives and create a working Visual Studio solution. The instructions for using and running the script are contained in the README.txt included in the same directory as the script.

Follow the steps in the README.txt file for unpacking and preparing the source code.

Updating your ellucian.license file in the solution

After you have unpacked and created a working solution using the provided PowerShell script, you will need to copy your version of the ellucian.license file into the solution. In the Procedures Library section, see Adding Your ellucian.license to the Solution for all of the necessary setup steps.

Configure the solution’s connection parameters

Configure the solution’s connection parameters by completing the steps listed in the Procedure Library section: Configuring the Solution’s Colleague Connection Parameters. These settings allow the solution to connect to the Colleague Development environment while debugging the solution using Visual Studio.
Unpacking and preparing the source into Git

Using the following steps to unpack, prepare, and add the source code to a Git repository. Only one developer needs to perform these steps.

1. Open Git Bash and move to the **Ellucian.Colleague.Api** repository.

   ```bash
   $ git branch ellucian
   $ git checkout ellucian
   ``

2. The source code will be added to this repository using a new branch. Branches allow you copy the current branch to isolate your work from the main codebase of the repository and will be used in your day-to-day development activities.

   To create a new branch named **ellucian** and move to it, type the following commands:

   ```bash
   $ git branch ellucian
   $ git checkout ellucian
   ```

3. If not already present, create a subdirectory named **source**.

   ```bash
   $ mkdir source
   $$
   ``

4. You are now ready to extract the source code from the zip archives using the included PowerShell script. Read the README.txt for full details on running the PowerShell script, making sure that you specify the source directory created in Step 3 above as the destination directory for the script (in Step 2d of the README.txt).

   ```bash
   PS \{source code directory from SA Valet\}>
   \EllucianColleagueApiSourceSetup.ps1 -DestinationDirectory "C:\Git\Ellucian.Colleague.Api\source"
   ```
5. After Step 4 completes, add your `ellucian.license` file to the solution. In the *Procedures Library* section, see *Adding Your ellucian.license to the Solution* for the necessary setup steps. This procedure will ask you to open the solution, which is located at: C:\Git\Ellucian.Colleague.Api\source\Ellucian.Colleague.Api\

![Image of file structure]

6. Build the solution to verify that it builds correctly by selecting **Build Solution** from the **Build** menu.

7. Configure the solution’s connection parameters by completing the steps listed in the *Procedure Library* section: *Configuring the Solution’s Colleague Connection Parameters*. These settings will be committed to source control so that they do not have to be entered every time a developer accesses the solution.

8. Run the tests in the solution to verify that all of the tests run successfully. If they do not, double-check that the `ellucian.license` file is in the correct directory under the Ellucian.Colleague.Api.Tests project as described above.

9. Close Visual Studio, being sure to save all changes.
10. Return to Git Bash and commit the solution code and ellucian.license changes by running the following commands. The last command is particularly important, because tagging the commit provides a quick way of identifying the unmodified solution source code prior to your customizations.

$ git add .
$ git commit -m "Ellucian Release 1.6 and updated license file"
$ git tag v1.6

11. Merge the ellucian branch back into the master branch using the following commands:

$ git checkout master
$ git merge ellucian

12. As a final step, to prepare your local repository for development work, create a development branch using the following command:

$ git branch dev

You now have a working solution that you can begin making customizations to. As a next step, you will likely want to move your repository to a web server so that the repository can be accessed by other developers, and also setup some additional functionality that will make working with Git easier. Read the Next Steps with Using Git section for these and other topics related to using Git.
Set startup project in Visual Studio

All developers will need to set the Ellucian.Colleague.Api project as the startup project in Visual Studio in order to run and debug the solution locally. Right-click on the Ellucian.Colleague.Api project (within the Web Service solution folder) and choose Set as StartUp Project.

Next steps with using Git

One of the best resources for help and information on Git is the Pro Git book (written by Scott Chacon and published by Apress). This book is available online (published under the Creative Commons Attribution Non Commercial Share Alike 3.0 license) at http://git-scm.com/book.

See the topics below for information that will make working with Git more useful.
Set up Git on a server

Ellucian strongly encourages you to set up Git on a server that is regularly backed up. This will also be necessary if your institution will have more than one developer working on customizations. Details on setting up Git on a server can be found in the ProGit book (the first four topics in Chapter 4), available at the following link:


After installing and configuring Git on the server, use the following steps to create and push your local Git repository to the server. Only one developer (the developer who unpacked the source code) needs to complete this task.

1. On the server, create a base directory for your server repositories such as C:\Git (which will be used in remaining steps below). If you choose a different directory or location, substitute it in the steps below.

2. Open Git Bash and ensure that your current working directory is the base repositories directory.

3. Create a new bare repository by typing the following command:

   $ git init –bare Ellucian.Colleague.Api

4. Return to your development PC and open Git bash, making sure the Ellucian.Colleague.Api repository is the current working directory and the current branch is master.

5. Define the server repository as a remote repository by typing the following command:

   $ git remote add origin “remote repo path and name”

Where the “remote repo path and name” is the protocol path used to access the repository that was setup on the server. In the example below, the protocol used is simply a file share of the Git directory on the server mapped to the ‘S’ drive. This will differ, for example, if you used a different protocol, such as SSH.
6. Push the local master branch to the remote repository by typing the following command:

```
$ git push origin master
```

You have now cloned your local repository to the server. Other developers can clone the server repository to their local development PC and begin working. New developers should clone the repository, create and checkout a development branch (dev), and begin working.

**Using the Git Visual Studio SCC plugin**

Visual Studio 2010 does not offer a Microsoft-developed source control provider plugin for Git (Visual Studio 2012 does). A third-party Git source control provider plugin/extension has been created and works well, providing many of the common source control tasks (such as committing and viewing an item’s history) directly in Visual Studio.

To install, open the Visual Studio Extension Manager (**Tools > Extension Manager**) and search the Online Gallery for “Git”. Select **Git Source Control Provider** from the list.

The plugin also includes and sets up a merge tool.
Making Customizations

This section discusses important concepts you need to know when making customizations and includes some recommended “best practices” for making your customizations.

<table>
<thead>
<tr>
<th>Section Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigating the Solution</td>
</tr>
<tr>
<td>Routing</td>
</tr>
<tr>
<td>Web API Under the Hood</td>
</tr>
<tr>
<td>How to Use the Colleague Data Reader</td>
</tr>
<tr>
<td>How to use the Colleague Transaction Invoker</td>
</tr>
<tr>
<td>Understanding Data Repositories</td>
</tr>
<tr>
<td>Understanding the Domain</td>
</tr>
<tr>
<td>Working with Data Transfer Objects (DTOs)</td>
</tr>
<tr>
<td>Understanding API Controllers</td>
</tr>
<tr>
<td>Understanding Program Evaluation</td>
</tr>
</tbody>
</table>
Navigating the solution

API controllers

The controllers handle the HTTP requests (GET, PUT, POST, DELETE). For example, the endpoint http://localhost/Ellucian.Colleague.Api/buildings will use the GetBuildings action method in the BuildingsController.

All the API controllers can be found in project Ellucian.Colleague.Api in folder Controllers. This folder contains subfolders by module area.

Modules

The Modules folder of the Ellucian.Colleague.Api solution separates API functionality by module areas. Depending on your licensing, you may have a number of subfolders (such as Base and Student) with the Modules folder. Each module contains projects for DTOs, Domain layer, Data layer, and Coordination services.

DTO - Data Transfer Objects

The data transfer objects are the objects that are serialized and sent in response to http requests. DTOs are deliberately simple and typically consist only of simple properties and no logic.
Figure 1: Example of a DTO Object using the Grade DTO

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Ellucian.Colleague.Dtos.Student.Grade
{
    /// <summary>
    /// An institutionally-defined course grade
    /// </summary>
    /// Unique ID for this grade
    public string Id { get; set; }
    // Decimal equivalent of this grade
    public decimal? GradeValue { get; set; }
    // Id of the grade scheme in which this grade is included
    public string GradeSchemeCode { get; set; }
    // Description of this grade
    public string Description { get; set; }
}
```

Domain – Entities and services

The domain defines entities and domain services to handle all business logic surrounding the application. In many cases, domain objects are mapped one-to-one to a data transfer object.

Domain services are found in Services folders within the same domain projects. An example of a domain service is the Program Evaluator (Planning), which is used to run program evaluations for students.

Repository Interfaces also exist in the domain layer. These interfaces expose the repository action methods that are to be implemented by the repositories (in the Data layer).
Data - DataContracts, transactions, and repositories

All objects involved in accessing and updating data in the Colleague database reside in the data projects. This includes the data contracts used by the Colleague Data Reader, the Colleague Transactions, and the data repositories used to take data contract information and create domain objects.

- DataContracts are automatically generated by the Colleague Data Contract Generator Visual Studio Extension and are used to read data from Colleague.
- Colleague Transactions are created within Colleague Studio and can be used to perform any action in the Colleague database (including updating files, executing existing Colleague subroutines, etc.)
- Repositories use Colleague Web Services to interact with the Colleague database. DataContracts are used to retrieve data when a simple query can be used to identify the needed data. Transactions are used for database retrieval that requires more complex logic and for database updates. The repository then uses the retrieved data to build domain objects. Many Colleague tables can be accessed in a single repository if needed to build a domain object or aggregate.

Coordination – Coordination services

The coordination services are the services and interfaces used to provide a link between the controllers that are fulfilling the request and the repositories that access the database. Controllers can access repositories directly, but often will utilize a service when the logic needed to fulfill a request is anything more than direct single repository access, but can also use a service for more complex data aggregation. For example, coordination services contain the logic that consolidates the repository access needed when the object returned comprises an aggregation of domain objects, and the building of the DTO where needed. In a simpler sense, coordination services are used when two or more repositories need to be accessed to construct the resulting DTO.
Routing

ASP.NET MVC routing is an important aspect of the Colleague Web API infrastructure. Routing is the mechanism used to map a URI request to an API controller action which fulfills the request. This topic will discuss some of Ellucian’s API conventions and talk about the major components involved in routing.

Use common URI conventions

The following are a few of the key guidelines Ellucian follows when defining API URIs:

- Use transparent (readable) URIs.
- Use dashes (aka hyphens) in lieu of camel case.
- Support nested resources when doing so is logical and only two or three levels deep.
- The use of plural resource names is preferred.

RouteConfig.cs

RouteConfig contains the complete list of the Colleague Web API endpoints. By design, there is no default route. Therefore, all routes must be defined explicitly. There is an entry for every possible API request. Requests that do not match one of the patterns defined by the route configuration are rejected with a 404 error. A route configuration entry cites the name of the controller and the method in the controller that should be called when a request is received for a particular endpoint and version.

Each route entry defines a MapHttpRoute object and must include the following:

- Name: A unique name; cannot be used for any other route.
- routeTemplate: Defines the logical route, including inline URL arguments.
- defaults: Identifies the Controller, the method within the controller, and query arguments.
- constraints: Specify constraints that must be met in order for the routing framework to use this route to fulfill a request.
Controllers

The Controllers folder within the Ellucian.Colleague.Api project provides the support for each Colleague Web API endpoint to which a request is directed by the RouteConfig. You will have the shared controllers, the base controllers, and all of the controllers under each module that you have licensed, such as Finance, Student and Planning, which can be seen by expanding the folder for each module.

The controllers do not have a lot of logic, but are responsible for invoking the services or calling a repository (if very simple) to get the data needed to fulfill the API request, then sends the DTO (data transfer object) or defined response back to the requestor.

Determining the route

The route that is displayed on the API help page is determined by a combination of the controller method arguments and the RouteConfig definition.

Example

The controller method arguments define the data that is expected from the request:

```csharp
/// <summary>
/// Criteria supplies keyword, requirement, or various filters which may be used to search
/// and narrow a list of courses.
/// </summary>
/// <param name="criteria">Course search criteria</param>
/// <param name="pageSize">integer page size</param>
/// <param name="pageIndex">integer page index</param>
/// <returns>A CoursePage of courses matching criteria with totals and filter information</returns>
public CoursePage PostSearch([FromBody]CourseSearchCriteria criteria, int pageSize, int pageIndex)
```

RouteConfig entry

The RouteConfig.cs entry provides the “template” that enables the API to identify a request when it comes in, and specifies with the “defaults” argument the controller and the method (aka Controller Action) that handles this request.

```csharp
routes.MapHttpRoute(
    name: "CourseSearch",
    routeTemplate: "courses/search",
    defaults: new { controller = "Courses", action = "PostSearch" },
    constraints: new
    {
        httpMethod = new HttpMethodConstraint("POST"),
        headerVersion = new HeaderVersionConstraint(1, true)
    }
);
```
Ellucian.Colleague.Api.Client

This project contains a service client implementation that can be used by other .NET applications to access the Colleague Web API functions. There is a service client method for each Colleague Web API endpoint. The service client methods are grouped by module, but you will get only the modules for which you are licensed.

- ColleagueApiClient.Base.cs
- ColleagueApiClient.cs
- ColleagueApiClient.Finance.cs
- ColleagueApiClient.Planning.cs
- ColleagueApiClient.Student.cs

ColleagueApiClient.cs defines constants for the basic paths and versions, and the version, session service methods for the version, user and session endpoints. It is an extension of the BaseServiceClient, defined in the Ellucian.Web.Http project, which provides base utility methods used to build any of the various HTTP requests (for example GET, PUT, POST).

All changes to the ColleagueApiClient require that an updated assembly be provided to the .NET solution using the API client.
High-level view of Colleague Web API

The following diagram provides a high-level view of how the different layers of the Colleague Web API communicate with each other to fulfill a request.

The next topics will discuss each layer, starting with the database access and working up to the API controllers.
How to use the Colleague Data Reader

The Colleague Data Reader uses entity data contracts to read data from the Colleague database. Entity data contracts are created and updated using the Colleague Data Contract Generator. The data contract class (.cs file) produced by the Colleague Data Contract Generator should never be added or updated directly within Visual Studio.

When you add or update an entity data contract, you specify the Colleague entity and specific data elements you want to include in the contract. See the example below that shows adding a new data contract to the DataContracts folder by right-clicking on the DataContracts folder.
To generate a new Entity data contract, select **Generate Entity Data Contract to Folder**. CTX Data Contracts invoke transactions within Colleague. They are covered in the next section.

Entity data contracts can be updated to add additional data attributes without compromising the existing implementation of the software. Generally, the best practice when adding new fields is to update the existing data contract to add the new field instead of creating different entity data contracts for the same entity and using different suffixes. Removing attributes is not advised.
The figure below shows an example of updating an existing data contract by right-clicking on the Courses data contract.

The work of populating the domain objects from the entity data contracts resides in the data repository methods.

When reading data from Colleague, the Colleague Data Reader provides three important methods:
**BulkReadRecord<T>**

*BulkReadRecord* extracts multiple rows of a Colleague table at once. This returns a collection of entity data contracts `<T>`. Where possible, all necessary data should be bulk read (rather than record by record) and then, once all needed data has been gathered, the domain entities can be created. It is much more costly to read data record by record.

BulkReadRecord has three important overload methods. You can specify one of the following:

- An array of IDs desired.
- Selection criteria.
- Extract all rows from the table.

**Example: Providing an array of IDs:**

```csharp
var ids = sectionData.Where(s => s.SecXlist.Length > 0).Select(s => s.SecXlist).Distinct();
Collection<CourseSecXlists> crossListData = dataReader.BulkReadRecord<CourseSecXlists>(ids.ToArray());
```

**Example: Providing a query string:**

```csharp
var query = "WAIT.COURSE.SECTION NE '' AND WAIT.TERM EQ " + queryQuotedTermIds + "'";
Collection<WaitList> waitlistBulkData = dataReader.BulkReadRecord<WaitList>(query);
```

**Example: Extracting all rows from the table:**

```csharp
Collection<Grades> gradesData = dataReader.BulkReadRecord<Grades>("");
```

**ReadRecord<T>**

*ReadRecord* reads a specific item from a Colleague table by record Id. The result is an entity data contract object.

**Example**

```csharp
DataContracts.Person record = dataReader.ReadRecord<DataContracts.Person>(id);
```
Select

Select selects records from a table that match specific criteria. A string[] containing the Ids selected is the result of the search.

Example

```csharp
string searchString = "DP.STUDENT.ID EQ '' + students.Recordkey + '';
string[] studentPlans = dataReader.Select("DEGREE_PLAN", searchString);
```

The Colleague Data Reader ensures that the person making the request for Colleague Data is an authorized user; however, certain types of public data can be read from Colleague as a “guest” or anonymous user using the anonymous version of the data reader.

The Web Services Parameters (WSPD) form in Colleague allows each client to control the data that they want to be “designated” as publicly available data. Entries in the Public Entity Name column are “public” as long as the associated column “Force Private” is set to No.

![WSPD - Web Services Parameters](image)

An un-authenticated request for publically accessible data requires using the anonymous data reader in the API repository—anonymousDataReader instead of dataReader. However, the simplest way to accomplish this is to use

```csharp
this.DataReader
```

which will use the IsAnonymous property to identify which data reader to invoke. For example, to bulk read information from the IMPORTANT.NUMBERS table (which may be defined as public data), you could use the following statement:
Collection<ImportantNumbers> impData =
this.DataReader.BulkReadRecord<ImportantNumbers>(string.Empty);

If this.DataReader.IsAnonymous is true, anonymousDataReader will be invoked, if this.DataReader.IsAnonymous is false, dataReader will be invoked and the user making the request must be authenticated.

If the user is not authenticated and the anonymousDataReader is unable to read the data because the file is not defined on the WSPD form, it will throw an exception.

Attributes on the API controllers or controller actions may be used to determine which requests must be authenticated and which allow anonymous requests.

Use the [Authorize] attribute when the action or actions must be authenticated.

Use the [AllowAnonymous] attribute when anonymous requests are permitted, or to allow an anonymous access to an action within a class.

```csharp
namespace Ellucian.Colleague.Api.Controllers
{
    /// <summary>
    /// Provides access to get and update Degree Plans.
    /// </summary>
    [Authorize]
    [LicenseProvider(typeof(EllucianLicenseProvider))]
    [EllucianLicenseModule(ModuleConstants.Planning)]
    public class DegreePlansController : BaseCompressedApiController
    {
    }
```

Use the [AllowAnonymous] attribute when anonymous requests are permitted, or to allow an anonymous access to an action within a class.
How to Use the Colleague Transaction Invoker

The Colleague Transaction Invoker uses Colleague Transaction (CTX) data contracts to run Colleague Transactions on the application server. Colleague Transactions are the primary way to update data in Colleague or to run existing subroutines in Colleague. CTX data contracts are generated from Colleague Transactions that were delivered by Ellucian, or that you create in Colleague Studio.

A Colleague Transaction must first exist in the Colleague environment before you can generate a CTX data contract. To generate a new CTX data contract, select Generate CTX Data Contract to Folder.
Changes to Colleague Transactions require changes to the transaction within Colleague Studio. If the transaction arguments are affected by the changes, then those changes also require an update to the CTX data contract using the CTX Data Contract generator.

Colleague Transactions are executed using the Colleague Transaction Invoker. The Request object of the transaction is provided, and a response object is returned. For example, rules are defined in Colleague to determine if students should be permitted to view their grades. A Colleague Transaction was created to evaluate the rules for a specified student ID. This Colleague Transaction can be executed from the data repositories by doing the following:

```java
GetGradeViewRestrictionsRequest gradeViewRequest = new GetGradeViewRestrictionsRequest();
gradeViewRequest.PersonId = id;
GetGradeViewRestrictionsResponse gradeViewResponse = transactionInvoker.Execute<GetGradeViewRestrictionsRequest, GetGradeViewRestrictionsResponse>(gradeViewRequest);
```

This transaction response returns a GetGradeViewRestrictionsResponse object containing attributes IsRestricted and Reasons (when IsRestricted is true).

Colleague Transactions are also used to make updates to the Colleague database, either by calling existing Colleague processes, or by doing data updates within the Colleague Transaction. The Colleague DataReader reads data, but does not update data. For example, the Student Repository uses the Colleague Transaction RegisterForSections, which in turns calls the standard registration process in Colleague to register the student for the desired course sections.
Degree Plan Repository method UpdateDegreePlan uses the transactionInvoker to execute the UpdateDegreePlan transaction. The Colleague Transaction UpdateDegreePlan\(^5\) makes use of a version number for each row in the DEGREE_PLAN table. If the version number in the database agrees with the version number on the update request, then the Colleague Transaction takes care of locking the record and updating the entire degree plan based on the information in the UpdateDegreePlanRequest. Errors in performing the update are returned in the UpdateDegreePlanResponse and can be verified by the Degree Plan Repository.

### Custom Colleague transactions

You may want to customize a Colleague Transaction used by the Colleague Web API. Because Colleague Transactions are Colleague Studio resources, you will need to follow the same techniques outlined in the *Managing Custom Software* manual. Just as with other types of resources, the method you choose to manage your customization will depend on the type of change you are making.

#### Minor Colleague transaction customizations

When making a minor change to a Colleague Transaction (one that does not affect the input or output arguments associated to the Colleague Transaction), you may want to change the existing version of the Colleague Transaction. Making this type of change may not affect the .NET code using this Colleague Transaction data contract.

For example, if a Colleague Transaction supplies a preferred name output argument and you are only changing how the preferred name is constructed, then change the Colleague Transaction within Colleague Studio and manage this custom change as you would other custom changes. There are, of course, minor changes to the Colleague Transaction that would impact the Web API .NET code, and you must make those modifications accordingly.

#### Extensive Colleague transaction customizations

When making more extensive changes to a Colleague Transaction, follow the steps outlined in Option 1 in the *Managing Custom Software* manual, and make a cloned version of the Colleague Transaction.\(^6\) When using this option, be sure the CTX’s alias is also changed to follow your naming conventions, and leave the version number of the CTX "as-is." It will be the same as the original CTX the clone was made from.

---

\(^5\) Named UPDATE.DEGREE.PLAN in Colleague Studio.

\(^6\) With the cloned option, you will have to address merge issues with future Ellucian releases of the Colleague Web API source code.
When cloning a Colleague Transaction, you will need to follow the additional steps in the .NET code in Visual Studio.

1. Generate the new transaction’s data contract, specifying the alias of the cloned CTX.

2. Find and replace any references to the original CTX’s data contract with the new data contract (and adjust code appropriately to function with the modified process). You can choose which places to change, or change them all, depending on your customization goal.
Understanding data repositories

Repositories are bridges between data and operations that are in different domains. In this case, it is the bridge between the Colleague database and the Colleague Web API using the entity data contracts and Colleague Transaction data contracts.

The Colleague Web API repositories extend the BaseColleagueRepository class which implements IColleagueDataReader (DataReader) and IColleagueTransactionInvoker (transactionInvoker) for data access, and ILogger to log error messages.

Repository interfaces are used to explicitly define the repository functionality that can be implemented. For example, the student module CourseRepository has a variety of action methods, but ICourseRepository exposes only the Get function and overload methods, which are the only public methods in the repository. Therefore, the API controllers and coordination services can only use these repository methods.
Logging errors

When building the domain objects from the entity data contracts, it is important to properly handle exceptions that can occur. For example, if the domain entity building requires a description, but the data returned in a building data contract has a null or empty description, then the domain will throw an exception and the building entity cannot be created. Because Colleague also requires a description to create a building code in Colleague, this situation means bad data exists in the database. Generally it is a good practice to "log" the occurrence of this so that bad data situations can be found and resolved.

The GradeRepository's method BuildGrades is a good example of how to log error messages. In the Get() method, all of the grades are gathered from Colleague using dataReader.BulkReadRecord. These data contracts are then assembled into valid Grade entities within the BuildGrades Method. For each grade contract, it tries to create a grade entity and if it fails, the catch uses ILogger to log the error. In this case, the valid grades are returned, the errors are logged, and no exception is thrown.

```csharp
private ICollection<Grade> BuildGrades(collection<Grades> grades)
{
    var grades = new List<Grade>();
    // If no data passed in, return a null collection
    if (grades == null)
    {
        foreach (var grd in grades)
        {
            try
            {
                var grade = new Grade(grd.RecordKey, grd.Grade, grd.GradeLegend, grd.GradeScheme);
                grade.GradeValue = grd.GradeValue ?? 0;
                grades.Add(grade);
            }
            catch (Exception ex)
            {
                var gradeError = "Grade record corrupt";
                var formattedCourse = objectFormatter.FormattedXml(grd);
                var errorMessage = string.Format("(0)" + Environment.NewLine + "(1)", gradeError, formattedCourse);

                // Log the original exception and a serialized version of the course record
                logger.Log(System.Diagnostics.DiagnosticsEventEnvironment.Error, ex);
                logger.Log(System.Diagnostics.DiagnosticsEventEnvironment.Information, errorMessage);

                //throw new ArgumentException("Error occurred when trying to build grade "+ grd.RecordKey);
            }
        }
    }
    return grades;
}
```
Caching data

Another important functionality available in the data repositories is the ability to cache relatively static data for a period of time and thus save on the expensive trip to pull data from the database. The Colleague Web API makes use of caching for data such as validation code tables, code files, and parameter records, but also caches data such as courses and faculty. For this type of data, the cache time out value is set to 24 hours, meaning it is only read from the database once each day.

In the case of Course Section information, the repository provides methods to retrieve cached section information (which is only updated every 24 hours) and different methods to provide “fresh” section information. Accessing “fresh” section information allows the consumer to receive up-to-date information regarding seat counts and waitlist numbers, but does incur the additional overhead associated with retrieving data from the database. Which repository method is used is determined by the header information in the API request. If the request header has CacheControl.NoCache set to true, then “fresh” data is returned. The default is to return cached section information.

Typically, data is read from the repository, formed into Domain Entities, and stored in that form in the repository cache. When another request needs the same data, if the data stored in the cache has not expired, it is retrieved from the cache. If the cache data has expired, the data is retrieved freshly from the database.

ICacheProvider implements such functions as Get, Add, Remove, and Contains. However, the infrastructure provides two methods to simplify using cache: GetOrAddToCache and AddOrUpdateCache.

GetOrAddToCache first checks for the existence of a cache and, if present, uses that; otherwise, it builds the data and caches it.

The following shows the Get method in the GradesRepository as an example of using the GetOrAddToCache method.

```csharp
public ICollection<Grade> Get()
{
    var grades = GetOrAddToCache<ICollection<Grade>>("AllGrades",
    () =>
    {
        ICollection<GradesData> gradesData = dataReader.BulkReadRecord<GradesData>("GRADES", "");
        var gradeList = BuildGrades(gradesData);
        return gradeList;
    });
    return grades;
}
```

In this example, the process first looks to see if “AllGrades” is already in cache. If so, this cached data is returned in the grades variable. If “AllGrades” is not already stored in cache, then the grades are pulled from the database using the dataReader.BulkReadRecord and these data contracts are transformed into grade entities in the domain via BuildGrades. This data is added to cache and is then returned.
AddOrUpdateCache is used to when you want to add a new item to cache, or if it already exists in cache, replace it with the updated value.

ContainsKey and .Get are also sometimes used, for example in the SectionRepository:

```csharp
string cacheKey = BuildFullCacheKey(AllRegistrationSectionsCache);
if (ContainsKey(cacheKey))
{
    IDictionary<string, Section> regSectionsDict = (IDictionary<string, Section>)localCache.Get(cacheKey);
    if (regSectionsDict != null)
    {
        foreach (var sectionId in sectionIds)
        {
            // Note: The two methods ContainsKey and Get require the fully built cache key, whereas the GetOrAddToCache and AddOrUpdateCache methods require only the original ID or name of the item written to cache.
```
Understanding the domain

The domain is the “heart” of the business rules of the API. The Domain classes define the form of the objects that are built when data is read from and updated into the repository, and are the objects that contain data used to build the DTOs that compose the responses to API requests.

Entities that comprise the domain are determined as a result of extensive domain-driven design. The result of domain-driven design is a domain of entities that make sense for the application. The entities defined in the domain are a refactoring of the entities that exist on the original database, reformed and aggregated in a way that makes sense for the application that uses them.

While entities comprise the majority of what is found in the domain, there may also be domain services that are designed to provide business logic that cannot be confined to a particular entity. A domain service embeds significant business rules in a meaningful function. The ProgramEvaluator (in the planning module) is an excellent example of this concept, because program evaluation is a significant function of the application, and operates across several domain entities.

The domain also owns the repository interfaces. The repository interfaces define the methods that the API expects each repository to provide. If a repository does not implement its interfaces, the application will not build. If for some reason another database is accessed, the changes are isolated to only the repository project; all other parts of the API can rest assured that the same call to a repository method will yield the same expected results.

The domain entities do not actually store data. The domain entities define only the form that data takes when it is transferred between the service and the repository and back again in response to a request.
Working with Data Transfer Objects (DTOs)

Data Transfer Objects are the objects that are serialized and sent in response to HTTP requests. The goal in building DTOs is to keep the objects lightweight, reusable, and concise. Adding attributes to existing DTOs and adding new DTOs to the Web API has no impact on any application using the API; however, changing and obsoleting elements can have serious consequences for consumers of the API. It is therefore a best practice not to add anything to a DTO unless it is completely necessary. If you want to add a new attribute to the course DTO, be sure to consider if you are willing to support the delivery of that data indefinitely.

When an API endpoint is tasked to return a course section (the Section DTO), the attributes it returns should be only the data the consumers of the API endpoint need. In the case of a section, this is defined as information such as the title, the number of credits, the start date, etc. DTOs are also kept lightweight by supplying IDs or codes for related data instead of nested objects. In the case of the Section DTO, a CourseId attribute is provided. If the application that uses this data needs more information about the course, it can use a different endpoint to request that data.

**Tech Tip:** All new DTOs and changes to existing DTOs will require that updated assemblies be provided to any .NET applications that are using those DTO objects to deserialize the response data, such as the Colleague Self Service application. In the Procedure Library, see the *Shared Dependent Assembly Updates* section for more information.

JSON serialization

The Colleague Web API uses the Newtonsoft.Json serializer to transform objects to JSON and then back again. For selected types of data, it is necessary to specifically define the Newtonsoft.Json.Converter to use. For example, enumerations are converted by applying the attribute `StringEnumConverter`.

```csharp
namespace Ellucian.Colleague.Dtos.Student
{
    /// <summary>
    /// Denotes whether a class is being taken for a grade, for pass/fail, or audit only.
    /// </summary>
    [JsonConverter(typeof(StringEnumConverter))]
    public enum GradingType
    {
        Graded,
        PassFail,
        Audit
    }
}
```
AutoMapper and adapters

The creation of DTOs is often a simple transformation of a domain entity. To assist with this transformation AutoMapper is used. AutoMapper uses a convention-based matching algorithm to match up source-to-destination values, so that logic does not need to be written to move data between an entity object and a DTO object. Complete documentation on AutoMapper can be found at:

https://github.com/AutoMapper/AutoMapper/wiki

Where additional mapping dependencies or custom mappings have been used, adapters have been created in the Coordination projects to assist with the mapping between DTOs and entities (or vice versa). For example, the faculty DTO carries a phone DTO within it, so the following faculty adapter was created to handle that situation in the mapping.

```csharp
namespace Ellucian.Colleague.Coordination.Student.Adapters
{
    public class FacultyEntityAdapter : AutoMapperAdapter<Dtos.Student.Faculty, Domain.Student.Entities.Faculty> {
        public FacultyEntityAdapter(IAdapterRegistry adapterRegistry, ILogger logger) {
            // Mapping dependency
        }
    }
}
```
Understanding API controllers

The API controllers handle the HTTP requests (GET, PUT, POST, DELETE). The Colleague API controllers extend the BaseCompressedApiController class which contains CreateHttpResponseException and CreateNotFoundException and extend the standard MVC4 class APIController.

All API controllers reside in the project Colleague.Api.Client in the folder “Controllers”. The Controllers folder is divided into subfolders for each module area (for example, Base and Student). The action methods within the controller are responsible for fulfilling the endpoint requests.

For basic requests to get data, the controller actions are as simple as asking the appropriate repository for the needed entities, using adapters to map the domain entities to the DTOs, and then returning the DTOs. By convention, the names of the controllers are the plural form of the entity followed by the word Controller – such as BuildingsController. For example, the endpoint:

http://hostname:port/ColleagueApi/buildings

uses the GetBuildings action method in the BuildingsController. Because building information is not added, updated, or deleted as part of the Colleague Web API services, this is the only action method in the BuildingsController.

The DegreePlansController contains the actions to Get a degree plan, Post a new degree plan, Put an updated degree plan, and action methods to apply a sample degree plan and register the sections for a degree plan term.
### Endpoint

<table>
<thead>
<tr>
<th>Endpoint</th>
<th>Controller Action Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET <a href="http://hostname:port/ColleagueApi/degree-plans/%7Bid%7D">http://hostname:port/ColleagueApi/degree-plans/{id}</a></td>
<td>Get</td>
</tr>
<tr>
<td>POST <a href="http://hostname:port/ColleagueApi/degree-plans">http://hostname:port/ColleagueApi/degree-plans</a></td>
<td>Post</td>
</tr>
<tr>
<td>PUT <a href="http://hostname:port/ColleagueApi/degree-plans/apply-sample">http://hostname:port/ColleagueApi/degree-plans/apply-sample</a></td>
<td>PutApplySample</td>
</tr>
<tr>
<td>PUT <a href="http://hostname:port/ColleagueApi/degreeplans/%7BdegreePlanId%7D/">http://hostname:port/ColleagueApi/degreeplans/{degreePlanId}/</a></td>
<td>PutRegistration</td>
</tr>
<tr>
<td>registration?termId={termId}</td>
<td></td>
</tr>
<tr>
<td>PUT <a href="http://hostname:port/ColleagueApi/degreeplans/%7BdegreePlanId%7D/">http://hostname:port/ColleagueApi/degreeplans/{degreePlanId}/</a></td>
<td>PutSectionRegistration</td>
</tr>
<tr>
<td>section-registration</td>
<td></td>
</tr>
</tbody>
</table>

An important responsibility of the API controller is to return the appropriate HTTP result codes in response to the requests. The normal HTTP code 200 (OK) is returned for successful requests, but it is also important to return the correct error message. Here are some examples of the http codes returned by the Web API.

<table>
<thead>
<tr>
<th>Condition</th>
<th>HTTP Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>200</td>
</tr>
<tr>
<td>Item not found</td>
<td>404 – Not Found</td>
</tr>
<tr>
<td>Version Number mismatch (unable to update)</td>
<td>409 – Conflict</td>
</tr>
<tr>
<td>Permission problem</td>
<td>403 – Forbidden</td>
</tr>
</tbody>
</table>

You can find the complete list of the Colleague API endpoints by accessing the API help page at the following link:

`http://hostname:port/ColleagueApi`
Understanding program evaluation

The program evaluation service is a coordination service. It does not use or touch any of the logic or data used by Colleague’s Evaluate Student Program (EVAL) and Batch Academic Evaluation (BEVL) processes. Customizing the EVAL process has no effect on the outcomes reflected on the My Progress tab in Colleague Self Service (likewise, customizations to Self-Service will not be reflected in Colleague EVAL). That must be customized separately. Prerequisite evaluation uses some of the lower-level code from this process, as described below.

At a high level, requirements are a three-level tree, with Requirements at the top, Subrequirements beneath, and groups at the bottom. Requirements and Subrequirements have relatively few specific properties of their own; they are mostly just collections of groups. The features of the requirement definition language in Colleague, such as “TAKE 2 COURSES FROM MATH*101, MATH 102; MIN_GRADE A” — most of those points are stored in, and evaluated at, the group level. Requirements and Subrequirements mainly demand that the student complete $x$ of $y$ lower-level units.

It would be useful to note that the requirement language entered on the Subrequirement Specifications (SRSP) form is not stored anywhere. It is parsed and converted into a series of data values and stored in the Colleague file ACAD.REQMT.BLOCKS. When you re-enter SRSP, the language is regenerated from those values. The structure in the Requirement, Subrequirement, and Group objects are very similar to the values stored in the Colleague ACAD.REQMT.BLOCKS file. Most of the requirements data is in that file, with ACRB.TYPE values of “10” for Requirements, “20” for Subrequirements, and “30” through “34” for Groups, where the last digit is specific to the language used in the group. Other requirement-level data can be found in the ACAD.REQMTS file.

Programs can have requirements. In addition, students may have customized programs that have their own requirements. Those are blended together for evaluation.

Here are the major parts of the evaluation process:

ProgramEvaluationService

ProgramEvaluationService gathers program requirements, student data, student program data, and custom program requirements. It gathers all of the possible Colleague rules from the requirement tree and compares all planned courses and credits to them. It creates a ProgramEvaluator and passes the collected data to it. After the program is evaluated, the service is returned a ProgramEvaluation. The service then calls its own OptimizeEval() function, which analyzes the data to see if any credits or planned courses were used to satisfy requirements that were no longer needed. If so, it marks those parts of the requirement tree as “to be skipped” and runs the Evaluator again.


**ProgramEvaluator**

It is important to note that the ProgramEvaluator has two main constructors – one for evaluation of requirements within the context of a program (My Progress) and the other for simple evaluation of requirements (Prerequisites.)

Another interesting feature is that the constructor must do a “deep copy” of the requirements tree. Evaluating the requirements against the student includes processing of what are called “exceptions” in Colleague and called RequirementModifications in the API. RequirementModifications may, as the name implies, actually change the requirement tree for that particular student. A complete separate copy is needed to keep the student’s evaluation from modifying the original requirement trees in the repository.

Then the credits and courses to be evaluated are copied into a structure called MasterResults in the ConstructResultSet() method. This is where courses and credits are sorted for priority. If a custom sort order was used previously, this method would be where you should customize that. Otherwise the code follows the default order used by EVAL:

The Ellucian-supplied DEFAULT sort type sorts in the following way:

Category 1: In-house, graded courses

Category 2: In-house, ungraded courses which are nonetheless considered complete (they have no grade scheme, so they are not expected to ever get a grade)

Category 3: Equivalencies (transfer and non-course)

Category 4: In-progress courses

Category 5: Pre-registered courses

Within each category, items are sorted by start date, earliest date first. Note that the term “courses” as used above actually refers to ANY STUDENT.ACAD.CRED record, even those which record only credits, and not a specific course, such as when a lump sum of credits is transferred in from another institution.

After the copied tree is created, any modifications the student has are applied to it. Then in the Evaluate() function, a ProgramEvaluation object is created to hold the results. It is roughly comparable to the ProgramRequirements object. It exists at that level, and holds a tree of lower-level results that parallels the requirement tree down to the group level.

---

7 “Exception” is a reserved word in C# and in most object-oriented programming languages.
Next, the requirement tree is “unrolled” down to the group level. Groups have different orders of evaluation based on whether they are part of a major, minor, or other kind of requirement. They are also affected by the language of the requirement. All other things being equal, “Take All” type requirements are evaluated before “Take Selected”:

- `TakeAll`, // old EVAL type 30
- `TakeSelected`, // old EVAL type 31
- `TakeCredits`, // old EVAL type 32
- `TakeCourses`, // old EVAL type 33
- `CustomMatch` // old EVAL type 34

This is to prevent a requirement that says, “TAKE 3 COURSES FROM SUBJECT MATH” from taking MATH*101 and preventing a more specific requirement that says “TAKE MATH*101” from getting it. Custom match is not supported in the new Evaluation, but the data can hold one, so the enumeration has a slot for it.

CUSTOM.MATCH: Unfortunately, this could not have been supported in the new system. Fortunately, the newer system is much easier to customize.

Finally, the groups are all lined up and the planned courses and credits are lined up. The evaluation proper can begin. For each group the procedure is:

1. Check to see if this is a subsequent run and the optimizer wants this group skipped.
2. Check to see if it is in a requirement or subrequirement that is already satisfied.
3. Check which credits are available to be used against this group – eliminate any already used against a requirement type that does not share credit with this requirement type.
4. Call the group level evaluator (detailed below).

The group level evaluator returns a GroupResult that contains information about what courses/credits were evaluated against this group, which were accepted or rejected, and why. It is important to note that the courses and credits are passed to the group in the form of AcadResults, which can contain either a course or a credit, and the metadata regarding what its eventual disposition was.

After all the groups are evaluated, then all of the subrequirements are evaluated to see if the requisite number of groups were completed and if they made GPA limits. After all of the subrequirements are evaluated, then all of the requirements are evaluated to see if the requisite number of subrequirements were completed and if they made GPA limits. Then the ProgramEvaluator examines the data to see if the requirements were satisfied by completed credit or planned courses, and sets the CompletionStatus and PlanningStatus indicators.

Then the Evaluator is finished and passes the completed ProgramEvaluation back – either to be optimized and rerun, or passed back out of the coordination service to the calling program.
Group

The Evaluate() method in the Group class contains specific data about each group's requirements.

Here are some examples of how the syntax used on the Subrequirement Specifications (SRSP) form appears in a Group:

<table>
<thead>
<tr>
<th>How Syntax Used on SRSP Appears in a Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAKE MATH*101</td>
</tr>
<tr>
<td>Courses = “id1”</td>
</tr>
<tr>
<td>TAKE 3 CREDITS FROM DEPT MATH</td>
</tr>
<tr>
<td>MinCredits = 3</td>
</tr>
<tr>
<td>FromDepartments = {</td>
</tr>
<tr>
<td>“MATH” }</td>
</tr>
<tr>
<td>TAKE 1 COURSE FROM MATH<em>101, MATH</em>102</td>
</tr>
<tr>
<td>MinCourses = 1</td>
</tr>
<tr>
<td>FromCourses = { “id1”, “id2” }</td>
</tr>
<tr>
<td>TAKE 2 COURSES FROM LEVELS 100,200 NOT</td>
</tr>
<tr>
<td>FROM DEPT CHEM</td>
</tr>
<tr>
<td>MinCourses = 1</td>
</tr>
<tr>
<td>FromLevels = { “100”, “200” }</td>
</tr>
<tr>
<td>ButNotDepartments = { “CHEM” }</td>
</tr>
</tbody>
</table>

There are too many data points to list. Check the code in the Group.cs file itself, it is well-commented. If needed, check the RequirementRepository to see what specific data items map to which item in the domain, but again, you’ll find most of that in the comments.

Once in Evaluate, the group is checked to see if it was waived or replaced by RequirementModification. Then it makes two major loops.

The first loop checks each credit/course to see if it is affected by overrides, and then calls CheckCredit() to see if it applies to the group at all. CheckCredit() runs any applicable rules (from the group up to the AcademicEligibilityRules for the program.) It then checks to see if the course/credit is on the Courses or FromCourses lists, and if it is from the right department, subject, or level, if specified in the group. If it passes all of these tests, it is marked as “related” to the group.

The second loop takes all the “related” credits and courses, and attempts to apply them to the group, one at a time, until the group is satisfied, or the function runs out of credits and planned courses to test. In this loop all of the ceilings are checked; maximum x credits per rule y, maximum z courses at level a, etc. Finally, if the course or credit passes all of those, the final hurdle is that it has to apply toward an unmet minimum, such as the minimum number of courses, credits, credits in department x, etc. If it does, the credit is then marked “applied.”
Customizing program evaluation

The main reason the Colleague CUSTOM.MATCH feature cannot be implemented is that the custom subroutine it calls has access to the intermediate metadata of the Evaluate Student Program (EVAL) process. That metadata does not exist in this process, because the old EVAL code is not being executed. However, it does exist in a form. There are several helper functions in the GroupResult class that allow the coder to see what has been applied to that group so far. Here is an example of the Group.Evaluate() function using them:

```csharp
private bool MinSubjectsNotMet(GroupResult gr)
{
}

private bool MinDepartmentsNotMet(GroupResult gr)
{
    return MinDepartments.HasValue && gr.GetAppliedDepartments().Count() < MinDepartments.Value;
}

The GetApplied() function is of particular use. It is used frequently to get all of the AcadResults applied to the current group so far. Often that is put into an IEnumerable list with the current acad result to see if the combination of the currently considered credit and the previously applied credits would violate some maximum.

Calculate GPA example:

Another place you might customize this process is in how GPA is calculated. The calculation is straightforward:

```csharp
/// <summary>
/// Returns the GPA of the credits applied to this group
/// </summary>

public decimal Gpa
{
    get
    {
        return (GetAppliedGpaCredits() == 0) ? 0 : GetAppliedGradePoints() / GetAppliedGpaCredits();
    }
}

/// <summary>
/// Returns the GPA of the institutional credits applied to this group
/// </summary>

public decimal GetAppliedInstGpa()
{
    return (GetAppliedInstGpaCredits() == 0) ? 0 : GetAppliedInstGradePoints() / GetAppliedInstGpaCredits();
}
```
The grade points and credits are taken from Colleague in this way:

```csharp
// "Term" (actual) credit values
public decimal AttemptedCredit { get; set; } // STC.ATT.CRED
public decimal CompletedCredit { get; set; } // STC.CMPL.CRED
public decimal Credit { get; set; } // STC.CRED
public decimal ContinuingEducationUnits { get; set; } // STC.CEUS
public decimal GradePoints { get; set; } // STC.GRADE.PTS
public decimal GpaCredit { get; set; } // STC.GPA.CRED

// Cumulative credit values (accounts for repeats)
public decimal AltCumAttemptedCredit { get; set; } // STC.ALT.CUM.ATT.CRED
public decimal AltCumCompletedCredit { get; set; } // STC.ALT.CUM.CMPL.CRED
public decimal AltCumCredit { get; set; } // STC.ALT.CUM.CRED
public decimal AltCumGpaCredit { get; set; } // STC.ALT.CUM.GPA.CRED
public decimal AltCumGradePoints { get; set; } // STC.ALT.CUM.GRADE.PTS
```

The easiest way to customize how the GPA was calculated would be to alter the `AcademicCreditRepository` class and the `StudentAcadCred` data contract to pull in your preferred values, and then change the GPA functions in the `GroupResult` class to use them.

**Rules**

Rules in the Web API are used primarily in program evaluation. Some rules are evaluated through an engine written in the Web API. Those that fall outside of the supported features in the Web API are evaluated by calling rules processing in Colleague. It is worthwhile to support rules as much as possible in the API, because evaluation is quicker than making a call to Colleague. The classes `AcademicCreditRuleAdapter.cs` and `CourseRuleAdapter.cs` would likely need to be changed to accommodate additional rule support for program evaluation. Rule support for other entities would involve the creation of a new rule adapter in the appropriate module’s data project (Ellucian.Colleague.Data.Module), including registration of the adapter in the module’s bootstrapper method.
Custom Code Samples

How to read additional data in a repository

There are two ways to access data in Colleague from the Ellucian Colleague Web API: either with a data reader or a transaction. Readers just read records. Transactions execute a subroutine within Colleague that can read or update data.

This code sample will illustrate how to add the Colleague field CRS.SCHED.TYPE to the repository. This assumes you have the Colleague Data Contract Generator plugin installed and configured. You can check for that by going to **Tools > Extension Manager**.
1. First, locate the data contract. It will always have the same name as the Colleague data file.

2. Right-click and select **Update Data Contract**.

3. Log into the development environment if prompted.
4. Select the environment and application in which to find the file. For example, COURSES is in ST. Expand the right pane to find the desired field:

5. Select the desired field.

- CRS.CONFLICT.DEDECER
- CRS.SCHED.TYPE
- CRS.BILLING.PERIOD.TYPE
6. Then click **Generate**. Now the field is read from the database when the Data Accessor is used. Next, a place is needed to put the data in the domain.

7. Open the source file and add a field for the data.
8. Next, alter the repository to store the data in the domain.

There are slight differences between the repositories, but they all generally use a "GetOrAddToCache" method to read data and cache it. There is usually a "BuildXXXXX()" method that builds the objects if not found in cache.

9. Generally, just look for a line using a dataReader and some kind of ReadRecord method for the file you are interested in, and then see where that data goes:

```csharp
var bulkData = dataReader.BulkReadRecord<Course>("COURSES", subList);
coursesData.AddRange(bulkData);
```

```csharp
var courseList = BuildCourses(coursesData);
return courseList;
```
10. Go to the BuildCourses method and hook the new field up with its home in the domain:

11. Build the solution and start using the new field.

How to update additional data in a repository

Updates to data in Colleague always involve a Colleague Transaction.

1. If you are adding to an existing Colleague Transaction, find the transaction that is updating the data you want to add to. If you are creating a new Colleague Transaction to update something in Colleague, just go to step 2 and create a new Colleague Transaction directly in Colleague Studio.

   // Update the plan
   Transactions.UpdateDegreePlanResponse updateResponse =
   transactionInvoker.Execute<Transactions.UpdateDegreePlanRequest, 
   Transactions.UpdateDegreePlanResponse>(updateReq);

   The request and response objects are generated. There will be a Colleague Transaction of the name UPDATE.DEGREE.PLAN in Colleague.

2. Using the procedures outlined in the Managing Custom Software manual, customize the Colleague Transaction you have selected. In the following example, it is assumed that you are cloning the process to a new name of CUST.UPDATE.DEGREE.PLAN. Be sure to declare the process as a custom version by Setting Original Process Name, and leave the version number of the transaction unchanged.
3. Go to the **Elements** tab. Add the data element you want to update to the **Transaction Elements** window, making sure that the usage is “LOCAL” and “Reference only” is unchecked.
4. Then, in order to pass the data in, add a variable for it to the **Transaction Variable Details** window.

Where necessary, make sure the Usage property is set to “IN” or “INOUT” and that there is an appropriate Alias for the attribute – the attribute’s alias is what will appear in the .NET side of the transaction.

Use the Output conversion string appropriately for decimal numbers and dates, and use the Multi-valued field for lists.
5. When updating multivalued attributes that are part of associations, be sure the Group for the association has been defined and that all the elements of the association are included.

6. After updating the Colleague Transaction code, generate the transaction using Colleague Studio and test it using the Colleague Studio debugger. Remember that the highlighted insert below must be commented out to run the Colleague Transaction with the debugger, and then uncomment it before checking in the changes.
7. When it functions satisfactorily, go back to Microsoft Visual Studio and import the new transaction.

The repository that uses this Colleague Transaction will need to be updated to use the new Colleague Transaction. Populate it with data, and the transaction will update it as instructed.

```csharp
var tc = new Transaction();
tc.SectionIds = plannedcourse;
tc.Credits = plannedcourse;
//custom
var tc = new Transaction();
tc.CreditCourse = 0;
```

You could also use the transaction to return data in the response object, by setting up a transaction variable, as above, with the Usage set to "OUT" and populating it.
How to add new data to a data transfer object

1. Find the DTO related to the domain object you want to alter, and add the field:

```csharp
namespace Ellucian.Colleague.Dtos.Student
{
    public class Course
    {
        public string Id { get; set; }
        public string SubjectCode { get; set; }
        public string Number { get; set; }
        public string ScheduleType { get; set; }
    }
}
```

If it’s a simple object, it may be automapped. If the field name and type are the same in the equivalent domain object and the new DTO, it may map without any further work.

2. If mapping an object using AutoMapper, you must add a dependency on the lower level adapter.

```csharp
{
    public class CourseEntityAdapter : AutoMapperAdapter<Course, Ellucian.Colleague.Dtos.Student.Course>
    {
        public CourseEntityAdapter(IAdapterRegistry adapterRegistry, ILogger logger)
            : base(adapterRegistry, logger)
        {
            // Mapping dependency
        }
    }
}
```
In this example, the Course entity is simple enough that most of the fields map automatically – only its child element, Corequisite, needs any help. Our previous example would work. However, more complicated objects, or objects without exact name-and-type matches, would need more work in the adapter:

```csharp
{
    {
        /// <summary>
        /// This class maps a program evaluation entity to an outbound program evaluation DTO
        /// </summary>
        public ProgramEvaluationDtoAdapter(IAdapterRegistry adapterRegistry, ILogger logger) : base(adapterRegistry)
        {
            {
                programEvaluationDto.ProgramCode = source.ProgramCode;
                programEvaluationDto.CatalogCode = source.CatalogCode;
                programEvaluationDto.Credits = source.Credits;
                programEvaluationDto.InstitutionalCredits = source.InstitutionalCredits;
                programEvaluationDto.InProgressCredits = source.InProgressCredits;
                programEvaluationDto.PlannedCredits = source.PlannedCredits;
                programEvaluationDto.CumGpa = source.CumGpa;
                programEvaluationDto.InstitutionalCreditsModificationMessage = source.InstitutionalCreditsModificationMessage;
                programEvaluationDto.InstitutionalGpaModificationMessage = source.InstitutionalGpaModificationMessage;
                programEvaluationDto.OverallGpaModificationMessage = source.OverallGpaModificationMessage;
            }
        }
    }
}
```
Testing

Unit testing is an important part of the Colleague Web API. Unit tests created at Ellucian during development of the API are delivered with the source code. In the API solution, you will find a test project for each project that contains testable classes. Ideally, the unit test classes should provide test coverage for every line of logic and every logical data scenario, particularly in the Data, Domain and Coordination projects for each module. For each additional piece of logic you add, you should also add unit tests. These unit tests are extremely important for verifying that any changes you make do not have regressive effects on the application. Debugging is much simpler with reliable unit tests, because tests can be written and logic verified without actually reading from a database. Unit tests should be written to fully exercise the code with consideration for all the ways the data may be presented and should make sure the code takes into account the possibility of “unexpected” or “bad” data.

*If you are unfamiliar with the concept of unit test classes, please read!*

The topic of testing must be taken seriously and time invested to understand how to conduct sound testing for the following reasons:

- Although creating tests demands more development time, it is much more time and cost-effective in the long run to have thorough unit tests for regression testing. You write a test only once, and it is there to verify every change you make thereafter. In contrast, testing using a web proxy tool like Fiddler takes an enormous amount of time to re-test every time, and the results will be affected by changes to the connected database.

- There are many data scenarios that are difficult to set up. You can set up virtually any scenario with a test data repository, and thereafter that scenario will be verified with each change to your logic without any extra effort.

Although it takes some time to learn how to set up a test project and write tests, unit tests you produce will save you logarithmically more time in the long run than it takes to write the test, and will result in more robust and more reliable software.
Test-first development

When an issue occurs, or a new feature is to be added, there is a specific sequence that Ellucian recommends that you follow.

1. Identify a behavior that is desired in the system.
2. Write the test to reflect the desired system behavior.
3. Run the test and verify that it fails.
4. Add code to the system to cause the desired behavior.
5. Run the test and verify that it succeeds.
6. Run all tests in the solution and verify that they all still succeed, even with the new code change.

Example

Suppose you want to change a method in the CourseRepository class so that it logs an error instead of throwing an error when there are courses that could not be found in the repository for the given IDs.

```csharp
public IEnumerable<Course> Get(IICollection<string> courseIds)
{
    var courses = new List<Course>();
    if ((courseIds != null) && (courseIds.Count() != 0))
    {
        // Select the courses in the list of all courses that match the given list of ids
        // Throw an error if any items not found
        var allCourses = GetAllCourses();
        string idsNotFound = "";
        foreach (var id in courseIds)
        {
            try
            {
                courses.Add(allCourses[id]);
            }
            catch
            {
                idsNotFound += " " + id;
            }
        }
        if (idsNotFound != "")
        {
            throw new ArgumentException("Courses not found for course IDs" + idsNotFound);
        }
    } return courses;
}
First, go to the CourseRepositoryTests class to create a test for this condition.

There is already a test for the behavior that is programmed, checking that an error is thrown when not all the specified courses are found:

```csharp
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void Get_Many_ThrowsErrorIfCourseNotFound()
{
    var courseIds = new Collection<string>() { "46", "3333"};
    var courses = courseRepo.Get(courseIds);
}
```

You want to change it to instead just return the ones it finds and not throw an error:

```csharp
[TestMethod]
public void Get_Many_IgnoresCoursesNotFound()
{
    var courseIds = new Collection<string>() { "46", "3333"};
    var courses = courseRepo.Get(courseIds);
    Assert.IsTrue(courses.Count() == 1);
    Assert.IsTrue(courses.ElementAt(0).Id == "46");
}
```

At this point when you run the test, it fails.
Make the change to the logic, and run the test again.

```csharp
public IEnumerable<Course> Get(List<string> courseIds)
{
    var courses = new List<Course>();
    if((courseIds != null) && (courseIds.Count != 0))
    {
        // Select the courses in the list of all courses that match the given list of ids
        // Throw an error if any items not found
        var allCourses = GetAllCourses();
        string idsNotFound = "";
        foreach (var id in courseIds)
        {
            try
            {
                courses.Add(allCourses[id]);
            }
            catch
            {
                idsNotFound += " " + id;
            }
        }
        //if (idsNotFound != "")
        //{
        //    throw new ArgumentException("Courses not found for course Ids" + idsNotFound);
        //}
        return courses;
    }
}
```

Then run all tests to ensure that you have not caused regression.

Note: This modification is for demonstration purposes only and is not a change that has been made to the actual repository as of this writing. It is always important to keep in mind that any change to the API may have ramifications to any consumer of the API. What is our consumer expecting? Will there be a problem with the display if there is not an error returned by this method when any one of the courses is not found? For these reasons, it is important to be extremely cautious when making changes to the API because any change can have an enormous impact on API consumers.
Test data repositories

Test data repositories provide a predictable database that can be used for writing tests; particularly repository tests. This is necessary because the development environment may change, and that could invalidate all tests. Therefore, create test data repositories; typically one for each repository class that in effect, emulates a database.

Fortunately, you do not need to do this from scratch. Test data repositories are delivered with the source code, so you can simply add to what is already there. Test data repositories are named TestRepositoryName.cs, where RepositoryName is the name of the Repository being tested. Some examples include TestAcademicCreditRepository.cs and TestTermRepository.cs. They can be found in the domain test project for each module. A test data repository is an implementation of a repository interface. TestTermRepository.cs is a good one to look at to get an idea of the layout of a typical test data repository.

Test data repositories can be used for Coordination Service unit tests, Repository unit tests, or any unit tests that need to emulate retrieval from a repository. For example, open up the TestTermRepository.cs and right-click on the class name, and select View all references to see a list of the many examples of how the test data repository is utilized by the unit tests.

The test data repositories are used directly by unit tests simply by instantiating the test repository and invoking a method from the repository. For example, the degree plan entity tests need the registration terms. The following statement instantiates the TestTermRepository to get the list of registration terms:

```
regTerms = new TestTermRepository().GetRegistrationTerms().ToList();
```
Test data repositories can also be used to build repository responses, used for the repository tests. For example, after retrieving terms from the TestTermRepository, this method in the TermRepositoryTests takes the list of term domain (entity) objects and builds a collection of objects to resemble a response that is to be received when a data contract is used to retrieve Terms from Colleague. This data contract response can be used over and over again in unit tests that verify the TermRepository. You can see how this response is specifically used in the next section on Mocking.

```csharp
private ICollection<Term> BuildTermsResponse(IEnumerable<Term> terms)
{
    ICollection<Term> repoTerms = new Collection<Term>();
    foreach (var term in terms)
    {
        var termData = new Term();
        termData.RecordKey = term.Code.ToString();
        termData.TermStartDate = term.StartDate;
        termData.TermEndDate = term.EndDate;
        termData.TermReportingYear = term.ReportingYear;
        termData.TermDesc = term.Description;
        termData.TermSequenceNo = term.Sequence.ToString();
        if (term.DefaultOnPlan)
        {
            termData.TermDefaultOnPlan = "Y";
        }
        if (term.ForPlanning)
        {
            termData.TermDegreePlanning = "Y";
        }
        termData.TermReportingTerm = term.ReportingTerm;
        repoTerms.Add(termData);
    }
    return repoTerms;
}
```

Mocking

Mocking is used to enable testing of components at every level of the solution. In general, it literally “mocks” the behavior of a method or function in the logic so that a unit test can verify functionality of the target component. In the API, Ellucian has chosen to use the mocking library Moq. It supports mocking interfaces and classes. There are other testing tools available for mocking as well. If the testing tool is changed, many of the tests would also need to be refactored to accommodate the mocking tool of choice.

More information about Moq can be found at:

https://code.google.com/p/moq/

Mocking is typically used in all tests except the domain Entity tests. Mocking is probably best learned by example, so the following is an example from TermRepositoryTests:
Mocking is used to "mock" almost everything except the component that is being tested. In testing the TermRepository, mock the transaction factory, the logger, and the cache provider; basically, everything that is an argument to the TermRepository constructor shown below:

```csharp
public TermRepository(ICacheProvider cacheProvider, IColleagueTransactionFactory transactionFactory, ILogger logger)
{
    // Using level 1 cache time out value for data that rarely changes.
    CacheTimeout = Level1CacheTimeoutValue;
}
```

And then you also need to mock the transactionInvoker, because you want to mock the response returned when the transaction is invoked, as shown in the following code from the TermRepository:

```csharp
public TermRepository(ICacheProvider cacheProvider, IColleagueTransactionFactory transactionFactory, ILogger logger)
{
    // Using level 1 cache time out value for data that rarely changes.
    CacheTimeout = Level1CacheTimeoutValue;
}
```

Often, mocking of an object is accomplished simply by instantiating Mock for the class of the item to be mocked. In the case of the transactionInvoker, however, you also need to set up the response to the call to the Execute method. This is so you can verify that the repository reacts properly to a specific situation. In this case, the unit test is "pretending" that no registration terms exist in Colleague. Therefore, the test sets up the transaction invoker to respond to any request to the GetRegistrationTerms transaction with an empty list of strings.

The repository itself is then instantiated using the mocked objects, and the specific repository method called, with an assert statement to verify that an empty list is returned by the repository method and not an exception.

Mocking can also be used to throw errors, mock property return values, and execute callback functions. You can find examples of all of these techniques in the delivered unit tests.
Test harness

In the Ellucian.Colleague.Coordination.Planning.Tests project, in the Services folder, there is the file `ProgramEvaluationServiceEnvironmentTests.cs`. This is a special test harness to allow Program Evaluations to be run against environment data within the context of the debugger. This is useful to see how changes to Evaluation processes affect Evaluations with real-world data sets.

It is important, at the top of the harness’ class file, that the “Ignore” directive be commented out to run the tests, or Visual Studio will ignore them. Then, on checking in any changes, it must be commented back in. This will prevent slower tests against the environment from running any time someone chooses “run all tests in solution.”

To use the harness, first get the configuration information from the administrator. The harness will log into the environment as if the student typed their login information into the interface. You can have several copies of the login information, commented out all but one, to facilitate switching between environments:

```csharp
{
    /// <summary>
    /// This starts with a copy of ProgramEvaluationServiceTests. The aim is to set up a harness that can
    /// be used to test against live environments. These tests need to be marked Ignore on checkin so they
    /// do not run against live databases for everyone.
    /// </summary>
    public class ProgramEvaluationServiceEnvironmentTests
    {
        [TestClass][Ignore]
        public void TestInitialize()
        {
            // Login
            service = new BaseColleagueService();
            service.Initialize();
            logger = new Mock<ILogger>().Object;
            settings = new DmiSettings();

            // TEST
            settings.AccountName = "test_account";
            settings.IpAddress = "yourserver.yourschool.com";
            settings.Port = 5000;
            settings.SharedSecret = "sharedsecret123";
            token = new ColleagueSessionRepository(settings).Login("studentuserid", "studentpassword");

            // LIVE (NEVER USE THIS)
            //settings.AccountName = "live_account";
            //settings.IpAddress = "yourserver.yourschool.com";
            //settings.Port = 5000;
            //settings.SharedSecret = "sharedsecret123";
            //token = new ColleagueSessionRepository(settings).Login("studentuserid", "studentpassword");
        }
    }
}
After the login information is entered, create a test method:

```csharp
[TestMethod]
public void NewChangeTest()
{
    var eval = programEvaluationService.Evaluate("1143352", "STSS.MATH.BS"); //hawking
    var eval = programEvaluationService.Evaluate("1143373", "STSS.MATH.BS"); //tyson
    var eval = programEvaluationService.Evaluate("1133031", "AA.HIST"); //sagan
    var eval = programEvaluationService.Evaluate("1143629", "STSS.MATH.BS"); //newton
    dumper.Dump(eval, "verbose");
}
```

and execute the test. The Dump() call sends useful output to the console.

With the “verbose” option it lists the disposition of each credit and planned course. The “brief” option just displays down to each group.
Debugging during development: Colleague Studio

Use the Colleague Studio debugger to debug Colleague Transactions. You can find complete documentation in the online *Colleague Studio Debugging Guide* within Colleague Studio.

**Note:** When debugging Colleague Transactions, remember to comment out the insert shown below. When your debugging is complete, add this insert back as the first line of the transaction logic and re-generate the code. Otherwise, the transaction will return an error when called from .NET. Also note that this insert cannot be used in anonymous transactions.

```sql
$INSERT I_VERIFY.CTX.SIGNATURE FROM UT.INSERTS
```

Debugging during development: Visual Studio

1) Debugging with a unit test repository

Unit test classes and test data repositories have been delivered with the source code, and are a very important part of the Colleague Web API. Please see the Testing section for more information on the importance of unit tests and writing unit tests.

Debugging is much simpler with robust unit tests and test data repositories because tests can be written and logic verified without actually reading from a database.

Follow these steps to debug from your test repository.

1. Find a test that tests the area you want to debug. For example, if you are having an issue with the course repository, find a test in *CourseRepositoryTests* and click in the margin to set a breakpoint in the logic of the test.

```
[TestMethod]
public void Get_Many_Returns_Empty_List_If_NoIdsSpecified()
{
    var courseIds = new Collection<string>();
    var courses = courseRepo.Get(courseIds);
    Assert.AreEqual(0, courses.Count());
}
```

2. Click somewhere within the logic of the test to set the context, and choose the option to run the test in debug mode by using the shortcut, or by clicking the highlighted icon.
3. When the breakpoint is hit, you can use **Step Over** to proceed forward one line, or **Step Into** to follow the logic into another method.

For example, if **Step Into** is chosen as shown above, you will proceed into the course repository `Get` method as shown below.
4. Click Debug.Windows.Locals to see the values of active variables at any point in time.

2) API debugging via a web debugging proxy

Presuming the route and the controller are set up, you can use a web debugging proxy to debug the logic in the API. Although no substitute for a test repository, this is a convenient method to verify that the API is responding properly to an actual web request. From Visual Studio, choose the option to Start Debugging. This option is available from the Debug menu, or by pressing F5.

1. Set a breakpoint in the logic that you want to examine. For example, suppose you want to see something that is happening in the constructor of the Course entity. You can set a breakpoint in the constructor by clicking in the margin. The red dot indicates a breakpoint is set.
2. Compose a request to the API. Fiddler2 is a great tool for doing this, and can be downloaded from www.fiddler2.com.

Unless you are testing logic on a route that allows anonymous access, you first need to compose a request to log into your colleague environment as shown below, and then click **Execute**.

![Fiddler2 Request Example](image.png)

```json
{
  "user": "mylogin",
  "password": "mypassword"
}
```
3. Click on your login request reflected in the **Web Sessions** window, and then on the **Inspectors** tab and the **TextView** tab in the bottom window to get the authorization string that is returned by the request.

Copy the authorization string.
4. Compose the API request to the specific endpoint you want to test, in this case the **Get Courses** endpoint. Don’t forget to add **X-CustomCredentials:** to your Request Header and paste in the authorization string.

Then click **Execute** to send the request.

![API Request Example](image)

The breakpoint you selected in the **Course** constructor will be hit while the API is working to fulfill the request.

```csharp
public Course(string id, string title, ICollection<string> departmentCodes, string subjectCode, string number,
{
    if (string.IsNullOrEmpty(title))
    {
        throw new ArgumentException("title");
    }
```
5. You can use the commands in Visual Studio to step through the code and observe the behavior to determine the source of the problem you are trying to debug. You can hover over a variable to see the value of the variable, or you can click Debug > Windows > Locals to see all the active variables when the breakpoint is hit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>&quot;101&quot;</td>
</tr>
<tr>
<td>title</td>
<td>&quot;Cost Accounting&quot;</td>
</tr>
<tr>
<td>departmentCodes</td>
<td>Count = 1</td>
</tr>
<tr>
<td>subjectCode</td>
<td>&quot;ACCT&quot;</td>
</tr>
<tr>
<td>number</td>
<td>&quot;100&quot;</td>
</tr>
<tr>
<td>academicLevelCode</td>
<td>&quot;UG&quot;</td>
</tr>
<tr>
<td>courseLevelCodes</td>
<td>Count = 1</td>
</tr>
<tr>
<td>credits</td>
<td>3</td>
</tr>
<tr>
<td>ceus</td>
<td>null</td>
</tr>
<tr>
<td>status</td>
<td>Active</td>
</tr>
</tbody>
</table>

6. Choose Continue (or F5) at any time to let the logic continue to the end, and you can observe the results from Fiddler2 by clicking on the request in the WebSessions window, and then choosing the Inspectors tab.
3) Service client debugging

To verify methods in the Service Client in the Ellucian.Colleague.Api.Client project, perform the following steps (requires a running application):

1. Start the application locally from the browser (url: localhost/Ellucian.Web.Student)

2. Choose Debug>Attach to Process. In the dialog, scroll to the bottom of the Available Processes window, and choose the w3wp.exe process attached to the app pool or user.
   
   Click Attach, and then click Attach again when the Attach Security Warning appears.

3. Set a breakpoint in the service client method you need to test. If attached properly, the breakpoint will appear as a solid dot in the margin.
Notice if you set a breakpoint outside of the ServiceClient, such as in a controller method shown below, the breakpoint will be an open circle, indicating it is unreachable.

4. Proceed with running the Student application until the breakpoint is hit. Step through logic as needed to debug.
Debugging after deployment

You must debug the API after deployment primarily through IIS and logging. There are several levels of resources that are available for debugging issues.

DMI log

The DMI log can be useful for debugging initial communication issues. Often issues that occur at the DMI level are as a result of improper setup. The DMI log should be cleared, and then turned on in “full” mode from SA Valet and the workflow executed from the application to capture the results. The following are some things to look for in the DMI log:

- Search for activity on the DMI log that indicates communication has occurred with DMI, such as LGRQ, indicating a login request. If no activity is found, there may be an issue in the Colleague Web API Configuration Settings, such as the DMI Application Listener IP Address or the DMI Application Listener Port. Be sure you are able to successfully test the login from the configuration page.

- Check for an error due to a mismatched shared secret, or some other error by searching for ERRS, indicating an Error Response.

- Search the log for the specific file name (COURSE.SECTIONS) or Colleague Transaction name (UPDATE.DEGREE.PLAN) to see if activity is occurring.

Debug using IIS MVC and API logs

If there appears to be activity without errors in the DMI log, check the API log. The API log will show errors that were thrown by the API for issues found by a domain item constructor, a repository, a coordination service, or a controller. The API log is turned on from the API configuration page on the server. If the log level is changed, this page will restart the app pool for you.
1. In IIS, expand the environment_WebApi site to reveal the ColleagueApi node.

2. Expand the ColleagueApi node to reveal the App_Data directory. Right-click the App_Data directory, and then choose Explore.

3. Navigate one level down into the Logs directory and Delete or rename the ColleagueWebApi.log entry (if for the current date) to clear the log. Keep this Explorer window open.

4. Back in IIS, with the ColleagueApi node selected, find “Browse Application” in the rightmost pane and click the “Browse…” link directly underneath. The Colleague Web API Help page displays in a browser window.

5. Click the Configuration Settings link in the upper-right corner.

6. On the configuration form, choose a log level of “Verbose” and save.

7. The application pool recycles itself when you save from this form (but only when you have made a change). If the application pool needs to be restarted, do this:

   7.1 In the list of Application Pools in the center pane of IIS, select the one named environment_WebApi.

   7.2 In the rightmost pane, Under Application Pool Tasks, click Stop. When the Start button is available, click Start.
8. After running the workflow, you will find the log in the App_Data directory.

Below is an example of what may be seen in the Web API log:

```
Value cannot be null.
Parameter name: title

at Ellucian.Colleague.Domain.Student.Entities.Course..ctor(String id, String title, ICollection`1 departmentCodes, String subjectCode, String number, String academicLevelCode, ICollection`1 courseLevelCodes, Nullable`1 credits, Nullable`1 ceus, CourseStatus status) in
C:\TOP\StudentSelfService\WebApi-Dev\Source\Ellucian.Colleague.Api\Ellucian.Colleague.Domain.Student\Entities\Course.cs:line 209
at Ellucian.Colleague.Data.Student.Repositories.CourseRepository.BuildCourses(IEnumerable`1 courseData) in
C:\TOP\StudentSelfService\WebApi-Dev\Source\Ellucian.Colleague.Api\Ellucian.Colleague.Data\Repositories\CourseRepository.cs:line 161
```

Note: If you are working with the Self Service Student UI, a similar procedure may be used to turn on and view the MVC log. Simply follow the same instructions from the Student node in IIS to access the configuration, turn on the log, and view the log.
Debug using failed request tracing

If nothing appears in the API or MVC logs generated from IIS, there’s a good chance that the request is failing to get to the API layer. Use Failed Request Tracing to see the details of the requests made to the API from the UI. There are many references on the web, such as the following instructions, for setting up failed request tracing. You may also use these steps:

1. Verify that Tracing is installed.

   On the API server, open Server Manager. In the WebServer (IIS) section, click **Add Role Services**. Look under **Health and Diagnostics**, and verify that Tracing is installed. If not, select the check box for **Tracing**, and click **Next** to install. Also, go to Traceviewer and install the trace viewer add-on from the API server.

2. Turn on Failed Request Tracing in IIS.

3. Click on the API site in the left panel. Under **Configure** in the right panel, click **Failed Request Tracing** and a dialog will pop up. (If you don’t see Failed Request Tracing right after installing it on your server, try closing and re-opening IIS). Select the **Enable** checkbox. While there, copy the path in the **Directory** field and open a Windows Explorer window and paste the path.
You may see several folders in the log files directory. When tracing is performed, a folder is created with a name W3SVCnn, where “nn” corresponds to the ID of the site being traced. If you click on the top-level Sites folder in IIS, a list of sites will display with corresponding IDs. Find your site to know the name the folder will be given. It would be a good idea to clear the folder now, before starting a new tracing session.
Set up failed request tracing rules

Back in IIS, click on your API site and double-click the Failed Request Tracing Rules icon. Remove any failed request tracing rules that appear in the middle window.

1. Under Actions, choose Add.

2. Choose to trace “All Content (*)” then in the next dialog, select the Status code(s) check box and enter “200 – 503” to trace all requests, and then click Next.

3. Verify that all providers on the final dialog page are checked, especially WWW Server, and then click Finish.

Run the application to produce the tracing files.
Analyze tracing results

If you view the folder path given when you turned on failed request tracing, you will see what look like meaningless files.

Double-click on any entry to see the detailed view of each request to diagnose the source of the problem. When you click on an item, it will be brought into the browser and will look something like this.

Click on Request Details to see an outline of the request, and on Compact View to see the details at each step of the request. First look for requests with a status code in the 400s or 500s. But sometimes failures can occur even when the status code is in the 200 range.

**Note:** You will have an easier time looking through these requests if you can install the IIS Trace Viewer on your server. When you install this add-on, you will see a Trace Viewer Icon in IIS for the site you are tracing, and when you click the icon, it will take you to a listing that displays the URL and the status code of each request without clicking into each one.
Machine Key error on the Colleague Connection Settings page during development

While debugging/running the Colleague Web API solution locally on a development PC and when deploying to a development test server it is normal to see the following error on the Colleague Connection Settings page of the Colleague Web API.

This error is show during development because the web.config file within the Ellucian.Colleague.Api project contains a static machine key. The static machine key is necessary during development as the decryption of the Shared Secret, stored in the settings.config file, is dependent on it. A separate error will be displayed if the encrypted Shared Secret cannot be read and that error would need to be corrected before being able to debug; however, the presence of the Machine Key error as shown above will not affect the software during development debugging. Release builds that are loaded on a test or production web server must heed this error and take the corrective action noted in the Setting Up Colleague Web API manual under the “Generate machine keys for security” section.
Packaging and Deployment

This section discusses the recommended methods available for packaging and deploying your customized solution to your various web servers. This section will first provide general background on the recommended options, followed by the setup required on the developer PC and web server, how to configure and use Visual Studio to package your customized solution, and finally how to deploy to the web server. Depending on the target Colleague/web server environment (development, test, or production), there are slight variations in the deployment and packing methods in order to shape what is deployed and how it is deployed to each environment. Read this section carefully before attempting any packaging and deployments as topics presented build on each other in order to create a complete understanding of the processes.

Overview of packaging and deployment methods

Two deployment methods will be presented in this section. Both techniques are based on the Microsoft Web Deploy package process. This process allows you to create a web deployment package in Visual Studio (which consists of a .zip archive with all of the required site content and files containing the deploy-time configuration settings), and then deploy that package to your web server locally or remotely from another PC. Both methods will begin with you configuring the Web Publishing parameters in Visual Studio and using Visual Studio to build the web deployment package, then you will use one of the two methods to deploy the web deployment package to the web server.

Method 1: Import application using IIS Manager

Web Deploy, when installed on a server with IIS, installs an IIS extension that allows you directly import the web deployment package to a new or existing website. The Import Application wizard will prompt you for various site settings and can also prompt for application configuration settings. When configured to do so, Ellucian has pre-configured the solution's web deployment packaging settings to prompt for some of the application settings, such as the Colleague connection details. This method is very simple and relatively quick, but it does require that you copy the web deployment package from the development PC to the web server or a mapped network drive because the Import Application wizard needs direct file access to the web deployment package. Another downside to this method is that you will need to re-enter the application settings each time you import the application.
Method 2: Remote deploy using Web Deploy

Web Deploy, when installed on a server with IIS, installs a service (Web Deployment Agent Service) that allows other servers and PCs with Web Deploy to remotely deploy web deployment packages to a new or existing website. The developer will install Web Deploy and using the Web Deploy command line, deploy the web deployment package to the target website. The Web Deploy process allows you to specify a settings file, which contains the website and application settings, as you deploy, meaning you can create settings files for each target Colleague environment and web server you deploy, without needing to re-package the web deployment package in order to deploy to a different environment. Web Deploy can also be used to sync websites on multiple web servers, which makes deploying to a server farm very easy. There is a bit more complexity with this method (as it is a command line process), but once the settings files are configured and the process is performed a few times, it proves to be a great method for deploying the customized solution.

See [http://www.iis.net/downloads/microsoft/web-deploy](http://www.iis.net/downloads/microsoft/web-deploy) for more information about Microsoft Web Deploy and to download the software.

The step-by-step procedures for each method above are presented later in this section.

Suggested build packaging, build storage, and deployment workflow

The following diagram illustrates the process of taking the output of the Visual Studio build configuration (the web deployment package) and storing and deploying that output to your various Colleague environments.
Visual Studio build configuration

The Visual Studio build configuration allows you to specify differing behavior and application settings when building the solution based on its intended purpose. These configurations will be used to dictate the quality and capabilities of the build. Later in this section the usage of the build configurations will be explained in more detail based on the Colleague environment being targeted. *Never* deploy a debug build to Production as the debug configuration, as delivered, has various features (such as remote errors and debug .NET compilation) enabled which can compromise the security of a production website.

When building a release-quality build, you should always deploy to Test first, verify the functionality, and then deploy the same web deployment package to Production. If you discover an issue in test, do not make a coding change, re-package, and then deploy directly to Production; always re-deploy any changes to Test, verify, and then deploy to Production. A technique for changing the settings of the Web Deploy package when targeting production will be discussed later in this section.

Build storage directory

The build storage directory provides a method for capturing the build output and, when using source control, tracking the web deployment package used for a given environment. You will copy the web deployment package build by Visual Studio to these locations, and then deploy the packages from that location. In the case of production, you will copy from the Test build directory, as what you deploy in Production should have been deployed and validated in Test first. You will need to create these directories the first time you copy the web deployment package into them.

Colleague/web server environment

This is the target Colleague environment/web server you will eventually be deploying to.
Developer PC setup

When using deployment Method 1: Import Application Using IIS Manager, there is no addition setup needed on the developer PC because all of the required components are included with the Visual Studio Installation.

When using deployment Method 2: Remote Deploy Using Web Deploy, you need to install Microsoft Web Deploy version 2.1 or higher on the development PC in order to have command line access to the MS Deploy executable.

Web server setup

This setup must be done regardless of which deployment method (discussed above) that you choose.

1. Install the Colleague Web API installer on the target server in order to create and provision the IIS website and application.

2. Follow the installation instructions found in the Setting Up Colleague Web API manual. Also, be sure to follow the instructions in the manual pertaining to server setup and component naming prior to running the Colleague Web API installer.

3. Install MS Web Deploy 2.1+

   Note: If you are using the remote MS Deploy method (method 2), make sure the Web Deployment Agent Service is running as a Windows service. If you are not planning on using the remote deploy method, you can choose to stop this service and set its startup options so they do not automatically start. The service uses server authentication during the deployment process, so it is secure and safe to leave running.

Visual Studio build configurations setup

The build configurations within Visual Studio should be configured using the following steps below for each included build configuration in the Ellucian.Colleague.Api solution.
Debug builds


4. In the project properties editor, select the Package/Publish Web tab.

5. On the Package/Publish Web tab: From the configuration drop-down (on the tab, not in the Visual Studio toolbar), select Debug.

6. Under the Items to deploy (applies to all deployment methods) heading, do the following:
   6.1 Select Only file needed to run this application.
   6.2 Clear the Exclude generated debug symbols check box.
   6.3 (Visual Studio 2010 only) Clear the Exclude files from the App_Data folder check box.

7. Under the Items to deploy (applies to Web Deploy only) heading:
   7.1 Clear the Include all databases configured in Package/Publish SQL check box.
   7.2 Clear the Include IIS settings as configured in IIS check box.

8. (Visual Studio 2010 only) Under the Web Deployment Package Settings heading:
   8.1 Select the Create deployment package as zip file check box.
   8.2 Set the location to: obj\Debug\Package\Ellucian.Colleague.Api.zip (default)
   8.3 Set the IIS Web site/Application name to use on the destination server to match the website name on the development web server:
      - If your website is named "dev_WebApi" then the value for this field is: "dev_WebApi/ColleagueApi".
      - The application name (/ColleagueApi) is always the same for this solution.
Release builds


4. In the project properties editor, select the Package/Publish Web tab. On the Package/Publish Web tab: From the configuration drop-down (on the tab, not in the Visual Studio toolbar), select Release.

5. Under the Items to deploy (applies to all deployment methods) heading, do the following:

   5.1 Select Only file needed to run this application from the drop-down.

   5.2 Select the Exclude generated debug symbols check box.

   5.3 (Visual Studio 2010 only) Clear the Exclude files from the App_Data folder check box.

6. Under the Items to deploy (applies to Web Deploy only) heading:

   6.1 Clear the Include all databases configured in Package/Publish SQL check box.

   6.2 Clear the Include IIS settings as configured in IIS check box.

7. (Visual Studio 2010 only) Under the Web Deployment Package Settings heading, do the following:

   7.1 Select the Create deployment package as zip file check box.

   7.2 Set the location to: obj\Release\Package\Ellucian.Colleague.Api.zip (default).

   7.3 Set the IIS Web site/Application name to use on the destination server to match the website name on the test web server:

      - If your website is named "test_WebApi" then the value for this field is: "test_WebApi/ColleagueApi".

      - The application name (/ColleagueApi) is always the same for this solution.

XML settings transformations setup

The XML transformation files within the Ellucian.Colleague.Api solution should be configured using the following steps. All of these transformation files are located within the Ellucian.Colleague.Api project of the solution.
Debug transforms

App_Data\settings.Debug.config

The settings.Debug.config delivered with the solution source is used by Ellucian development for different debug builds; however, for client customizations the debug build is deployed to the development colleague environment, which should already be configured in the main settings.config file of the Ellucian.Colleague.Api project. Replace the values below with those from the main settings.config file.

```
<accountName>cv160\main\wst\rt\accountName>
<ipAddress>sd10\Wapp\databl\idd\ipAddress>
<port>5000</port>
<secure>true</secure>
<certificateHostnameOverride>/certificateHostnameOverride>
<connectionPoolSize>1</connectionPoolSize>
<sharedSecret>5d15dd88a1d873df51ab7a4d1626ac66f15a697b708bfc5c6663f5a477f8592f4</sharedSecret>
<logLevel>10</logLevel>
</settings>
```
web.Debug.config

The `web.Debug.config` file, as provided, needs to be edited to perform the correct transforms when packaging a debug build.

- The customErrors transform needs to have its transform action changed from *Insert* to *SetAttributes*
- The machineKey transform needs to be changed from `<machineKey validationKey="" decryptionKey="" xdt:Transform="SetAttributes"/>` to `<machineKey xdt:Transform="Remove"/>`

```xml
  <system.web>
    <customErrors mode="Off" xdt:Transform="SetAttributes"/>
  </system.web>

  <!-- remove machinekey if present-->
  <machineKey xdt:Transform="Remove"/>
</configuration>
```
Release transforms

App_Data\settings.Release.config

The settings.Release.config should be filled in with as many of the values you know for your test Colleague environment. The shared secret should be left empty here and filled in using the Colleague Web API configuration settings page after the web deployment package has been deployed to the target web server.

```
<?xml version="1.0" encoding="utf-8" ?>
<colleague>
  <accountName/>
  <ipAddress/>
  <port/>
  <certificateHostnameOverride/>
  <connectionPoolSize>0</connectionPoolSize>
  <sharedSecret/>
  <logLevel>Disabled</logLevel>
</settings>
```

web.Release.config

The web.Release.config file, as provided, is configured as it should be with one exception: the transform for the custom errors element needs to have its transform action changed from "Insert" to "SetAttributes" as shown below.

```
<system.web>
  <!-- Always turn off debug compilation in production environments -->
  <compilation xdt:Transform="RemoveAttributes(debug)" />
  <!-- Ensure remote exceptions are off -->
  <customErrors mode="RemoteOnly" xdt:Transform="SetAttributes"/>
  <!-- remove machinekey if present -->
  <machineKey xdt:Transform="Remove"/>
</system.web>
</configuration>
```
Creating the web deployment package using Visual Studio

The following files make up the web deployment package. These files will be located in the Ellucian.Colleague.Api\obj\Debug\Package for a Debug build and Ellucian.Colleague.Api\obj\Release\Package for a Release build.

- Ellucian.Colleague.Api.deploy.cmd – Used to execute the msdeploy.exe process. See the readme.txt file for more information.
- Ellucian.Colleague.Api.SetParameters.xml – Used to set environment-specific parameters that are read in by the msdeploy.exe.

Debug builds – Visual Studio 2010

1. Set the build configuration to debug.
3. After the package creation has completed, right-click on the Ellucian.Colleague.Api project in the Solution Explorer and choose Open folder in Windows Explorer.
4. Navigate to the obj\Debug\Package directory.
5. Copy the following files from the Package directory to the builds\dev directory. If it does not exist, create it.

- Ellucian.Colleague.Api.deploy.cmd
- Ellucian.Colleague.Api.deploy-readme.txt
- Ellucian.Colleague.Api.SetParameters.xml
- Ellucian.Colleague.Api.SourceManifest.xml
- Ellucian.Colleague.Api.zip

**Note:** If the *Ellucian.Colleague.Api.SetParameters.xml* file already exists in the builds\dev directory, then do *not* overwrite it. You should overwrite the other files specified.

---

### Debug builds – Visual Studio 2012


3. On the **Profile** tab of the Publish Web wizard, verify that the selected profile is "debug". If this is the first time publishing, select <new…> from the drop-down and create a profile named "debug".

4. Complete the following fields on the **Connection** tab of the Publish Web wizard:

   - **Publish method.** Select **Web Deploy Package** from the dropdown.
   - **Package location.** Enter `obj\Debug\Package\Ellucian.Colleague.Api.zip`
   - **Site/application.** Enter the name of the development website (as named in IIS) on the development web server, followed by "/ColleagueAPI". For example, if the development website is named "dev_WebApi", then the value for this field is: "dev_WebApi/ColleagueApi".

5. Complete the following fields on the **Settings** tab of the Publish Web wizard:

   - **Configuration.** Select **Debug** from the dropdown list.
   - Under the File Publish Options, clear the **Precompile during publishing** check box if checked.
   - Under the File Publish Options, clear the **Exclude files from App_Data folder** check box if checked.

6. Click **Publish** to create the Web Deployment Package.
7. After the package creation has completed, right-click on the Ellucian.Colleague.Api project in the Solution Explorer and choose **Open folder** in Windows Explorer.

8. Navigate to the obj\Debug\Package directory.

9. Copy the following files from the Package directory to the builds\dev directory. If it does not exist, create it.
   - Ellucian.Colleague.Api.deploy.cmd
   - Ellucian.Colleague.Api.deploy-readme.txt
   - Ellucian.Colleague.Api.SetParameters.xml
   - Ellucian.Colleague.Api.SourceManifest.xml
   - Ellucian.Colleague.Api.zip

   **Note:** If the *Ellucian.Colleague.Api.SetParameters.xml* file already exists in the builds\dev directory, then do **not** overwrite it. You should overwrite the other files specified.

---

**Release builds – Visual Studio 2010**

1. Set the build configuration to release.

2. Right-click on the *Ellucian.Colleague.Api* project in the Solution Explorer and select **Create Deployment package**.

3. After the package creation has completed, click on the *Ellucian.Colleague.Api* project in the Solution Explorer and select **Open folder in Windows Explorer**.

4. Navigate to the obj\Release\Package directory.

5. Copy the following files from the Package directory to the builds\test directory. If it does not exist, create it.
   - Ellucian.Colleague.Api.deploy.cmd
   - Ellucian.Colleague.Api.deploy-readme.txt
   - Ellucian.Colleague.Api.SetParameters.xml
   - Ellucian.Colleague.Api.SourceManifest.xml
   - Ellucian.Colleague.Api.zip

   **Note:** If the *Ellucian.Colleague.Api.SetParameters.xml* file already exists in the builds\test directory do **not** overwrite it. You should overwrite the other files specified.
Release builds – Visual Studio 2012


3. On the Profile tab of the Publish Web wizard, verify that the selected profile is "release". If this is the first time publishing, select <new…> from the dropdown and create a profile named "release".

4. Complete the following fields on the Connection tab of the Publish Web wizard:
   - **Publish method.** Select Web Deploy Package from the dropdown.
   - **Package location.** Enter obj\Release\Package\Ellucian.Colleague.Api.zip
   - **Site/application.** Enter the name of the test website (as named in IIS) on the test web server followed by "/ColleagueAPI". For example, if the test website is named "test_WebApi", then the value for this field is: "test_WebApi/ColleagueApi".

5. Complete the following fields on the Settings tab of the Publish Web wizard:
   - **Configuration.** Select Release from the dropdown list.
   - Under the File Publish Options, clear the Precompile during publishing check box if checked.
   - Under the File Publish Options, clear the Exclude files from App_Data folder check box if checked.

6. Click Publish to create the Web Deployment Package.

7. After the package creation has completed, right-click on the Ellucian.Colleague.Api project in the Solution Explorer and choose Open folder in Windows Explorer.

8. Navigate to the obj\Release\Package directory.

9. Copy the following files from the Package directory to the builds\test directory. If it does not exist, create it.
   - Ellucian.Colleague.Api.deploy.cmd
   - Ellucian.Colleague.Api.deploy-readme.txt
   - Ellucian.Colleague.Api.SetParameters.xml
   - Ellucian.Colleague.Api.SourceManifest.xml
   - Ellucian.Colleague.Api.zip

   **Note:** If the Ellucian.Colleague.Api.SetParameters.xml file already exists in the builds\dev directory, then do not overwrite it. You should overwrite the other files specified.
Deploying: Import application method

The following steps are the same regardless of which Colleague Environment you deploy to; just be sure to use the correct deployment package from the correct builds directory based on which Colleague Environment you are deploying to.

1. Remote desktop to the web server.
2. Copy the deployment package (Ellucian.Colleague.Api.zip) from the appropriate builds directory (builds\dev, builds\test, or builds\prod) to the web server.
3. Open IIS Manager.
4. Right-click on the website and choose **Deploy**, and **Import Application**.
5. Navigate to the location of the web deploy package that you copied to the web server in Step 2 and click Next.

![Image of Import Application Package dialog]

6. Accept the selected site contents by clicking Next.

![Image of Select the Contents of the Package dialog]
7. Enter the application settings on the next form.

**Warning**: Due to a bug in the import application workflow, all fields need to have value defined, or the resulting import will completely wipe out the `settings.config` file of the deployed website, making it unusable. The only field where you may need to do this is in the Certificate Override field. You can simply enter a period (.) here, and it will bypass the bug and the period will also be removed. When done, click **Next**.
8. Choose to overwrite all files and click **Next**.

9. Click **Next** to complete the import. After completing, the results of the application import are shown on the final page of the wizard.
10. Navigate to the deployed web site, configure the shared secret, and test the connection.

**Colleague Web API**

<table>
<thead>
<tr>
<th>Account DMI Registry Name</th>
<th>development ri</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMI Application Listener IP Address</td>
<td>coldev.school.edu</td>
</tr>
<tr>
<td>DMI Application Listener Port</td>
<td>54321</td>
</tr>
<tr>
<td>Connect to Application Listener Securely</td>
<td></td>
</tr>
<tr>
<td>Certificate Host Name Override</td>
<td></td>
</tr>
<tr>
<td>Connection Pool Size</td>
<td>1</td>
</tr>
<tr>
<td>Shared Secret</td>
<td>**********</td>
</tr>
<tr>
<td>Confirm Shared Secret</td>
<td>**********</td>
</tr>
<tr>
<td>Log Level</td>
<td>Off</td>
</tr>
</tbody>
</table>

[Test Connection] [Save] [Cancel]

**Deploying: Remote Microsoft deploy method**

**Deploy to development web server**

1. Navigate to the builds\dev directory.
2. Edit the `Ellucian.Colleague.Api.SetParameters.xml` file in the builds\dev directory, defining the parameters needed to connect to the development Colleague environment.
4. Verify that the website is functional, and that the settings on the Colleague Web API configuration page are correct.

**Deploy to test web server**

1. Navigate to the builds\test directory.
2. Edit the `Ellucian.Colleague.Api.SetParameters.xml` file in the builds\test directory, defining the parameters needed to connect to the test Colleague environment.
4. Verify that the website is functional and that the settings on the Colleague Web API configuration page are correct.
Deploy to production web server

1. Navigate to the builds\test directory.

2. Copy the contents from the builds\test directory to the builds\prod directory.

   **Note:** If you have previously performed this step, you do not need to copy the `Ellucian.Colleague.Api.SetParameters.xml` file as it will already exist and will have the correct settings for production (but you should always verify this prior to deploying).

3. Edit the `Ellucian.Colleague.Api.SetParameters.xml` file in the builds\prod directory, defining the parameters needed to connect to the production Colleague environment.


5. Verify that the website is functional and that the settings on the Colleague Web API configuration page are correct.
Integrating Your Customizations with a New Release

This section discusses the concepts and procedures for integrating your customizations with new releases of the source code.

General process

Whenever a new version of the Colleague Web API is released, updated source code will be released as well. The source code accompanying a new release will reflect the current state of the Ellucian codebase, meaning that if you were to build from the source code you would end up with the current binary version of the software. Currently, Ellucian is simply providing the source items as a snapshot (using the .zip archives and the unpacking script to create a working solution), meaning that differences will appear as new, removed, or modified files with no change history. (A future goal is to host the source code using a repository that would allow you to see the change history as well). In order to integrate your current version of the Ellucian.Colleague.Api solution with the new codebase, a technique using source control and branching strategies will be utilized to merge your changes into the new base code release.

The high-level process is made up of the following steps:

1. Complete any in-progress customization efforts and merge all change to a main or master source control branch.
2. Create or refresh the development branch with the code from the main branch.
3. Move to the ellucian branch. This branch and the development branch will be used to perform the merging.
4. Delete all source code in the ellucian branch.
5. Download and unpack the latest Ellucian source code into the ellucian branch.
6. Merge the ellucian branch into the development branch. This action will detect and merge the differences between your customized solution and the new codebase.
7. Resolve all merge conflicts from step 6.
8. Build and test the solution to verify your customizations and new functionality are working.
9. After all functionally is verified, merge development back to the main branch.

Merging the latest version of the Ellucian source code (or any external modifications of source code for that matter) with your revisions of the source code is one of the more difficult tasks that a developer must undertake when managing source code. However, source control systems provide several tools to aid in this process. In many cases, the tools of the source control system are able to automatically merge changes (usually the case for text-based files where the same
line(s) have not been modified in two files being merged), but there will almost always be cases where you will need to manually resolve merge conflicts.

Before going into the procedures, concepts and practices will be presented that can be used to make this process easier, reduce risk, and provide “undo” points if the merge starts to go wrong. The following assumes that a source control system, with branching capabilities, is being used to manage this source code.

**Don't merge into a master or main branch, use a development branch**

A best practice of development in a source control system that supports branching is to use, at a minimum, two branches: main/master and development. As the names begin to suggest, main/master is used to keep the latest, stable copy of your source code and development is used to make modifications to your source code.

After you complete your modifications in the development branch, you merge those changes to main/master. When updating your source code with the newest Ellucian source code, you should treat this process as just another development task and use your development branch when merging in the latest source code. This also allows you to recover if something goes wrong while merging in the Ellucian changes as you can undo the changes, if needed, and restart from main/master.

**Make use of a vendor branch where only revisions of the third-party code are kept**

Another common practice used when managing source code provided by a third-party vendor is to use an exclusive branch where only changes from the vendor are kept. In this way, your vendor branch (if you used Ellucian’s steps for setting up the source code using Git, this is the ellucian branch), in theory, always contains the latest software from Ellucian.

In order to merge your changes with the latest vendor changes, all you have to do is merge the vendor branch into your development branch and resolve the resulting merge conflicts. This also provides item history that can be tracked back to the vendor branch so you can see what changes came from the vendor. Vendor branches are commonly used when the source code provided by the third-party is not supplied using an upstream source control system (which is how this source code is being provided to you using the .zip archives).

Each time a new revision of the source code is released by the third-party, you delete all files in the workspace of the vendor branch, dump the latest source from the third-party into the workspace, and then commit the changes to the source control system. This, in a very simplistic manner, allows you to see which files have been added, modified, or deleted (a rename of a file is a delete of the old file, and an add of the new file).

The remaining topics of this section discuss some additional things to watch out for as you integrate your customizations with the latest release, and provide specific procedures if you are using Git as your source control system.
Specific items requiring careful merging

While careful merging of all source items is important, there are a few source items that require special attention while merging. The following list contains a handful of source items that require special attention to merging to ensure a successful and simplified upgrade experience. In a future release of the source code, more items will be added to this list, and detailed guidelines for merging each.

1. Visual Studio solution file (.sln)

The Visual Studio solution file (.sln) contains global metadata that tells Visual Studio which high-level items make up the solution, such as the projects. If a project reference is removed, it will not be loaded within the Solution Explorer when the solution is opened. Entries in the .sln will change when either you or Ellucian add new projects to the solution. Also, on occasion, the entries can become re-ordered.

2. C# project files (.csproj)

The C# project files (.csproj) contain metadata pertaining to each individual project within the solution. In particular, the .csproj files contain a list of all items within the project and all references to external assemblies and other projects. The items in the list will change as files are added and removed in the project. You will likely encounter conflicts here and should carefully inspect both source files being merged to ensure the correct merged result.

3. Assemblies and assembly debugging symbols (.dll and .pdb)

Assemblies and assembly debugging symbols (.dll and .pdb) files contain binary data and cannot be merged. Therefore, when you encounter a merge conflict, it is best to take the version of the .dll or .pdb that is being merged in and discard the current version. Realize that after the merge, if any of the .dls or .pdbs that were overwritten came from a dependent solution that you have also modified (such as the DTOs that are created within the Ellucian.Colleague.Api solution and copied to the Ellucian.Web.Student solution), you will have to rebuild those dependent solution(s) and re-copy the affected .dlls and .pdbs into the merged solution. If .dlls are out of sync, the solution will likely not build correctly.

4. Colleague Entity data contracts

The Colleague Entity data contracts define which fields are read from Colleague when using the Data Reader. If you and Ellucian add additional fields to a data contract, you will need to ensure the resulting merged contract contains the fields added within both source files. If you accidently leave out a field definition, the solution will fail to build.
Updating the Ellucian branch using Git

Use the following steps to update the ellucian branch of your Git repository with the latest source code. This procedure assumes that you used the Unpacking and Preparing the Source into Git procedure; if you are using Git and would like to start using this approach to track the source changes provided by Ellucian going forward, you may do so by simply creating an ellucian branch from your master branch at this time.

1. Open Git Bash and move to the Ellucian.Colleague.Api repository.
2. Checkout the ellucian branch.
   
   `$ git checkout ellucian`
3. Move into the source directory of the repository.
   
   `$ cd source`
4. Make a backup of the ellucian.license file. This must be done as the next step will delete this file (and it is not re-delivered in the new source code). Make sure your current working directory is the source directory!
   
   `$ cp
   ellucian.license`
5. Delete the Ellucian.Colleague.Api directory within the source directory. This will clear the way for the new source to be unpacked into the repository in the next step. Make sure your current working directory is the source directory.
   
   `$ rm -r Ellucian.Colleague.Api`
6. Unpack the latest source code into the repository using the PowerShell script provided with the latest source code. Make sure that you specify the source directory of the Ellucian.Colleague.Api repository when running the script. This step is run in Windows PowerShell and not Git Bash. Please refer to the README.txt file for full details on running the PowerShell script.
   
   `> .\EllucianColleagueApiSourceSetup.ps1 -DestinationDirectory
   "C:\Git\Ellucian.Colleague.Api\source"`
7. Restore and remove the backed-up ellucian.license file.
   
   `$ cp ellucian.license
   `$rm ellucian.license"
8. Update the repository's branch index to match what is currently in the workspace. This will detect any modified or deleted files.

   $ git add -u

9. Add any new source files that were added to the workspace.

   $ git add .

10. At this point, all changes in the ellucian branch have been staged and are ready to be committed. Commit the changes, making sure to provide a commit message that specifies the release version. At this time you should also tag the commit with a version number for quick future reference.

    $ git commit -m "Ellucian Release 1.2"
    $ git tag v1.2

At this point the ellucian branch has been updated to the latest delivered version of the source code. The remaining procedure will merge these latest changes with your customizations.

Integrating your customizations with a new release using Git

The following text and steps detail the process of integrating your customizations with the latest Ellucian release.

Note: The procedure below assumes that you have already completed the Updating the ellucian branch using Git procedure.

The high-level process involves merging the source contained in the ellucian branch with that contained in your current development branch. If you currently have no active development branch(s) (your latest source is in the master branch), you should create a development branch in order to perform the code merge, as you will want to verify and test the resulting merged code prior to promoting those changes to your master branch. It is also recommended, if new to Git or source control systems, to read chapter 3.2 of the Git-SCM book: Git Branching - Basic Branching and Merging (http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging) for what to expect when merging and what merge conflicts will look like.

Prior to performing a merge, you will want to install and configure a visual merge tool for use with Git. Neither Git or Git for Windows comes pre-packaged with a visual merge tool. There are several choices available, such as KDiff3, DiffMerge, and Meld to name a few. For each tool, you will have to install the merge tool, and then configure Git's .gitconfig file to use the tool. The configuration of each tool is slightly different, but all involve defining the merge tool within the .gitconfig. It is recommended that you configure the machine-specific .gitconfig (typically in the Documents and Settings/Users directory) instead of the project specific .gitconfig, so that each developer at your institution can use a different merge tool if desired.

This procedure will use the Git Gui tool (installed along with Git for Windows) instead of the command-line Git Bash to perform the merge as the gui view is easier for those new to Git. If you are experienced with merging in Git and want to use the command line, you may do so as well.
1. Open Git Gui for the Ellucian.Colleague.Api repository. You can do this by either opening Git Gui from the Start menu and navigating to the repository directory, or by right-clicking on the repository directory in Windows Explorer and choosing Git Gui from the context menu.

2. Check out your development branch by choosing Checkout... from the Branch menu. The Checkout Branch dialog will be displayed.

   ![Checkout Branch dialog]

3. Using the Checkout Branch dialog, select the development branch and click Checkout.
4. To begin the merge, select **Local Merge from the Merge** menu. The Merge dialog will be displayed.

![Merge dialog](image1.png)

5. Using the Merge dialog, choose the ellucian branch and click **Merge**.

6. A message window will be displayed with the results of the merge command. Before continuing, right-click and choose **Copy All** and paste this text into a text document for reference.

![Message window](image2.png)

**Error: Command Failed**
Git will auto-merge as many of the files as it can and state which conflicts it was unable to automatically resolve. If there were no merge conflicts, the dialog will show that the command succeeded, but likely the merge will fail with unresolvable conflicts as shown above.

Note: At any point, prior to committing the merge changes, you can abort the merge and start over by selecting **Abort Merge...** from the Merge menu.

7. The merge process is not complete until all merge conflicts are resolved. During a merge, the branch is placed into a “merging” state where unstaged changes represent files that have a merge conflict and must be resolved, and staged changes represent the files that were auto-merged. Using Git Gui, you can select files from either the Unstaged Changes or Staged Changes lists and see the file, along with its status.

The next step in this procedure will begin the merge conflict resolution, but first you should understand how Git labels the source files that will be used during the merge conflict resolution.

**Base:** The base version of the file (if available) represents the version of the file that is the common ancestor of the two conflicting file versions.
Local: The local version of the file represents the version of the file that is currently in the development branch.

Remote: The remote version of the file represents the version of the file from the ellucian branch.

8. For each file in the Unstaged Changes list, do this:

8.1 Select the file and based on the file type, do the following:

**Binary Files.** For binary files, you must choose either the file version from the local (development) or the remote (ellucian) branch, as you cannot combine portions of two binary files into one. It is recommended that you always take the remote (ellucian) version of the binary file when the binary file is an assembly (.dll). For other binary files (such as images), you will need to determine which file to keep.

**Text-Based Files.** For text-based files, you can combine portions of each source file to preserve your customizations and the changes made by Ellucian. This is best accomplished using the merge tool that you configured for use with Git. The merge tool will allow you to select with portions of local (development) and remote (ellucian) file versions you want in the final file.

8.2 Resolve each conflict by right-clicking within the file preview pane and choosing the appropriate merge action based on the type of file being resolved:

- **Run Merge Tool.** Select Run Merge Tool to launch the merge tool to resolve conflicts in text-based files.

  - **Note:** After resolving the conflicts in the merge tool (depending upon the external merge tool) you may need to manually stage the file for commit by clicking on the file in the Unstaged Changes list and selecting *Stage To Commit* from the Commit menu.

- **Use Remote Version.** Select Use Remote Version to use the remote (ellucian) branch version of the binary or text-based file.

- **Use Local Version.** Select Use Local Version to use the local (development) branch version of the binary or text-based file.
9. After all merge conflicts have been resolved, conclude the merge by committing the merged changes. As-is, the commit message in Git Gui will note all of the conflicts that occurred and that you resolved during this merge. It is best to use this text as the commit comments for the merge commit.

10. Open the solution in Visual Studio to verify the functionally of the solution. You should build, run unit test, and then run the solution in debug mode to verify functionality, making any changes just like you normally would during the development process, if needed.
This section provides a collection of various procedures that other sections of this manual ask you to perform.

Installing the InstallShield version of Colleague Web API

This procedure will list the steps for installing and configuring the Colleague Web API on a web server.

Installing the InstallShield version of the Colleague Web API is necessary to obtain your version of the ellucian.license file (which is created only by the InstallShield) and also creates the IIS website that you will deploy your customized solution to. Furthermore, this procedure helps establish that the unmodified Colleague Web API functions correctly with your Colleague environment prior to modifying any source code.

All of the steps in this procedure reference installation and setup steps from the Setting Up Colleague Web API manual.

Procedure

1. Complete the steps for prepping the web server in the Preparing for Installation and Web Server Preparation Steps sections.
2. Ensure the correct Colleague and Envision software updates are installed by reading the Installing Software Updates section.
3. Complete the steps for installing a new website in the Installing Colleague Web API section.
4. Complete the configuration of the Web API as described in the Configuring the Web Service Parameters and Configuring Colleague Web API sections.
5. Be sure to test the connection to the Colleague Environment from the Web API configuration web page to ensure the setup is correct and that the Web API can communicate with Colleague.
Adding your ellucian.license to the solution

This procedure will guide you through the process of adding your ellucian.license file to the Ellucian.Colleague.Api solution.


Procedure

1. In IIS Manager, expand the Colleague Web API website node.
2. Expand the ColleagueApi application node.
3. Right-click on the App_Data folder and choose Explore.
4. Copy the ellucian.license file to your development PC.
6. Right-click on the Ellucian.Colleague.Api project (under the Web Service folder) and choose Open Folder in Windows Explorer.
7. Copy the ellucian.license file to the App_Data directory shown in this directory.
8. Return to Visual Studio, right-click on the App_Data folder under the Ellucian.Colleague.Api project and choose Existing Item from the Add menu.
9. In Windows Explorer, navigate to the Ellucian.Colleague.Api/App_Data directory, select the ellucian.license file, and then click the **Add** button.

10. Return to Visual Studio, right-click on the ellucian.license file that was just added, and choose **Properties**.

11. In the properties dialog, change the **Build Action** from *None* to *Content*.

   ![Image of properties dialog]

12. In Visual Studio, right-click on the Ellucian.Colleague.Api.Tests project and select the **New Folder** from the **Add** menu. Name this new folder *TestData*.

13. Right-click on the *TestData* folder created in step 10 and add another folder named *App_Data*.

14. Right-click on the *App_Data* folder created in Step 11 and choose **Open Folder in Windows Explorer**.

15. Copy the same ellucian.license from Step 4 to this directory.

16. Return to Visual Studio, right-click on the *App_Data* folder created in Step 11 and choose **Existing Item** from the **Add** menu.

17. In Windows Explorer, navigate to the Ellucian.Colleague.Api.Tests\TestData\App_Data directory, select the ellucian.license file, and then click the **Add** button.

Configuring the solution’s Colleague connection parameters

This procedure will list the steps needed to configure the Ellucian.Colleague.Api solution’s connection parameters so that you can run and debug the solution using Visual Studio. Two files within the Ellucian.Colleague.Api project must be edited: web.config and App_Data\settings.config. These steps assume that you have installed and configured a working version of the Colleague Web API using the InstallShield.

Procedure

1. On the development web server, open IIS Manager.

2. Right-click on the API development website and choose Explore.

3. Open the web.config located in this directory in a text editor.

4. Copy the machineKey element of the system.web element to the clipboard.

   <system.web>
   <machineKey decryptionKey="C2C5C414D45E5E46C8EDC54BF57B5CEB634F7FF48A54710D"
validationKey="429E43FDED01A2A540E8720CDD72CF3B303E032B4928DF556759DEC1F1DC06175B317FOCED44DC25BCB3873BA64E262D4C1541EDC695113A98E293C8C2246A51" />
   </system.web>

   Note: Word wrap has been enabled in this screenshot in order to show the entire machineKey element. When pasting this text in the next step ensure that the attributes of the machineKey element do not contain a whitespace from the word wrap formatting.

6. Replace the `machineKey` element in the Ellucian.Colleague.Api’s project’s `web.config` with the `machineKey` element copied from the API development website’s `web.config` in Step 4.

```xml
</namespaces>
</pages>
<machineKey decryptionKey="C2C5C414D45E5E46C8EDC54BF57B1"
customErrors mode="Off"/>
</system.web>
```

7. Save the `web.config` file in Visual Studio.

8. Return to the development web server and open the `ColleagueApi` directory that was in the same directory as the `web.config` file.

9. Within the `ColleagueApi` directory, open the `App_Data` directory.

10. Open the `settings.config` file in a text editor and copy the entire settings XML to the clipboard.

12. Replace the entire settings XML with the XML copied in Step 10.

```xml
<settings>
  <colleague>
    <accountName>dv1805main_wstst01_rt</accountName>
    <ipAddress>sd18w8appint.datatelsdd.com</ipAddress>
    <port>6006</port>
    <secure>False</secure>
    <certificateHostnameOverride/>
    <connectionPoolSize>1</connectionPoolSize>
    <sharedSecret>ES2F745C55943B073E12B5FF7C81C264F4636348E9E4B13272A;</sharedSecret>
  </colleague>
  <logLevel>Disabled</logLevel>
</settings>
```

13. Save all in Visual Studio and build/debug the solution by pressing F5. The API Explorer page should be displayed. If not, you may need to set your startup project in Visual Studio to be the Ellucian.Colleague.Api project.

14. Select the Configuration Settings link at the top of the page.

15. Re-enter the Shared Secret on the configuration page.
16. Test the settings by using the Test Connection feature of this page.

17. Click Save on the configuration page.

Shared dependent assembly updates

Adding attributes to data transfer objects, or adding new data transfer objects affect all consumers of the Web API downstream. One of the primary consumers of the Web API is the Colleague Self Service solution.

If you use the Colleague Self Service solution, you need to pay careful attention to the fact that Ellucian.Web.Student has direct dependencies on assemblies from the Web API. When any of the shared assemblies change in the Web API, new versions of the assemblies must be moved to the Colleague Self Service Solution, Ellucian.Web.Student.

For example, if you add a new DTO to the Ellucian.Colleague.DtosStudent project, the Colleague Self Service solution will not “know” about it until a new version of the assembly Ellucian.Colleague.Dtos.Student.dll is moved to the Ellucian.Web.Student Dependencies folder.

When making changes to the Web API DTOs or to the API client, use the following instructions to move the updated assemblies from one solution to the other.

1. Make the changes to the Web API as normal and make sure the Ellucian.Colleague.Api solution builds and all unit tests are successful.


Right-click on the Ellucian.Colleague.Api solution node and choose Open Folder in Windows Explorer. This will open up the file system path where the solution lives.

3. Navigate to the Ellucian.Colleague.Api.Client\bin\Debug folder. This will be the folder where the assemblies are copied from.

5. Navigate to the *Ellucian.Web.Student/Dependencies* folder. This will be the folder where the assemblies are copied to.

6. Within the *Ellucian.Colleague.Api.Client/bin/Debug* folder, highlight the Colleague.Api.Client.dll assembly and all other Ellucian.Colleague.Dtos.* assemblies. (If you are absolutely certain that you know exactly which assemblies were modified during rebuild, then you can restrict your file selection.)

7. Copy the highlighted files to the Dependencies folder under *Ellucian.Web.Student*. You will be prompted to overwrite, so choose **Copy and Replace**.


9. Your DTO and Colleague API client changes are now available for local testing.

10. When testing is complete and you are satisfied with all testing, you will need to check in the assembly change for *Ellucian.Web.Student* to source control, in addition to all the other code changes.