API for SIS Second Steps:
Set Up Web Services Integration

eCollege
4900 South Monaco Street
Denver, CO 80237

Contents

About this document

If you are reading this document, you have completed reading *API Integration Choosing Your Solution* and have selected to use web service integration. This document assumes you have someone at your institution that is familiar with XML and web services.

How this document is organized

This document is organized into the following sections:

- **An overview of web services** – This section reiterates the purpose of the web service, the advantages of using web service calls and XML to transfer your data, and accessing web services.
- **Initial setup with client services** – This section discusses the initial setup of web services, including a description of the information your Client Services representative will provide and the information you will provide to your Client Services representative.
- **Getting started** – This section discusses synchronous vs Asynchronous web service calls, available web services, web security, and web security errors. Additionally, this section provides code examples for your reference.
- **Data format for web services** – This section discusses in detail what information the web services are expecting and in what format; and provides the location of the eCollege web services WSDL and URI.
- **Testing the implementation** – This section provides instructions for you to test your web services integration.
- **Checking the request status** – This section discusses how to check the status of your message requests.
- **Appendices** – The appendices in this document include samples of auto-generated email and XML data, an overview of XML, and common XML errors.

Overview of web services

A web service essentially enables one web-based application to make use of data that is inside another web-based application whether the two systems have any "knowledge" of each other or not. In this case, the web service allows your Student Information System (SIS) to interact with the eCollege system. By using the eCollege SIS API web service integration, you can automatically update your users' information within the eCollege system whenever you create, modify, or enroll a user in your Student Information System, without having to go through an additional manual process.

Purpose of web services communication

eCollege's API for SIS provides an efficient method of transferring user and enrollment information between your student information tracking system and the eCollege system, regardless of your underlying system for housing student information. The goal of the API for SIS is to provide one system of record so that user creation and enrollments in either your SIS or the eCollege system are reconciled in real time (synchronously) or within 24 hours (asynchronously).

Specifically, you can use the eCollege API for SIS to:

- create users in your online program
- automatically enroll users in courses
- update personal information, like names and email addresses, as well as enrollment statuses (like Add or Drop)

As a result, the purpose of the web service communication is to provide a link between your SIS and the eCollege database so that any changes made to your system are automatically made in the eCollege system as well. When the web service request is done processing, the state of the eCollege system will match the state of your SIS system.

During times of high activity, such as the rollout of your online campus or the beginning of a term, the API for SIS allows you to create and enroll tens of thousands of students in a single operation. The synchronous API integration option automatically updates user and enrollment information for a single user in your online campus as soon as a change is made in your SIS.

Advantages of using web services and XML

Integrating your SIS with the eCollege API using web services gives you several advantages over the more traditional FTP or manual data upload methods:

- **Synchronous updates**. Single user requests are handled in near real time, meaning that as soon as a change is made in your SIS, that change is automatically made in the eCollege system without any additional action from your end.

- **Extended user properties**. Formatting your user data in XML gives you the option of using extended or custom user properties (for example, user program, user start date, campus location, user's assigned advisor, military service, maiden name, etc.).

- **Editing User Login information**. Only XML data allows you to change a user's login after it's been created.

- **Maximum integration flexibility**. Because they use standard XML, web services are platform and language independent. As a result, an SIS programmed in C++ running on Windows can still consume a web service programmed in Java running on Linux.

Accessing web services

It's important to note that information made available through a web service is always accessed by software, *never* directly by a person, and that even though web services depend on existing web technologies, they have no relation to web browsers and HTML. That said, however, web services do have a URI through which they are consumed. The URI (or address) for the eCollege SIS API can be found at https://api.ecollege.com/usermanagementServices.asmx.

Once you know where the web service is located, you need some instructions for how the service should be called upon, or invoked. XML web services provide a way to describe their interfaces in enough detail that a user can build a client application to talk to them. This description is provided in an XML form using the Web Services Description Language (WSDL).
The WSDL explains how the web service should be invoked. The WSDL (or invocation instructions) for the eCollege SIS API can be found at
https://api.ecollege.com/usermanagementServices.asmx?WSDL

Initial setup with Client Services

Your Client Services representative will need to provide you with some information as you begin your integration. Similarly, your Client Services representative will need to collect information from you to ensure that the integration process for your institution is set up correctly from the eCollege side. The table below summarizes what information you can expect to receive from your Client Services representative and what information you'll need to provide to your Client Services representative.

API Setup Information	Description
Your Client Services representative will provide:	
Username and password	eCollege requires a username and password to access the web services server. Your Client Services representative will provide you with your security credentials. For more information, see Web Service Security.

API Setup Information	Description
You will provide your Client Services representative with:	
Using Division Tools?	**Yes or No.** *This is only relevant for institutions using Division Tools.* If you're not using Division Tools, this value is "No."
Send Student Email?	**Yes or No.** The API can be configured to send users autogenerated emails based on either account creation or enrollment in a course. Emails can be differentiated by enrollable node, role type (i.e. Student Email, Faculty Email) or action (i.e. Enrollment Confirmation, Drop Confirmation, Waitlist, etc.). These emails can also pass the following user-specific information:

	• User Name • User Login ID • User Password • Course(s) Enrolled Auto-emails must be configured during the initial set-up of the API, and can be updated by your CS representative. You can create your own email "templates" and send them to your CS representative, or you can use the eCollege template, available in Appendix A, as a starting point.
Generate User Passwords?	**Yes or No.** The API can be configured to auto-generate random passwords for all new users if you leave the password field blank. The auto-generated password will be included in the instruction email sent to the user (if you've chosen to send student email alerts). However, if a password is supplied, it will be used.
Update User Properties?	**Yes or No.** The API can be configured to update user properties, based on each user's login ID. If this option is set to "yes," any changes in name, password, email address, or any of the optional user property fields will result in updated information within the eCollege system when the API file is processed. If this property is set to "no" or "off," any properties differing from what is in the eCollege database will **not** be updated when the API request is processed. Thus, if you expect user information to be updated through the eCollege system, and that information is not also updated in your SIS, it is possible that updated information in the eCollege system will be replaced by "old" information from your SIS during the integration process. If changes are made only rarely in the eCollege system, and your SIS is always up to date, then this risk is minimal. Similarly, if your SIS only sends incremental data (i.e., only records that have changed since the last integration), rather than snapshot or cumulative data, there is little risk.
If **not** updating user properties, source for student email address?	**SIS or eCollege database.** If you've chosen to send students email notifications (upon user creation or enrollment, for example) but you selected above **not** to update user properties, there's a chance that the email address in the eCollege system may become out-of-date. As a result, you need to decide the "default" system of record from which student email notifications should come—send notifications to the student email address in your SIS file or to the email address in the eCollege database?
Email EP?	**Yes or No.** This option sends an auto-generated summary email to the submitter or to an email alias you specify when the file has been processed. One email is sent per API request. Note that synchronous requests sent via web services will **NOT** receive a summary email, regardless of this setting. This is because of the potential of receiving dozens or hundreds or even thousands of emails in a single day, depending on the number of real-time requests processed.
If yes, EP Email Address?	This is the email alias you would like automatic updates sent to.

Getting Started

Knowing where the web service is located and how to call it (Accessing web services), you are ready to begin your integration. Invoking a web service involves passing messages between the client and the server. Messages are passed either synchronously or asynchronously. Although there are several transfer protocols, for the purposes of the eCollege web services you will be invoking the web service call using a standard web protocol, SOAP (Simple Object Access Protocol). The SOAP protocol specifies how to format requests to the server, and how the server should format its responses. The information contained within the body of the SOAP message is encrypted using SSL to secure communications between your SIS and the eCollege server. (More about web services security is discussed later in this document.)

The following sections will

- Differentiate between synchronous and asynchronous web service calls
- Define, specifically, what web services are available
- Explain in general terms web service security

Synchronous vs. Asynchronous web service calls

The eCollege SIS API provides a set of direct web service interfaces that allow your SIS to "talk to" the eCollege system in either real-time (synchronously) or at regularly scheduled intervals (asynchronously).

Establish whether your system will transfer data in bulk or in single operations to determine whether a synchronous or asynchronous web service call is the best method for you. When the web service request is done processing, the state of the eCollege system will match the state of your SIS system.

Synchronous web service calls

The synchronous web service integration option offers the best solution for a tightly integrated, near-real-time solution between eCollege and your SIS, as it allows you to automatically update user and enrollment information in your online campus as soon as a change is made in your Student Information System. Synchronous communication can only contain one user and one course enrollment for that user; however, you can send as many communications as you want; each will be created or updated in real-time.

Asynchronous web service calls

During periods of high activity, such as when your online campus is starting up or at the beginning of a semester, you can use the API for SIS web service to perform bulk operations, creating and enrolling tens of thousands of students in a single operation. In this case, your SIS contacts the web service on the eCollege system with IMS Enterprise data that can be processed within a 24 hour window. This process—from initial request through final result—affects multiple users and is an **asynchronous** operation.

What web services are available?

The web service used to process data from your system is determined by the web service call (synchronous or asynchronous) you are using. Currently, there are two web services that you can consume:

- **ProcessSingleRequest**
- **ProcessRequest**

ProcessSingleRequest

This is a synchronous web service call for processing a single Person element. The ProcessSingleRequest web service allows a single user to be created, updated, and enrolled in courses and nodes in near real-time. Thus, as soon as a change is made to your SIS, the web service automatically processes the information and updates the data in the eCollege system almost instantaneously. Although the ProcessSingleRequest web service only accommodates one user per web service call, because the process is synchronous, an infinite number of requests can be sent and processed at any given time. Expected IMS Elements for the ProcessSingleRequest web service:

- Person
- Group
- Membership

Returns: the received XML message with additional messages indicating the success or failure of any operations performed.

ProcessRequest

This is an asynchronous web service call for batch processing information from your SIS. This request handles multiple users (maybe two, or maybe thousands) in a single operation. Users are created, updated, and enrolled in courses and nodes as appropriate. Although this web service request occurs in real-time, the eCollege system actually acts on the request at a later time—within 24-hours from the time the web service was invoked.

Expected IMS Elements for the ProcessRequest web service:

- Person
- Group
- Membership

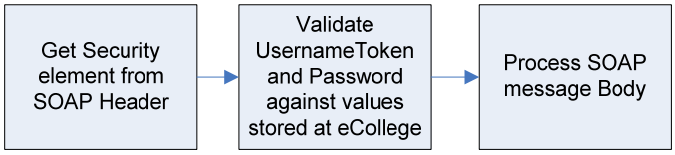
Returns: a Job ID, which can be used in the eCollege Administrative Pages to locate and identify the status of the request and its results.

Web Service Security

eCollege requires a username and password to access the API web services. In addition, the API software performs various internal checks on the data to prevent accidental or deliberate modification of data. Your Client Services representative should have provided you with the necessary login credentials.

Web services will be secured both by securing the pipe (HTTPS) and by authenticating the user via WS-Security with the username and password provided by your Client Services representative. By applying the WS-Security standard, eCollege can authenticate the source of each request. The eCollege API will only accept web service requests that can be authenticated using the WSS UsernameToken. The web service will refuse any requests where the WSS Username token was either not supplied or was invalid. The password must be of the PasswordDigest type. The use of the Nonce and Created elements is optional, but encouraged. The PasswordDigest type protects the password from being revealed, and the Nonce and Created elements help protect the exchange of student information from replay or the ability of someone to capture a legitimate exchange data and attempt to parts of it to spoof the system.
To help protect any passwords or confidential student information contained within the body of the SOAP message, communications between your SIS and the eCollege server will be encrypted using SSL. For more information about the WS-Security specification, refer to the Web service implementation resources section in this document.

Visually, the authentication process looks like this:



Implementing the WS-SEC library

Correctly implementing the WS-Security standard can be tricky and time consuming. For this reason, eCollege recommends using a library or similar resource where the WS-Security standards have already been implemented for your programming language of choice. For a list of libraries and resources, see Web service implementation resources.

The following sections describe specific elements taken from the WS-SEC library and applied to a SOAP message.

WS-Addressing

The WS-Addressing section must include the following elements:

- Action
- To
- MessageID

Element	Description
wsa:Action	This is the particular soap action. o The value should be either **http://api.eCollege.com/pa/UserManagement/ProcessSingleRequest** --OR-- **http://api.eCollege.com/pa/UserManagement/ProcessRequest** o The value must match the value of the HTTP SOAP Action header. o Note that the value is not quite the same as the URL used to access the web service. This is because the namespace assigned to the web service *method* contains some additional levels.
wsa:To	This is the actual URL of the web service. o The value is **https://api.ecollege.com/UserManagementServices.asmx** o Note the difference from the Action, which specified the method as well as other levels of the eCollege namespace hierarchy.
wsa:MessageID	This is the ID for the message. The MessageID is assigned by the person sending the message.

WS-Security

The WS-Security section of the SOAP message header must include the following elements:

- Created
- Expires
- UsernameToken

Element	Description
wsu:Created, wsu:Expires	These are the timestamp elements. Timestamp elements specify the date range for which the security element is still valid. o The date/time value complies with the ISO 8601 standard. o The combined date and time look like **2007-05-01T15:05:01-06** --OR-- **2007-05-01T21:05:01Z** ▪ Note that the combined date and time designates the Time portion by inserting the "T" between the date and time. ▪ Because the SIS and eCollege are likely to be in different local time zones, it is important to give time according the UTC – either in UTC, as designated by the "Z" at the end, or by providing an offset for the local time to UTC. Both of the examples give 3:05 in the afternoon of May 1st 2007 in Denver, Colorado. ▪ The wikipedia entry on the ISO 8601 standard provides more information on the standard.
wsu:UsernameToken	This is the hashed password supplied by your Client Services representative. o The password must be of the password digest type (note the value of the Type attribute in the example) o WS-Security specifies that the hash is the Base 64 representation of a SHA-1 hash of the password, as well as the Nonce and Created values if those are being used. o The nonce value must base64 encoded and 16 bytes in length. When generate the password value, ensure the nonce used is the raw bytes and not the base64 encoded bytes. See the specification on the Oasis group's Web site for more details. The basic formula is: `Password Value = Base 64(SHA-1(Nonce + Created + Password))`
wsse:Nonce	A random value used in the password hash to prevent replay attacks. o The Nonce should be generated using a cryptographicly secure random number generator or similar source. o A Nonce can only be used once. Reusing a Nonce will result in a security error. (Number Once).

Code Examples

This section contains code examples for the following:

- Implement WS-SEC library
- Username Token in the Header of a SOAP message

Example - Implement the WS-SEC library

The following example contains the source code & compiled executable for a basic Windows Form written in C# that implements the WSE2 library. See also Web service implementation resources.

Note: At a minimum eCollege requires that you use the Username Token with the hashed password. It is strongly recommended that you use the Nonce and Created timestamp. Because of the software libraries that eCollege uses to validate the WS-Security portion of the soap headers, you will need to include additional information. In the UsernameToken, the Password must be the digest type.

Use the following legend to interpret portions of code in this example.

Legend

Blue	Basic soap elements tags.
Purple	Addressing soap header elements
Green	Security soap header elements
Orange	eCollege web service method element
Pink:	Root element for the IMS Enterprise v1.1 data*
Black	IMS Enterprise v1.1 data
Red	Some key data for authenticating the request, identifying the user and identifying the course.

[NEED HTTP HEADERS]

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <soap:Header>

    <wsa:Action>http://api.eCollege.com/pa/UserManagement/ProcessSingleReques
t</wsa:Action>
        <wsa:MessageID>uuid:8d0c3e2e-e789-460b-b317-
d5edf3bb3735</wsa:MessageID>
        <wsa:ReplyTo>

    <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonym
ous</wsa:Address>
        </wsa:ReplyTo>
        <wsa:To>http://api.ecollege.com/UserManagementServices.asmx</wsa:To>
        <wsse:Security soap:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="Timestamp-b23cf78f-09f8-4eec-9106-
3a9a52819842">
                <wsu:Created>2007-04-23T22:37:12Z</wsu:Created>
                <wsu:Expires>2007-04-23T22:38:12Z</wsu:Expires>
            </wsu:Timestamp>
```

Use the following legend to interpret portions of code in this example.

Legend

Blue	Basic soap elements tags.
Purple	Addressing soap header elements
Green	Security soap header elements
Orange	eCollege web service method element
Pink:	Root element for the IMS Enterprise v1.1 data*
Black	IMS Enterprise v1.1 data
Red	Some key data for authenticating the request, identifying the user and identifying the course.

```
            <wsse:UsernameToken wsu:Id="SecurityToken-8836043e-8f9b-4e56-
8e2b-5610e9ce280f">
                <wsse:Username>test</wsse:Username>
                <wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
1.0#PasswordDigest">BXlm1gHnmRQcyShNse5Ne6nQN8I=</wsse:Password>
                <wsse:Nonce>fYMGg4c/0tZbHZE2pAIOEQ==</wsse:Nonce>
                <wsu:Created>2007-04-23T22:37:12Z</wsu:Created>
            </wsse:UsernameToken>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <ProcessSingleRequest
xmlns="http://api.eCollege.com/pa/UserManagement">
            <enterprise
xmlns="http://schemas.ecollege.com/Common/2006/01/ims_epv1p1.xsd">
                <properties lang="">
                    <datasource/>
                    <datetime/>
                </properties>
                <person recstatus="1">
                    <sourcedid sourcedidtype="New">
                        <source>CUOnline</source>
                        <id>20070206a</id>
                    </sourcedid>
                    <userid password="somepassword">20070206a</userid>
                    <name>
                        <fn>Sam Pull</fn>
                        <n>
                            <family>Pull</family>
                            <given>Sam</given>
                            <partname partnametype="Middlename"
lang="">Harold</partname>
                        </n>
                    </name>
                    <demographics>
                        <gender>Male</gender>
                    </demographics>
                    <email>sam@nonexistent.com</email>
                    <tel>tel_1</tel>
```

Use the following legend to interpret portions of code in this example.

Legend

Blue	Basic soap elements tags.
Purple	Addressing soap header elements
Green	Security soap header elements
Orange	eCollege web service method element
Pink:	Root element for the IMS Enterprise v1.1 data*
Black	IMS Enterprise v1.1 data
Red	Some key data for authenticating the request, identifying the user and identifying the course.

```
                    <adr>
                        <street>123 Main Street</street>
                        <locality>Denver</locality>
                        <region>CO</region>
                        <pcode>80237</pcode>
                        <country>USA</country>
                    </adr>
                </person>
                <group recstatus="1">
                    <comments lang=""/>
                    <sourcedid sourceidtype="New">
                        <source>CUOnline</source>
                        <id>TERRA-101</id>
                    </sourcedid>
                    <grouptype>
                        <typevalue level="">Call Number</typevalue>
                    </grouptype>
                    <description>
                        <short/>
                    </description>
                </group>
                <membership>
                    <sourcedid sourceidtype="New">
                        <source>CUOnline</source>
                        <id>TERRA-101</id>
                    </sourcedid>
                    <member>
                        <sourcedid sourceidtype="New">
                            <source>CUOnline</source>
                            <id>20070206a</id>
                        </sourcedid>
                        <idtype/>
                        <role recstatus="1">
                            <subrole>2</subrole>
                            <status/>
                        </role>
                    </member>
                </membership>
            </enterprise>
        </ProcessSingleRequest>
    </soap:Body>
</soap:Envelope>
```

Example – Username Token in the Header of a SOAP message
Following is an example of the Username token in the Header of a SOAP message taken from the *Oasis* organization WSS documentation:

```
<S11:Envelope xmlns:S11="…" xmns:wsse="…">
    <S11:Header>
        …
        <wsse:Security>
            <wsse:UsernameToken>
                <wsse:Username>Zoe</wsse:Username>
                <wsse:Password Type="…#PasswordDigest">
                    weYI3nXd8LjMNVksCKFV8t3rgHh3Rw==
                </wsse:Password>
                <wsse:Nonce>WScqanjCEA40mQ0BE07sAQ==</wsse:Nonce>
                <wsu:Created>2003-07-16T01:24:32Z</wsu:Created>
            </wsse:UsernameToken>
        </wsse:Security>
        …
    </S11:Header>
    …
</S11:Envelope>
```

WS-Security Errors

Should there be a security error, or other problem with the SOAP message, the eCollege web service returns a SOAP message to your system with a SOAP fault detailing the particular problem. The eCollege server responds to your system with an HTTP status code of 500. Remember, when you initially set up your web services integration, it is helpful to capture the full response with the SOAP message, as well as the HTTP response. In some systems, the 500 status might terminate the HTTP response stream; thus, the program might not receive the details of the SOAP fault.

When you send your SOAP request to process user information, the web service will reply with a SOAP response indicating the status and outcome of your request (asynchronous web service calls will respond with a Job ID). If the SOAP request was incorrect, the web service will respond with an error message.

For example, in the case of a:

- **ProcessSingleRequest**, the web service response will contain the original IMS Enterprise data enriched with result elements in the extension for the person or membership/member describing the success or failure of any actions.

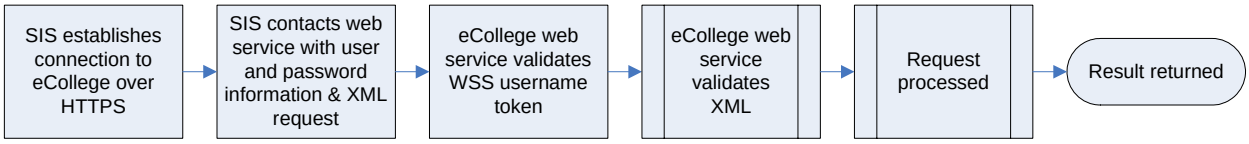


Figure 1: Synchronous application-to-application communication.

- **ProcessRequest**, the web service response will always contain a Job ID number. Additionally, processing state and result information will be available through the Administrative pages.

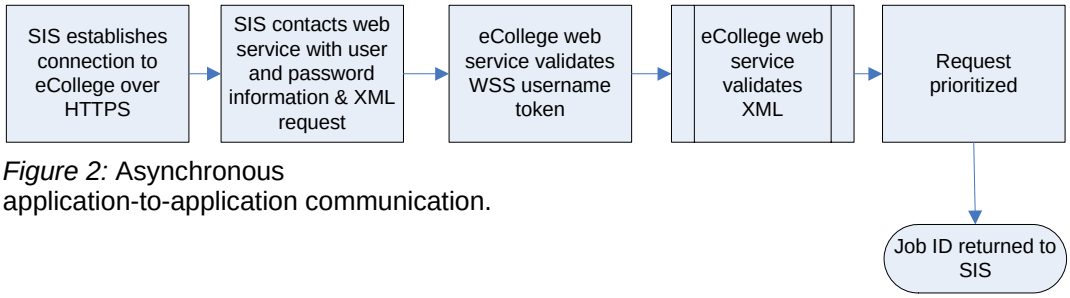


Figure 2: Asynchronous application-to-application communication.

Web service implementation resources

The following table lists links to the Oasis group standards and other resources.

Resource	Description
WS-Security	
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss	The general site for all WS-Security related information from the Oasis group.
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf	The core WS-Security 1.1 specification in pdf format.
http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf	The Username Token Profile 1.1 specification in pdf format.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	The schema for WS-Security
http://www.oasis-open.org/committees/ballot.php?id=407	The history of the WS-Security specification from the Oasis group
http://wso2.org/projects/wsf/php	An implementation of the Web Services Framework in PHP
WS-Addressing	
http://schemas.xmlsoap.org/ws/2004/03/addressing	The schema for WS-Addressing
http://www.w3.org/Submission/ws-addressing/	The W3C specification for WS-Addressing
Implementing WS-Security	
http://ws.apache.org/wss4j/#WS-Security%20Features	An implementation of the WS-Security standard by the Apache group using Java.
http://www.oracle.com/technology/products/id_mgmt/phaos/wss10/doc/usersguide.html#WSSAPI	Phaos is a Java implementation of WS-Security by Oracle. Specific details on their API can be found at http://www.oracle.com/technology/products/id_mgmt/phaos/wss10/doc/apidoc/index.html
http://java.sun.com/webservices/docs/2.0/xws-security/ReleaseNotes.html#limitations	Information on the Java libraries for WS-Security provided by Sun.
http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html	A Sun tutorial on implementing WS-Security with the Java Web Services Developer SDK 1.6
http://www.microsoft.com/downloads/details.aspx?familyid=fc5f06c5-821f-41d3-a4fe-6c7b56423841&displaylang=en	The source for the Web Security Enhancements 2.0 from Microsoft.

http://pluralsight.com/blogs/kirillg/archive/2005/04/13/7315.aspx	A blog entry on getting Microsoft's WSE2 and IBM's Websphere 6.0 to integrate.
http://msdn2.microsoft.com/en-us/library/ms954606.aspx http://msdn2.microsoft.com/en-us/library/ms998284.aspx	Microsoft documentation on interoperability between WSE 2 and Sun's JDWSP 1.4 and 1.5 respectively
http://msdn2.microsoft.com/en-us/library/ms998291.aspx	Microsoft documentation on interoperability between WSE2 and WebLogic Workshop 8.1.4
ISO 8601 Date & Time Information	
http://en.wikipedia.org/wiki/ISO_8601	A Wikipedia definition of ISO 8601 Date & Time Information

Data format for web services

Information transmitted from your SIS to the eCollege API via web services, whether synchronously or asynchronously, must be formatted as **Extensible Markup Language** (XML). The eCollege API for SIS uses the open IMS Enterprise Specification v 1.1 for input of User and Enrollment information. This standard allows you to clearly communicate user, course, and enrollment information in XML regardless of the underlying SIS system you may be using at your institution. The web services *only* support the IMS XML data format. Most likely, your SIS will generate XML for you.

The IMS Global Learning Consortium defines the technical specifications for interoperability of applications and services in distributed learning and supports the incorporation of the IMS specifications into products and services worldwide. IMS seeks to promote the widespread adoption of specifications that will allow distributed learning environments and content from multiple authors to work together. eCollege is a member of the IMS Developer Network. More information about the IMS is available at their web site, http://www.imsglobal.org/.

The web services *only* support the IMS XML format for the data. For a description of each of the element data structures you will use to build your XML document, please refer the **eCollege Annotated Guide to the IMS Specification**. This document can be accessed in PDF format from the SDK or is available from your Client Services representative.

The following sections will:

- Explain how the IMS Specification is interpreted
- Provide the location of the eCollege web services WSDL and URL

How does the eCollege SIS for API interpret the IMS specification?

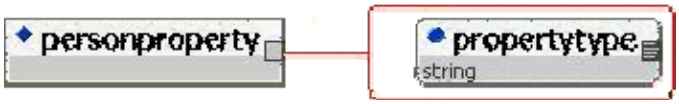
The eCollege web services support the IMS Enterprise v1.1 specification for data exchange. The schema diagrams below illustrate eCollege extensions to the IMS specification and the elements of the schema that the eCollege system is interested in.

*For a complete description of each of the element data structures you will use to build your XML document, please refer to the **eCollege Annotated Guide to the IMS Specification.** This document can be accessed in PDF format from the SDK or is available from your Client Services representative.*

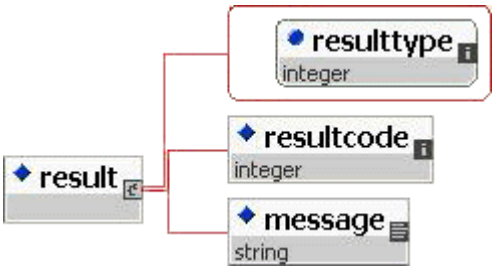
Note: In the IMS Schema files, only the `membership/member/role` element allows any form of cumulative data. Also, any missing (or extra) elements will be ignored by the SIS API.

eCollege Extensions to the IMS Enterprise Schema

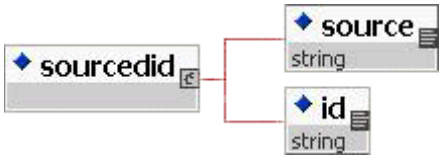
The `propertytype` attribute identifies an extended user property in the eCollege system:



The `result` element may be used in the extension of the person, group, or member to identify the results of processing that element. This includes error, success, and warning messages. The `@resulttype` attribute is an enumeration of the particular message types. The `resultcode` is a numeric code assigned to an error or warning message to help facilitate an automated response to particular problems. For success messages, the result code will be 0. The `message` is a description of the result intended for an end user.



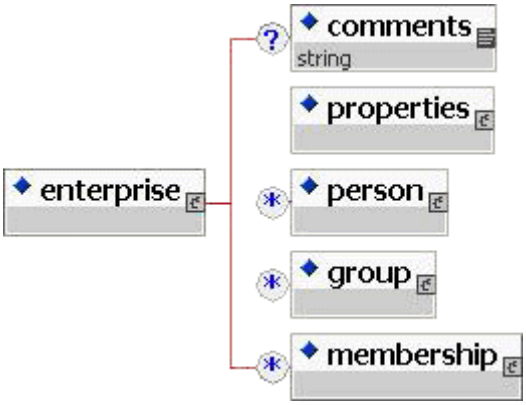
Sourced Identifiers



The SIS-generated sourcedid's will be maintained and mapped to the appropriate entity within the eCollege system. Note that a sourcedid may only identify one entity. The source for all eCollege elements will be ECLG.

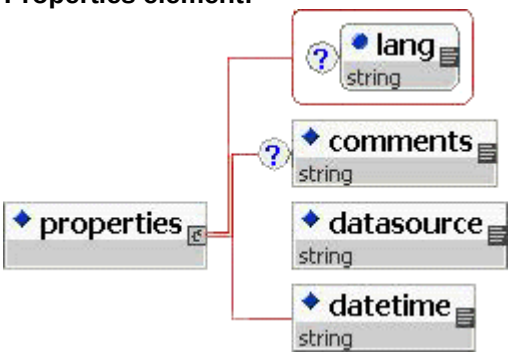
All eCollege sourced entities must expose a proxy key rather than an actual database key. However, pre-existing exceptions will continue for the convenience of users who may already have dependencies on those keys.

Schema top level
As defined by the IMS schema, the root element must be <enterprise>:



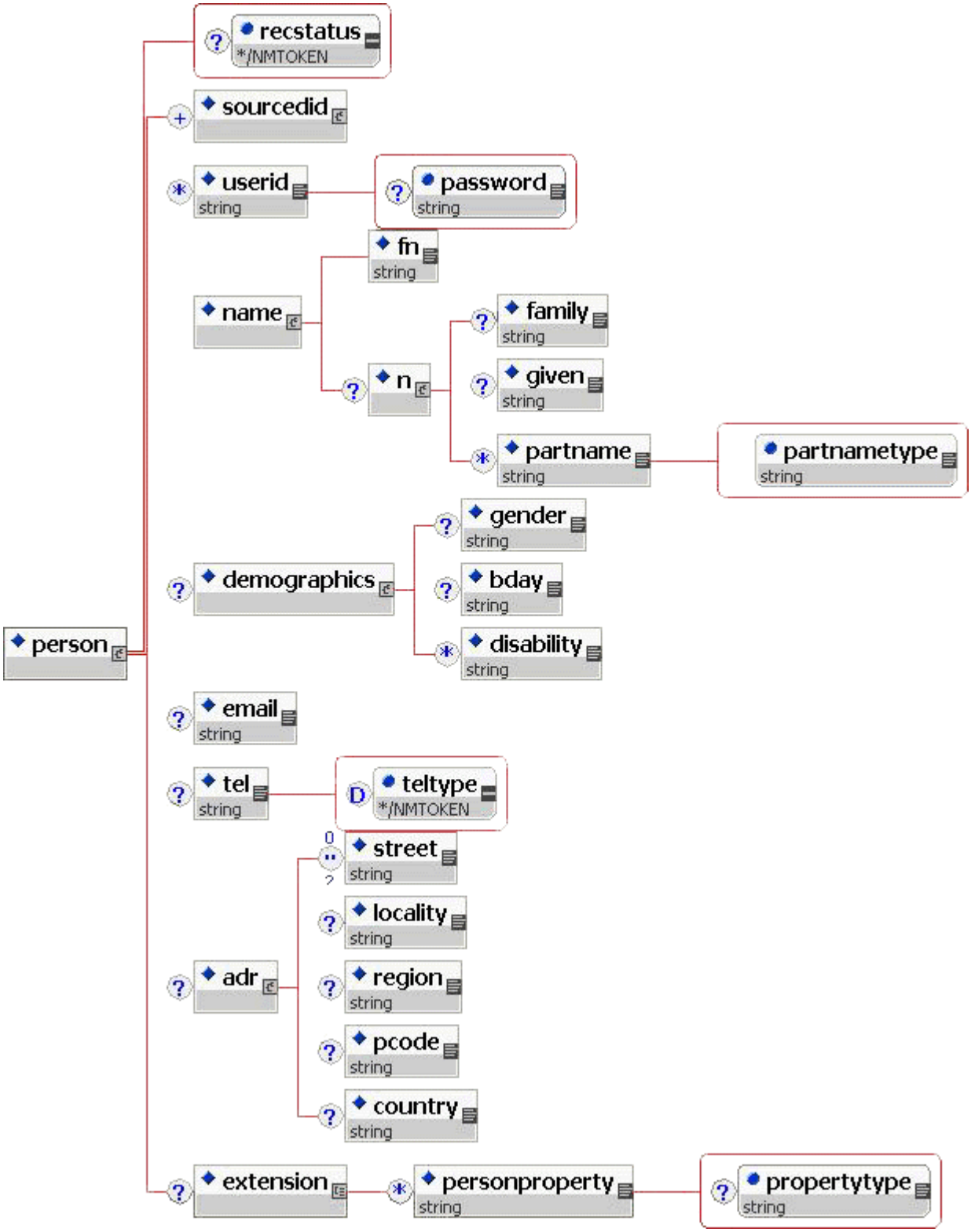
The `properties` element gives information about the data being sent to eCollege. The meat of the data is in the `person, group`, and `membership` elements. The comments element is generally ignored. The results of processing are placed as result elements in the appropriate `extension` element.

Properties element:



The `person` element defines all of the properties of a user, although nothing about their relationship to a course or to your online campus. The relationship is defined in the `membership` element, which describes which `persons` are members of which `groups`. The `group` may be either a course or a node.

Person element:



Elements in the schema map to user properties as follows:

IMS Enterprise Schema	eCollege data
userid	Login ID
password	Password (unencrypted)
name/fn	*ignored*
name/n/family	LastName
name/n/given	FirstName
name/n/partname	An extended user property where **only** Middlename is currently accepted. All other extended user properties are handled through personproperty.
demographics/gender	Gender
demographics/bday	Birthday
demographics/disability	Disability
email	Mail
tel	First telephone number only; will be homephone
tel/@teltype	*Ignored* (voice, fax, etc. are not recognized in the eCollege system)
adr/street	First instance, street1; second instance, street2; any additional instances are ignored
adr/locality	City
adr/region	State
adr/pcode	PostalCode
adr/country	Country

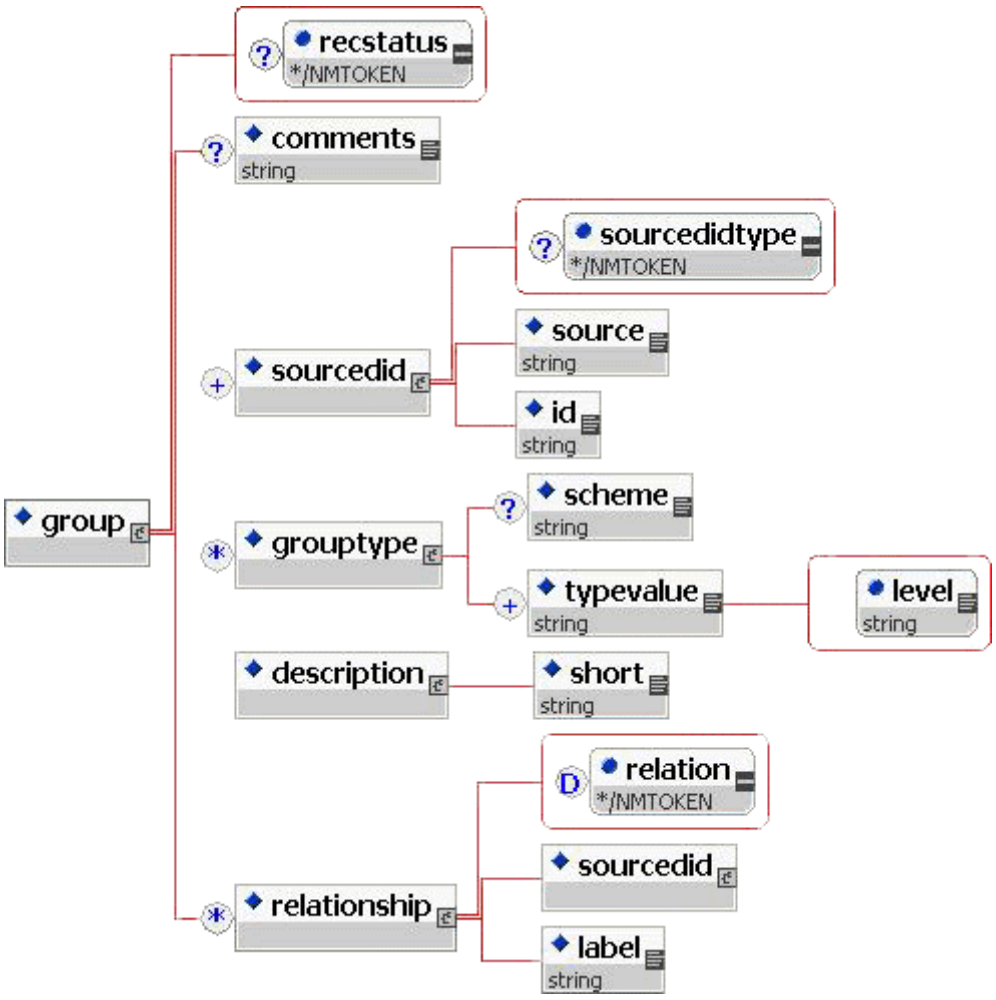
Any other extended user properties are supported through the eCollege extension `personproperty`. The value of the element is the value of the extended user property, while the `propertytype` attribute is the `propertyname` of the extended user property. For example, the extended property for a person's dog's name would be `<personproperty propertytype="dogsname">Fido</personproperty>`.

Group element

A `group` can be used to identify either a course or an enrollable node. The differentiation is through the `grouptype` element with a `typevalue` of "Call Number" or "Enrollable Node".

When identifying a course, the `sourcedid/source` is the client sort string (or Student Information System), and the `sourcedid\id` is the course call number. If an enrollable node must be provided, the `relationship` element is used to identify the enrollable node. Because eCollege is the source for an enrollable node, an additional `group` element is not required – only the sourcedid.

When identifying an enrollable node, the `sourcedid/source` must be ECLG and the `sourcedid\id` is the Client Sort String for the enrollable node.

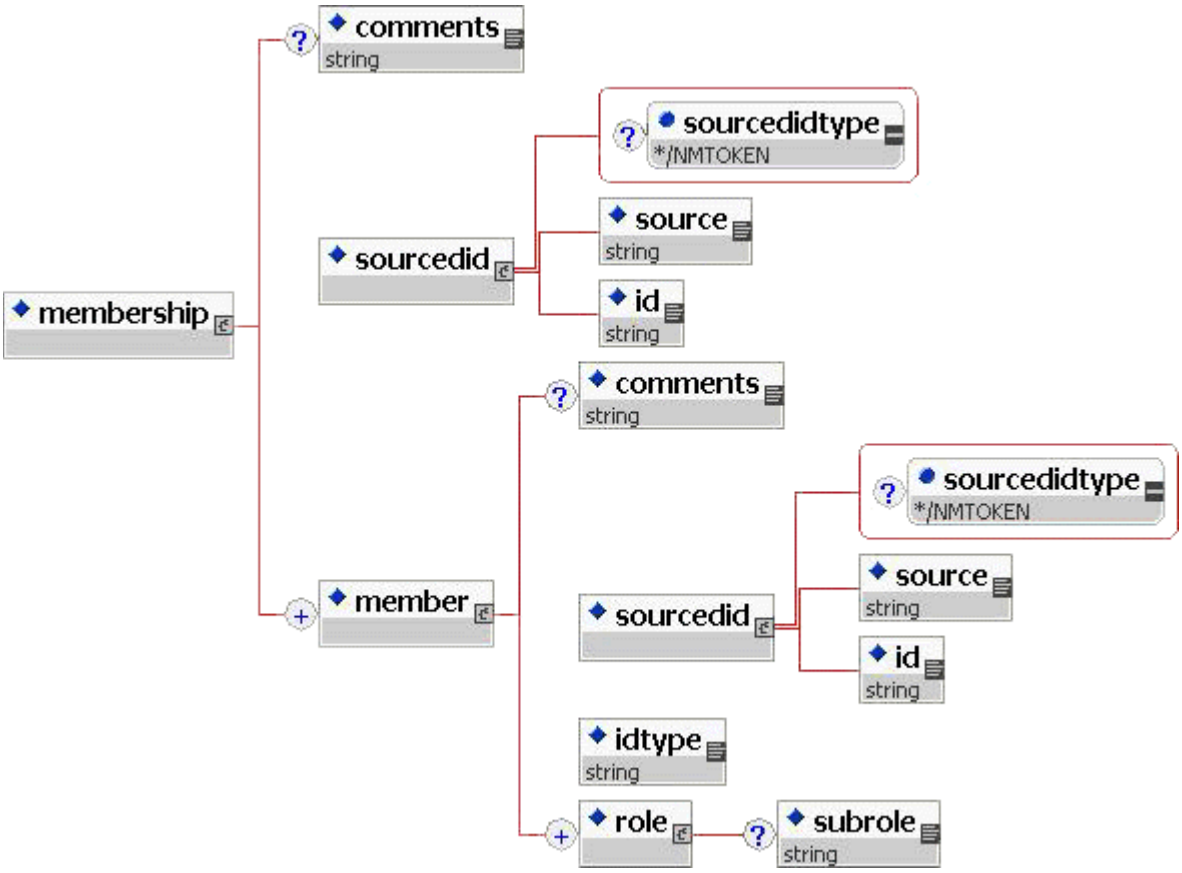


Elements in the schema map to group properties as follows:

IMS Enterprise Schema	eCollege data
recstatus	*Ignored*
sourcedid/id	EP Call Number *if* grouptype/typevalue is “Call Number” or Client Sort String *if* grouptype/typevalue is “Enrollable Node”

Membership element

The `membership` element is the key to using the IMS Enterprise specification for enrolling users in the eCollege system. It defines the relationship between a `person` and a `group` (course or node). `Member` elements are persons, as defined by the `membership\member\idtype`. The `membership\sourcedid` refers to a `group` elsewhere in the data, while the `membership\member\sourcedid` refers to a `person` element. The `role\subrole` defines the user’s eCollege role in the course or node. The `role` element itself cannot be used since it is insufficiently precise to identify an eCollege role, roughly corresponding to the eCollege Role Type. The value of the `subrole` is an eCollege RoleID.



Elements in the schema map to group properties as follows:

IMS Enterprise Schema	eCollege data
member/role/subrole	RoleID

Where can I see the WSDL?

Web services are self-describing. This means that once you've located the web service, you can ask it to "'describe itself" and tell you what operations it supports and how to invoke it. The **Web Services Description Language**, or **WSDL**, is a specific type of XML schema that defines a language for expressing web services in a way that common XML software can understand and use it. WSDL includes a description of the data types and structures used in web services messages, as well as information required for mapping the web service definition to an executing environment. Thus, the web service responds in the WSDL language with information about how the service should be invoked.

You can view the WSDL document for any .NET web service by visiting the .NET web service endpoint through a browser. Doing so will display information about the web service, which includes a hyperlink called "Service Description." You can also view the WSDL by adding the querystring "?WSDL" to the URL. For example, you can view the WSDL for the eCollege API at:

https://api.ecollege.com/usermanagementServices.asmx?WSDL

What's the URL for the web service?

The eCollege API includes one web service with two methods. Contact https://api.ecollege.com/usermanagementServices.asmx when you're ready to pass data to the web service for processing.

Testing the implementation

You will want to test the web services implementation before you start communicating with live data. Working with your Client Services representative, we've provided a way for you to "try out" the web services calls using test terms, courses, and users with no impact on your live data. This test will help ensure that you're implementing the IMS specification correctly in your XML data and that your SIS and the eCollege API are communicating appropriately.

Setting up a test

Before you begin, you'll need to work with your Client Services representative in gathering the information from the Setup Checklist. This should be completed one week prior to the test data being sent.

Your Client Services representative will send you the eCollege Software Developer's Kit (SDK). This online reference guide will give your developers everything they need to set up and invoke the web services, including sample code, exceptions, and common workflow scenarios.

You'll also need to make sure you have WS Security set up. Your Client Services representative can put you in contact with the eCollege IT department should you need assistance.

Additionally, your Client Services representative will set up a "test" term with test courses and users in preparation for the test. The test term, courses, and users that you'll be working with are listed below.

Once your Client Services representative gives you the go-ahead, you can test the web service call and send over .XML data via web services.

Term Name:
SIS API Testing Term

Course Code	Corresponding Course Titles:
SIS API 1	SIS API Testing Course 1
SIS API 2	SIS API Testing Course 2
SIS API 3	SIS API Testing Course 3
SIS API 4	SIS API Testing Course 4
SIS API 5	SIS API Testing Course 5
SIS API 6	SIS API Testing Course 6
SIS API 7	SIS API Testing Course 7
SIS API 8	SIS API Testing Course 8
SIS API 9	SIS API Testing Course 9
SIS API 10	SIS API Testing Course 10

Call Numbers for SIS API Testing Courses 1-10:
SISAPITEST1
SISAPITEST2
SISAPITEST3
SISAPITEST4
SISAPITEST5
SISAPITEST6
SISAPITEST7
SISAPITEST8
SISAPITEST9
SISAPITEST10

Test Users:

First Name	Middle Name	Last Name	Login ID	Password
Peter	Mark	Aberlard	paberlard	paberlard
Judy	M	Butler	jbutler	jbutler
John	David	Dewey	jdewey	jdewey
David	K	Hume	dhume	dhume
William	Henry	James	wjames	wjames
John	Paul	Locke	jlocke	jlocke
John	Stewart	Mill	jmill	jmill
Ayn	Karen	Rand	arand	arand
Donna	Elizabeth	Haraway	dharaway	dharaway
Thomas	D	Hobbes	thobbes	thobbes

Error messages

Your SIS may receive .Net Framework-generated errors for any of the following reasons:

Error Condition	Description
the web service is unavailable	The underlying connection was closed: Unable to connect to the remote server
the required WSS Username token was not supplied	Failed to call the Webservices: Value cannot be null. o Parameter name: Username -- OR -- o Parameter name: Password
an invalid Username token was used	Microsoft.Web.Services2.Security.SecurityFault: The security token could not be authenticated or authorized.
the password was not the required PasswordDigest type	Failed to call the Webservices: The security token could not be authenticated or authorized

Error Condition	Description
the web service call uses invalid web method parameters, including a badly formed XML request	Failed to call the Webservices: The security token could not be authenticated or authorized
the XML violates the IMS schema error in schema validation	Failed to call the Webservices: Failed to process batch user IMS request XML elements are out of order The following is an example of an error message to a file with XML elements out of order. To validate against the IMS schema, elements must follow the order as specified in the schema: person elements, group elements, and then membership elements. As the following exaple shows, the enterprise (top level element) has a child that is invalid. The child is a group and it is expected to find membership. A group element was incorrectly placed after a membership element. **Example** `</SourcedID>` `- <Result type="Error in Schema Validation">` `<Code>-2147220720</Code>` `<Message>The element 'http://schemas.ecollege.com/Common/2006/01/ims_epv1p1.xsd:enterprise' has invalid child element 'http://schemas.ecollege.com/Common/2006/01/ims_epv1p1.xsd:group'. Expected 'http://schemas.ecollege.com/Common/2006/01/ims_epv1p1.xsd:membership'. An error occurred at , (133, 4).</Message>`  `</Result>`  `<Extension />`

For Username and Password issues, make sure you're using the security credentials exactly as they were provided to you by your Client Services representative.

Finally, make sure your XML data includes all required elements and that the XML conforms to the eCollege interpretation of the IMS schema (refer to the *eCollege Annotated Guide to the IMS Specification*, available from the SDK or from your Client Services representative).

Checking the request status

Once you've submitted your request, you can check the status of the request, as well as view errors and validation results. You can use the reporting option in the eCollege Program Admin Pages to view the status of a specific request, or you can view the status of all requests within a certain timeframe.

To check the status of your request from the Program Admin Pages, as well as view errors:

1. Log in to the Program Administration Pages.

2. Click the **Automated User Management** link in the left-side navigation. In the screen that displays, you can upload requests or view the status of previously submitted jobs.

3. Select to view a single, specific user management request, or to view the status of all requests processed on a certain date; then click **Next**.

Follow the instructions on-screen to complete the process, or refer to the online Help for more detailed information. Requests submitted via web services are designated by "**ws**" in the **Submitter** column.

You can view the status of a request for up to 60 days. After that, the detailed record is no longer available in the Program Admin Pages. However, you can download or export a report to your local machine for your own records during this 60-day period.

Appendix A: Sample autogenerated email

eCollege can designate an auto-email to be sent to users as they are created or enrolled in their online courses. If you'd like to automatically send email notifications, please send an email template with the information you would like sent to your users to your Client Services representative. Feel free to use our standard auto-email as a template for your own:

Hello [Student First, Student Last],

This message is a notification of enrollment as a student in a course hosted on the [Your institution name] eLearning Platform. Login at http://www.yourinstitution.net.

Use the following credentials when asked for your username and password:

Username: [user name]

You have been enrolled in the following course(s):
QA111 | Test Course 111
QA112 | Test Course 112

- If you are a new student, your password is a combination of the year/month you were born (yyyymm)(i.e., If you were born in June of 1975, your password would be 197506).
- If you are a continuing student, your password was not changed, and will still be the last password you used when accessing your courses on www.yourinstitution.net.

When you log in, you will see your courses listed on your Personal Home Page. If you are new to the [Your institution name] eLearning Platform, you should view the Student Tutorial listed on your Home Page. Your instructor or facilitator will provide you with further information about participating in your eLearning courses.

If you have difficulties logging in, please contact the Help Desk at helpdesk@yourdomainname.net. If you do not know why you received this e-mail, please contact your campus Academics Department or the staff at your Center.

[Your institution name] eLearning Support Team

Appendix B: XML Overview

Like HTML, XML is a markup language based on the concept of applying tags to content and then interpreting and displaying the information "described" by the tags. Unlike HTML, though, XML allows you to define your own markup language with its own set of tags rather than using the pre-defined tags associated with HTML. XML is about making information self-describing using meta-data.

Having a basic understanding of the building blocks of an HTML document will help you to identify the elements you will use to create XML for student enrollment and registration exchange with eCollege.

The following sections will discuss some XML basics, such as elements, the root element, nesting elements, attributes, and schemas. The eCollege SIS for API uses the IMS Enterprise v1.1 schema standard.

Elements

An XML element is the most basic component of your document. Elements can contain elements within elements, attributes, and values. Like HTML tags, XML uses open and close tags.

Basic rules to remember about elements:

- You can only have one root element in an XML document (in this case `<enterprise>`).
- Every element must be closed. This can take the form of a pair of tags, one open and one closed `<enterprise>…</enterprise>`, or a single self-closing tag `<enterprise/>`.
- Element names are case sensitive. `<enterprise/>` is not the same element as `<Enterprise/>`.
- All element names are all lowercase (an IMS schema rule). This also applies to all eCollege extensions.
- Element names cannot contain spaces. For example, `phone number` becomes `<phonenumber>`.

Root Element

The first element defined in an XML document is the root element. All of the content within an XML document is contained within the root element. The purpose of the root element is to specify what type of XML file you are creating. For example, if we create an XML document for a cooking recipe, we might define our root element as <recipe>. XML documents begin and end with a root element. You can only have one root element per document.

In an HTML file, the root element would be <html>. But because the elements in HTML are predefined, the root element will always be <html>; however, in XML you can name the root element anything. Remember, the XML document you will create for eCollege must follow the IMS Enterprise v1.1 schema. The IMS schema defines the root element as <enterprise>. (See the section on schemas in this appendix.)

Nesting elements

Elements within another element are considered "nested." When an element is nested within another element, it is encapsulated by the root (outer) element. Nesting elements is a way of keeping order in an XML document. XML elements must be closed in the

order that they are opened. The outer element is referred to as the parent and the inner elements are referred as children, such as:

```
<parent>
   <child>
   </child>
</parent>
```

Attributes

Attributes provide a way to specify additional information about elements. Attributes consist of a name and value (values are always in quotes), and appear within an element. For example, if you had an element like `<phonenumber>, you` might use an attribute `type` to specify that the number is a home, work, fax, or cell phone number.

Basic rules to remember about attributes:

- All attributes values are contained in double quotation marks. Where `<table width=640>…</table>` is acceptable in HTML, in XML it must be written as `<table width=”640”>…</table>`
- All attributes must have a value. For example, in HTML `<checkbox checked/>` is valid; in XML it needs to be `<checkbox checked=”true”/>`
- Like element names, attribute names cannot contain spaces.

Schemas

When the architecture of an XML document is established, it is saved as its own document known as a schema. XML schemas provide a means for defining the structure, content and semantics of XML documents. In the XML documents you will be creating for eCollege, you will be using the IMS Enterprise v1.1 schema.

Remember, a schema defines the structure, not the format, of the XML document. It consists of rules about which elements are allowed.

Valid or Well-Formed XML document?

XML documents that follow the general rules of XML elements are considered well-formed. XML documents that follow the general rules of XML documents and follow the specific rules defined in their schema are valid. An XML document must be well-formed before it can be valid. Only well-formed XML can display in a browser.

Remember, when an XML document is associated with a schema, such as the IMS schema, you must follow the rules defined in that schema. For example, the root element in the IMS schema is defined as <enterprise>. If you were to make the root element in your XML document any other name, the document would not be considered IMS Enterprise v1.1 XML.

IMS Elements Implemented by eCollege

Obviously, creating a schema and defining elements to an XML document can become quite a daunting task. For your convenience, we have already created an XML template for you using the element data structures defined in the rules of the IMS schema.

For a description of each of the element data structures you will use to build your XML document, please refer to the ***Annotated Guide to the IMS Specification***, available from the SDK or from your Client Services representative.

Appendix C: Sample XML data

Following is a sample XML request which conforms to the IMS schema as interpreted by eCollege:

```
<enterprise xmlns="http://schemas.ecollege.com/Common/2006/01/ims_epv1p1.xsd">
   <properties lang="">
     <datasource/>
     <datetime/>
     <extension/>
   </properties>
   <person recstatus="1">
     <sourcedid sourcedidtype="New">
          <source>SchoolOnline</source>
          <id>38641</id>
     </sourcedid>
     <userid pwencryptiontype="" authenticationtype="" password="peskykids"
   useridtype="">scoobydoo</userid>
     <name>
          <fn>Scooby Dooby Doo</fn>
          <n>
               <family>Doo</family>
               <given>Scooby</given>
               <partname partnametype="Middlename" lang="">Dooby</partname>
          </n>
     </name>
     <demographics>
          <gender>Male</gender>
     </demographics>
     <email>user@ecollege.com</email>
     <url>www.mysteryinc.com</url>
     <tel teltype="1">tel_1</tel>
     <adr>
          <street>123 Scooby Snack Avenue</street>
          <locality>Scoobyville</locality>
          <region>CO</region>
          <pcode>88888</pcode>
          <country>USA</country>
     </adr>
   </person>
   <group recstatus="1">
     <comments lang=""/>
     <sourcedid sourcedidtype="New">
          <source>ECLG</source>
          <id>SCHOOL.-DENVER.CEAS.</id>
     </sourcedid>
     <grouptype>
          <typevalue level="">Enrollable Node</typevalue>
     </grouptype>
     <description>
          <short/>
     </description>
   </group>
   <membership>
     <sourcedid sourcedidtype="New">
          <source>ECLG</source>
```

```
            <id>SCHOOL.-DENVER.CEAS.</id>
        </sourcedid>
        <member>
            <sourcedid sourcedidtype="New">
                <source>SchoolOnline</source>
                <id>38641</id>
            </sourcedid>
            <idtype/>
            <role recstatus="1" roletype="01">
                <subrole>2</subrole>
                <status/>
            </role>
        </member>
    </membership>
</enterprise>
```

Appendix D: Common XML errors

XML files must be validated for basic formatting issues and valid schema. In most cases, the creation of XML data will be automated and therefore, generally error-free and well-formed.

However, there are several common XML errors that will cause the request to fail:

- the XML is badly formed (refer to Appendix B and C for an XML overview and sample data); and/or
- the XML violates the schema (refer to the eCollege interpretation of the IMS schema for more information)

The best way to quickly check and see if you have well-formed XML is to open up the XML in a web browser. It will be broken if there is bad XML or if the format is off. Other errors to check for include:

XML-specific errors	
SourcedID not unique; Login ID unique	If the **Update User Properties** option is NOT enabled during setup, this will cause the request to fail.
Unclosed XML tag	All elements, or tags, must be closed, either in a single tag<name/> when the element contains no other elements or text, or with an opening and closing element tag <name>User</name>.
Incorrect UsernameToken	Your Client Services representative will provide you with a Username and Password (token). This token must be passed within your XML data structure.
Incorrect Password	Your Client Services representative will provide you with a Username and Password (token). This token must be passed within your XML data structure.
Unsupported character encoding	

Some other general things to consider are:

- An XML document must have one element that contains all of the others. This is called the Root.
- XML is case sensitive. <name/> is not the same as <Name/>, and neither of these are <NAME/>.
- Elements must be closed, either in a single tag<name/> when the element contains no other elements or text, or with an opening and closing element tag <name>Dean James</name>.
- Similarly, every element must be closed in the order that the elements were opened, so <address><street>…</street></address> is valid, but <address><street>…</address></street> is not.
- All attributes must have a value: <checkbox checked="true"/> not <checkbox checked/>
- All attribute values must be enclosed in double quotes <address type="home"/> not <address type=home/>
- Like element names, attribute names cannot contain spaces.

Note, also, that eliminating as much redundancy in your files will help decrease, sometimes significantly, the time it takes to process the file.