# WebCT® Campus Edition™ 4.0

# Technical Reference Guide

Technical Communications

April 30, 2003

ABOUT THIS GUIDE ..... 8
Section 1: Campus Edition Focus License ..... 8
Section 2: Campus Edition Institution License ..... 8
INTERNATIONAL SUPPORT ..... 10
*Overview* ..... *10*
Some background ..... 10
*How to display multiple languages in the same course in the same WebCT installation* ..... *10*
Creating courses to display multiple languages ..... 10
Converting courses from one character set to another ..... 11
Importing data into WebCT ..... 11
Using the Standard API ..... 12
Using the IMS API ..... 12
Exporting data from WebCT ..... 13
SECTION 1: CAMPUS EDITION FOCUS LICENSE ..... 14
CHAPTER 1 USER AUTHENTICATION ..... 15
*Choosing An Authentication Method* ..... *15*
Browser-Based Authentication Process ..... 15
Ticket Based Authentication Process ..... 16
How WebCT Generates Tickets ..... 16
*Implementing Ticket Based Authentication* ..... *16*
CHAPTER 2 STANDARD API ..... 18
*Overview of the Standard Application Programming Interface* ..... *18*
*Choosing the Appropriate Interface for your Requirements* ..... *18*
*Functionality in the Standard API* ..... *18*
*Implementing the Standard API* ..... *19*
Command Line Interface (webctdb) ..... 19
Syntax ..... 19
Functions ..... 23
Adding users ..... 23
Adding a single user to the global database ..... 23
Adding a single user to the student database ..... 24
Adding multiple users to the global database ..... 24
Adding multiple users to the student database ..... 26
Updating users ..... 26
Updating a single user in the global database ..... 27
Updating a single user in the student database ..... 27
Updating multiple users in the global database ..... 28
Updating multiple users in the student database ..... 29
Deleting users ..... 30
Deleting a single user from the global database ..... 30
Deleting a single user from the student database ..... 30
Deleting multiple users from the global database ..... 31
Deleting multiple users from the student database ..... 31
Finding users ..... 32
Finding a user in the global database ..... 32
Finding a user in the student database ..... 33
Changing WebCT IDs ..... 33

Changing a single user's WebCT ID ..... 33
Changing multiple users' WebCT IDs ..... 34
Web-based Interface (serve_webctdb) ..... 34
1. Setting the API Shared Secret Value ..... 35
2. Developing a Program to Generate an HTTP Request ..... 36
Creating Message Authentication Codes ..... 36
Option 1: Using the get_authentication C function ..... 36
Option 2: Using the Message Authentication Code Generator (get_md5 executable) ..... 37
Option 3: Create a MAC using a language of your own choice ..... 38
How the serve_webctdb MAC is Generated ..... 38
Assembling the HTTP request ..... 38
Syntax ..... 39
Functions ..... 40
Adding users ..... 40
Adding a user to the global database ..... 40
Adding a user to the student database ..... 41
Updating users ..... 42
Updating a user in the global database ..... 42
Updating a user in the student database ..... 43
Deleting users ..... 45
Deleting a user from the global database ..... 45
Deleting a user from the student database ..... 45
Finding users ..... 46
Finding a user in the global database ..... 46
Finding a user in the student database ..... 47
Changing WebCT IDs ..... 48
Changing a user's WebCT ID ..... 48
SECTION 2: CAMPUS EDITION INSTITUTION LICENSE ..... 49
CHAPTER 1 USER AUTHENTICATION ..... 50
*Choosing An Authentication Method* ..... *50*
Browser Based Authentication Process ..... 50
Ticket Based Authentication Process ..... 51
How WebCT Generates Tickets ..... 51
*Implementing Ticket Based Authentication* ..... *51*
Choosing an Authentication Source ..... 52
Using one authentication source ..... 52
Using multiple authentication sources ..... 53
Specifying the LDAP settings ..... 55
Specifying the Kerberos settings ..... 55
Specifying the Windows 2000 Domain Controller settings ..... 55
*Implementing Custom Authentication* ..... *56*
UNIX/Linux ..... 56
WINDOWS 2000 ..... 57
CHAPTER 2 AUTOMATIC SIGNON FROM OTHER SYSTEMS ..... 58
*Automatic Signon Process* ..... *58*
*Implementing Automatic Signon* ..... *59*
Setting Shared Secret Values and Enabling Ticket Based Authentication ..... 59
Developing a Program to Automatically Authenticate a User ..... 60
Creating Message Authentication Codes ..... 60
Option 1: Using the get_authentication C function ..... 60
Option 2: Using the Message Authentication Code Generator (get_md5 executable) ..... 61

Option 3: Create a MAC using a language of your choice ..... 61
Finding the IMS ID for Automatic Signon ..... 62
Finding the IMS ID using the Web-based IMS API ..... 62
Finding the IMS ID using the command line IMS API ..... 62
Finding WUUIs ..... 63
Finding the WUUI using the Web-based Standard API ..... 63
Find a user's WUUI from an IMS ID ..... 64
Finding the WUUI using the command line Standard API ..... 64
Making a Request to the Automatic Signon CGI ..... 65
How the Automatic Signon MAC is Generated ..... 65
CHAPTER 3 OVERVIEW OF THE APPLICATION PROGRAMMING INTERFACES ..... 67
*Differences Between the IMS Enterprise API and the Standard API* ..... *67*
Functional Differences ..... 67
Operational Differences ..... 68
*Choosing the Appropriate Interface for your Requirements* ..... *68*
CHAPTER 4 IMS ENTERPRISE API ..... 70
*IMS API Adaptors* ..... *70*
SIS Grade Export ..... 71
*Functionality in the IMS Enterprise API* ..... *71*
*Terminology* ..... *71*
*Implementing the IMS API* ..... *72*
Command line interface (ep_api.pl) ..... 73
Syntax ..... 73
Functions ..... 74
Import ..... 74
IMS Import Logging ..... 75
Export ..... 78
IMS Export Logging ..... 81
Configure ..... 82
Ims Configuration Logging ..... 83
Web-based interface (serve_ep_api.pl) ..... 84
Setting the API Shared Secret Value ..... 84
Developing a Program to Generate an HTTP Request ..... 85
Creating Message Authentication Codes ..... 85
Option 1: Using the get_authentication C Function ..... 85
Option 2: Using Message Authentication Code Generator (get_md5 executable) ..... 86
Option 3: Create a MAC using a language of your own choice ..... 87
How the serve_ep_api.pl MAC is Generated ..... 87
Generating a checksum ..... 88
Assembling the HTTP request ..... 88
XML File Format Guidelines ..... 90
IMS objects and WebCT relationships ..... 91
Properties object: System identifier ..... 91
Group object: Course ..... 91
Orgunit Object: Categories ..... 92
Relationship Object: Cross-listed Courses ..... 93
Person object: User ..... 98
Membership Object: User Type ..... 99
Specific Group Object: Terms ..... 102
Complete specifications ..... 105
Other XML considerations ..... 105

Syntax ..... 105
Functions ..... 106
Import ..... 106
Export ..... 108
Configure ..... 111
CHAPTER 5 STANDARD API ..... 113
*Functionality in the Standard API* ..... *113*
*Implementing the Standard API* ..... *114*
Command Line Interface (webctdb) ..... 114
Syntax ..... 114
Functions ..... 117
Adding users ..... 117
Adding a single user to the global database ..... 117
Adding a single user to the student database ..... 118
Adding multiple users to the global database ..... 118
Adding multiple users to the student database ..... 119
Updating users ..... 120
Updating a single user in the global database ..... 120
Updating a single user in the student database ..... 120
Updating multiple users in the global database ..... 121
Updating multiple users in the student database ..... 123
Deleting users ..... 124
Deleting a single user from the global database ..... 124
Deleting a single user from the student database ..... 124
Deleting multiple users from the global database ..... 124
Deleting multiple users from the student database ..... 125
Finding WUUIs ..... 125
Finding WUUIs using IMS IDs ..... 126
Finding users ..... 126
Finding a user in the global database ..... 126
Finding a user in the student database ..... 127
Changing WebCT IDs ..... 127
Changing a single user's WebCT ID ..... 127
Changing multiple users' WebCT IDs ..... 127
Exporting myWebCT in XML format ..... 128
Web-based Interface (serve_webctdb) ..... 129
Setting the API Shared Secret Value ..... 129
Developing a Program to Generate an HTTP Request ..... 130
Creating Message Authentication Codes ..... 130
Option 1: Using the get_authentication C Function ..... 130
Option 2: Using the Message Authentication Code Generator (get_md5 executable) ..... 131
Option 3: Create a MAC using a language of your own choice ..... 132
How the serve_webctdb MAC is Generated ..... 132
Assembling the HTTP request ..... 132
Syntax ..... 132
Functions ..... 134
Adding users ..... 134
Adding a user to the global database ..... 134
Adding a user to the student database ..... 135
Updating users ..... 136
Updating a user in the global database ..... 136

Updating a user in the student database ..... 138
Deleting users ..... 139
Deleting a user from the global database ..... 139
Deleting a user from the student database ..... 139
Finding WUUIs ..... 140
Examples ..... 140
Finding a user's WUUI from a WebCT ID ..... 140
Finding a user's WUUI from an IMS ID ..... 141
Finding users ..... 141
Finding a user in the global database ..... 141
Finding a user in the student database ..... 142
Changing WebCT IDs ..... 143
Changing a user's WebCT ID ..... 143
Exporting myWebCT in XML format ..... 143
RESOURCES ..... 145
*LDAP Resources* ..... *145*
Web Sites ..... 145
*Kerberos Resources* ..... *145*
Web Sites ..... 145
*IMS Resources* ..... *145*
Web Sites ..... 145
APPENDIX ..... 146
*Supported Character Sets* ..... *146*

# ABOUT THIS GUIDE

The *Technical Reference Guide: WebCT Campus Edition™ 4.0* is a how-to manual for carrying out advanced administration, integration and reporting tasks not available through the administrator interface. It is written for system administrators and Web developers who are deploying WebCT Campus Edition™ 4.0 for Focus License holders and WebCT Campus Edition™ 4.0 for Institution License holders.

This guide is separated into two main sections, one for each of the two license types available. Prior to the two sections is a chapter describing WebCT's International Support, which describes how administrators can enable courses to display multiple languages through WebCT system administrator interface settings, and through Standard and IMS API commands.

## SECTION 1: CAMPUS EDITION FOCUS LICENSE

Campus Edition™ Focus Use License holders have access to the Standard API, which provides advanced WebCT administrative and reporting functions. Examples in this guide focus primarily on syntax and assume a strong Web programming background. For more detailed examples, download the Practical Examples document from the WebCT Documentation, API Guides section at `http://download.webct.com`.

An overview of Section 1 follows:

Chapter 1: User Authentication

- Describes methods for providing secure access to WebCT.

Chapter 2: Standard API

- Describes how to add, update, and delete one or multiple users.
- Describes how to find users in the global or student database.
- Describes how to change WebCT IDs.

## SECTION 2: CAMPUS EDITION INSTITUTION LICENSE

Campus Edition™ users have access to the IMS API and the Standard API. The IMS API enables integration with existing campus systems, such as student information systems and portals. The Standard API allows access to advanced WebCT administrative and reporting functions.

Examples in this guide focus primarily on syntax and assume a strong Web programming background. For more detailed examples, download the Practical Examples document from the WebCT Documentation, API Guides section at `http://download.webct.com`.

An overview of Section 2 follows:

Chapter 1: User Authentication

- Describes methods for providing secure access to WebCT, using one or more authentication sources.

Chapter 2: Automatic Signon from Other Systems

- Describes how to implement automatic signon.

Chapter 3: Overview of the Application Programming Interfaces

- Describes the functional and operational differences between the IMS Enterprise API and the Standard API.

Chapter 4: IMS Enterprise API

- Describes how to import data into the WebCT database, export data from the WebCT database, and how to set the IMS ID for group and person objects.
- Describes log files created during IMS import, export and configure events.

Chapter 5: Standard API

- Describes how to add, update, and delete one or multiple users.
- Describes how to find WUUIs and how to find users in the global or student database.
- Describes how to Change WebCT IDs.
- Describes how to export myWebCT in XML format.

# INTERNATIONAL SUPPORT

## OVERVIEW

WebCT Campus Edition™ supports both ISO 8859-1 and UTF-8 character sets. Functionality in the Standard API, the IMS API, and the system administrator interface allow multiple languages to be displayed in the same course in the same WebCT installation. This is required, for example, if an instructor teaching a Japanese course to English speaking students wants English and Japanese characters to be displayed in the same course. Note that while a course can display multiple languages, the actual course content will continue to be in one language. In the course described above, the language of the course would be English, but an instructor can upload and display Japanese files in the course.

### SOME BACKGROUND

Textual data in a file is input and displayed in a character set, which is a specific collection of characters representing an alphabet. The ISO-8859-1 character set, for example, can display a file written in English, French, Spanish and most other Western European languages. In previous versions of WebCT, the ISO-8859-1 character set was used exclusively. If a file is encoded in a character set other than ISO-8859-1, such as Japanese, the characters in this file would be displayed incorrectly in a WebCT course.

WebCT now allows files to be converted to the UTF-8 character set, which supports over 650 of the world's languages including Japanese, Chinese, Russian, French, and German. Once converted, files are saved in WebCT's database in the UTF-8 character set.

**Important:** Courses converted from UTF-8 to ISO-8859-1 may lose some characters in the conversion.

## HOW TO DISPLAY MULTIPLE LANGUAGES IN THE SAME COURSE IN THE SAME WEBCT INSTALLATION

Displaying course material in multiple languages requires some work by a system administrator. This work revolves around the requirement that WebCT know the character set of the data it is dealing with to properly support the mixing of character sets. Administrators will need to perform tasks related to internationalization when doing the following:

- Creating courses to display multiple languages
- Converting courses from one character set to another
- Importing data into WebCT
- Exporting data from WebCT

Details about each task are provided below.

### CREATING COURSES TO DISPLAY MULTIPLE LANGUAGES

WebCT ships with two default English language plug-ins: English ISO-8859-1 and English UTF-8. In addition, you can install other language plug-ins, each of which will have a UTF-8 version. A setting in the WebCT administrator interface allows you to choose one of the installed languages as the default language to be used when creating new courses. If your instructors have a common need to mix characters from different languages within courses then you should select a UTF-8 language as the default setting for new courses, and selectively

create ISO-8859-1 courses only as required. However, if a majority of your instructors use ISO-8859-1 files for their course content then you should select an ISO-8859-1 language as the default setting for new courses, and selectively create UTF-8 courses only as required.

**Note:**

- If new courses are created in the UTF-8 character set, this enables designers of mixed language courses to upload files as UTF-8.
- If new courses are not created in the UTF-8 character set, designers do not have access to the functionality allowing them to upload files as UTF-8.

For instructions on setting the default language for new courses, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 5, User Management, "Setting the Default Language for New Courses."*

## CONVERTING COURSES FROM ONE CHARACTER SET TO ANOTHER

Courses created in previous versions of WebCT will commonly be in the ISO-8859-1 character set. If required, these courses can be converted to the UTF-8 character set. Courses can also be converted from UTF-8 to ISO-8859-1. You can perform the conversion using a command line utility. For instructions, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 3, Course Management, "Converting the Course Character Set."*

**Note:** Converting courses using this utility will convert all content in a course except *Mail* and *Discussions* attachments.

## IMPORTING DATA INTO WEBCT

The character set for the administrator interface is always UTF-8. If you use the administrator interface or the API to import data that is encoded in a character set other than UTF-8, WebCT converts the data to UTF-8. In the administrator interface, there is a setting that allows you to select the character set of your region so that whenever you import or export data, the data can be converted. The default is set to convert from ISO-8859-1.

You can override this default setting when importing using the command line interface to the Standard API and the IMS API. For example, if you are importing a UTF-8 file and your administrator interface character set is ISO-8859-1, you can override this setting.

We recommend you configure any background scripts that use the Standard or IMS API to specify an override for the default character set. Further details and instructions to perform each of these tasks are provided in the next section.

For instructions on selecting the character set of your region, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 5, User Management, "Set the Character Set for Imported and Exported Administrator File."*

**Note:** This setting does not change the language of the administrator interface. To change the language, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 5, User Management, "Set the Language for the WebCT Administrator Interface."*

## USING THE STANDARD API

### *To override the character set for imported and exported administrator files when using the command line interface to the Standard API*

When you perform an operation using the Standard API, you can override the character set for imported and exported administrator files, which is set in the administrator interface. For example, to import a file created in UTF-8 through the command line interface when your character set for administrator is ISO-8859-1, you will need to override the file character set by specifying a character set of UTF-8 at the command line. See Section 1: Campus Edition Focus License*, Chapter 2 Standard API, Implementing the Standard API,Command Line Interface (webctdb)*, page 19 in this guide for details about how to set the CHARSET parameter.

### *To override the character set for imported and exported administrator files when using the Web interface to the Standard API*

Some system administrators have created scripts that automatically import data files from outside systems to WebCT on a regular basis. By default, these scripts rely on the character set for imported and exported administrator files, which is set in the administrator interface. You should change all automatic scripts to explicitly specify the character set of the files being imported to match the actual character set of the source data. This will ensure that background scripts import data smoothly, no matter the changes made to the character set for imported and exported administrator files. See *Section 1: Campus Edition Focus License, Chapter 2 Standard API, Implementing the Standard API,Command Line Interface (webctdb)*, page 19 in this guide for details about how to set the CHARSET parameter.

## USING THE IMS API

### *To override the character set for imported and exported administrator files when using the command line interface to the IMS API*

XML header files contain a tag specifying the character set of the file. This tag will override the character set for imported and exported administrator files, which is set in the administrator interface. However, as some sources of IMS XML data will generate files in which the tag indicating the character set in the header doesn't match the actual character set used in the contents of the file, it is recommended that you always specify the file character set parameter at the command line to override the XML header tag. This will ensure data is imported correctly. See *Section 2: Campus Edition Institution License, Chapter 4 IMS Enterprise API,* Import, Example 2 page 75 in this guide for details about how to set the CHARSET parameter.

### *To override the character set for imported and exported administrator files when using the Web interface to the IMS API*

Some system administrators have created scripts that automatically import data files from outside systems to WebCT on a regular basis. By default, these scripts rely on the character set for imported and exported administrator files, which is set in the administrator interface. You should change all automatic scripts to explicitly specify the character set of the files being imported to match the actual character set of the source data. This will ensure that background scripts import data smoothly, no matter the changes made to the character set for imported and exported administrator files. See *Section 2: Campus Edition Institution License, Chapter 4 IMS Enterprise API,* Import, Example 2 page 75 in this guide for details about how to set the CHARSET parameter.

## EXPORTING DATA FROM WEBCT

To ensure that data exported from WebCT can be displayed correctly in the target system, the character set for imported and exported administrator files, which is set in the WebCT administrator interface, must match the character set of the local machine used by the target system for viewing exported files. You should specify the setting to match the machine to which you most often export, which may be your own. Exported files are converted to the character set specified in the setting.

If the target system requires a different character set than that indicated in the setting, you can override the setting. You can do so using the command line interface of the Standard API and the IMS API. See the relevant sections of the *Importing data into WebCT* in this guide for details.

We recommend you configure any background scripts that use the Standard or the IMS API to always specify an override for the default character set. See the relevant sections of the *Importing data into WebCT* in this guide for details.

# SECTION 1: CAMPUS EDITION FOCUS LICENSE

# CHAPTER 1 USER AUTHENTICATION

WebCT Campus Edition™ 4.0 provides two major methods for user authentication:

| | |
|---|---|
| Browser Based Authentication | • Users are authenticated through a browser dialog box that prompts for a username and password. The username and password are verified against WebCT's internal database. If the user is authorized, a Basic Authentication Header is provided. Subsequent page accesses to WebCT are authorized according to the browser header.<br>• This authentication method is used in previous versions of WebCT. |
| Ticket Based Authentication | • Users are authenticated through a login page that prompts for a username and password. The username and password are verified against WebCT's internal database. If the authentication is successful, the user is issued a browser cookie that serves as a ticket. Subsequent page accesses to WebCT are authorized according to the ticket. |

## CHOOSING AN AUTHENTICATION METHOD

Certain features of WebCT Campus Edition™ 4.0 require ticket-based authentication, including:

- Logout
- Server lockdown
- Session timeout
- Customizable login page

Browser-based authentication is primarily provided in WebCT Campus Edition™ 4.0 as a legacy option. Choose this method of authentication if:

- Your institution has an information technology policy forbidding the use of applications that employ browser cookies.
- It is critical that the authentication method remains the same as used in your previous version of WebCT, and if the previously used method was browser-based authentication.

### BROWSER-BASED AUTHENTICATION PROCESS

Browser-based authentication has served as the standard authentication method for all previous versions of WebCT. When a user a requests a URL, authentication of the user occurs as follows:

1. The Web server checks to see if the requested URL requires authorization.
2. If none is required (e.g., a user requests a course Welcome Page), then the Web server delivers the page to the browser.
3. If authorization is required, the Web server checks to see if the user has already provided a username and password by checking to see if a valid Basic Authentication Header was provided in the request. If the header is valid, the page is delivered.

4. If the Basic Authentication Header is invalid, or no header is provided, the user is prompted with a username and password dialog box. The cycle is then repeated.

### TICKET BASED AUTHENTICATION PROCESS

When a user requests a URL, authentication of the user occurs as follows:

1. The Web server checks to see if the requested URL requires authorization.
2. If none is required, the Web server delivers the page to the browser.
3. If authorization is required, WebCT checks for a valid ticket.
4. If a valid ticket is found (i.e. the user has been authenticated and is authorized for the resource), the page is delivered to the browser.
5. If a ticket is not found, a login form is delivered to the browser. The user submits the form and WebCT authenticates the user, issuing their browser a cookie. The URL is re-requested and the cycle repeats.

#### HOW WEBCT GENERATES TICKETS

WebCT tickets (in the form of browser cookies) contain the following information:

- Username
- Encrypted Password (DES encryption)
- Timestamp (UNIX Epoch format)
- Message Authentication Code (MAC)

The MAC is generated in three steps:

1. The username, encrypted password, timestamp, user agent information (if sent), and a shared secret value are concatenated.
2. The concatenated string is encrypted with the MD5 algorithm.
3. The encrypted string is encrypted a second time with the MD5 algorithm.

## IMPLEMENTING TICKET BASED AUTHENTICATION

With ticket-based authentication, you authenticate using WebCT's internal database.

1. From the *Admin* toolbar, click **Server Mgmt**. The *Server Mgmt* toolbar appears.
2. From the *Server Mgmt* toolbar, click **Settings**. The *Administrator Settings* screen appears.
3. Under *User Authentication*, select *Use ticket based authentication*.
4. Choose whether the **Logout** link should appear in the course *Menu Bar*:
    - To display the **Logout** link, select *Display Logout link in course Menu Bar*.

- To hide the **Logout** link, deselect *Display Logout link in course Menu Bar*. **Note:** If you run WebCT in a framed environment (such as a portal) where a logout link or "Return to Portal" link already exists, you can hide the **Logout** link.

5. In the *Ticket shared secret value* text box, either leave the shared secret value that was automatically generated by WebCT or enter a new shared secret value. For security reasons, the value *secret* does not work. The secret value
   - is case-sensitive
   - cannot exceed 256 characters
   - cannot contain tab or other control characters
   - should not contain end-of-line characters. **Note:** By default, the UNIX text editors vi and pico automatically add end-of-line characters. Check the file size to ensure that the number of characters equals the number of bytes.

6. In the *Tickets remain valid for* text box, enter the number of minutes until ticket time-out. This value controls the expiry time of the ticket based on the user's last access and therefore affects how long a user can stay logged in while inactive. The default is 180 minutes.
7. Choose whether to allow WebCT authentication across a domain. Authentication across a domain allows users to access all servers in the domain, without having to re-authenticate for each one.
   - To allow authentication across a domain:
     a) Select *Allow WebCT authentication across a domain*.
     b) In the *Please specify your domain* text box, enter the domain name. The domain name must have a period in front of it. Example: `.webct.com`
   - To disallow authentication across a domain, select *Do not allow WebCT authentication across a domain*.

8. Under *User is authenticated using*, from the drop-down list for the authentication source that you are using, select *First*. **Note:** With a Focus Use License, only the WebCT internal database can be used as the authentication source.
9. For all other authentication sources, select *Never*.
10. Scroll to the bottom of the screen and click **Update**.

# CHAPTER 2 STANDARD API

The Standard API gives administrators and developers access to the WebCT databases via command line or Web-based interfaces, to perform a variety of administration and reporting tasks.

## OVERVIEW OF THE STANDARD APPLICATION PROGRAMMING INTERFACE

Application Programming Interfaces (APIs) allow users and other systems to directly interface with WebCT without the graphical user interface. This chapter describes the proprietary WebCT CE 4.0 Standard API. This API has two interfaces: a command line interface and a Web-based interface.

## CHOOSING THE APPROPRIATE INTERFACE FOR YOUR REQUIREMENTS

WebCT provides two interfaces of the Standard API: a command line interface and a Web-based interface. Choosing an interface is not a one-time decision; it will vary depending on the task that you need to accomplish.

Use the following table as a guide for choosing the best interface for your task.

| Task | Suggested Interface |
|---|---|
| Processing multiple records simultaneously | command line |
| | |
| Processing a single record | command line |
| | |
| Integrating systems that are on the same physical server and run as the same user as WebCT | command line |
| | |
| Debugging | command line |
| | |
| Integrating external system with WebCT<br>(e.g., you want to integrate your institution's SIS with WebCT) | Web-based |

## FUNCTIONALITY IN THE STANDARD API

The Standard API allows you to manipulate two separate databases within WebCT: the global database and the student database.

The global database contains the central listing of all users on the WebCT server. By default, the global database contains the WebCT ID, Password, First Name, Last Name, Courses, and Registered Courses fields. All users must have a record in this database in order to access a course.

The student database is a term for a collection of databases specific to a course. Every WebCT course has its own student database that contains, by default, the User ID, Password, First Name, and Last Name fields. The

information in the student database can be viewed in the designer interface of the *Manage Students* feature in each WebCT course.

Generally, a WebCT ID is linked to a User ID for each course that a user is enrolled in. Users can have different User IDs from their WebCT IDs, as well as different First Name and Last Name data in the student and global databases.

The functionality of the Standard API can be divided into the following basic categories:

| | |
|---|---|
| Adding Users | • Adding a single user to the global database or student database<br>• Adding multiple users to the global database or student database |
| Updating Users | • Updating a single user in the global database or student database<br>• Updating multiple users in the global database or student database<br>• Updating user types |
| Deleting Users | • Deleting a single user from the global database or student database<br>• Deleting multiple users from the global database or student database |
| Finding Users | • Finding a user in the global database or student database |
| Changing WebCT IDs | • Changing a single user's WebCT ID<br>• Changing multiple users' WebCT IDs |

# IMPLEMENTING THE STANDARD API

## COMMAND LINE INTERFACE (WEBCTDB)

The command line interface to the standard API provides a simple interface to the WebCT API. The executable file `webctdb`, is located in the directory `<install_dir>/webct/webct/generic/api/`.

**Important:** WebCT strongly recommends you run the Standard API as the WebCT user. Operating the API as the Root user can prevent students and designers from logging into WebCT.

### SYNTAX

The general syntax for each of the Standard API operations is as follows:

| Operation | Field Names |
|---|---|
| `add` | `<db> <course> <fieldsData_pair_list> <separator> [encrypted] [charset]` |
| | |
| `delete` | `<db> <course> <WebCT_ID \| user_id> [charset]` |
| | |
| `changeid` | `<db> <course> <fieldsData_pair_list> <separator> [charset]` |
| | |

| Operation | Field Names |
|---|---|
| update | <db> <course> <fieldsData_pair_list> <separator> [encrypted] [charset] |
| | |
| find | <db> <course> <WebCT_ID \| user_id> <separator> [user_type] [charset] |
| | |
| fileadd | <db> <course> <filename> <separator> [encrypted] [charset] |
| | |
| fileupdate | <db> <course> <filename> <separator> [encrypted] [charset] |
| | |
| filedelete | <db> <course> <filename> [charset] |
| | |
| filechangeid | <db> <course> <filename> <separator> [charset] |

Details about each field name are provided below:

**Field Name:** db
**Value:** global or student
**Example:** global
**Description:** This is the name of the database, either global database or student database.

**Field Name:** course
**Value:** Course ID
**Example:** cs100
**Description:**
- Required for student database operations.
- For global database operations, enter the placeholder value xxxx.

**Field Name:** fieldsData_pair_list
**Value:** A double quote-enclosed list of field-data pairs in the form: field_name=data_value.
**Example:** "WebCT ID=student1"
**Description:**
- The field names must exist in the WebCT global database or student database. The *separator* must be inserted between each of the field-data pairs.
- For the global database, the optional fields Courses and Registered Courses are available for adding and/or modifying courses and registered courses to which a global user belongs. The values for these fields can be a colon-separated list of course IDs for Courses or course names for Registered Courses. For example, Courses=cs100:psyc100:math100. If you also specify a user type with the course, this is separated from the course ID by a semicolon, for example, Courses=cs100;D:psyc100;TA. **Note**: The default user type is (S)tudent.

- A user can be added as a primary designer or as a secondary designer. The first WebCT ID added to the course as a designer becomes the primary designer; every subsequent designer becomes a secondary designer.

**Field Name:** `fieldsData_pair_list (cont.)`

**Description:** **Note**: The following are reserved words in the fieldsData_pair_list:

- Login ID (this is old terminology and is supported for backward compatibility only. It has the same meaning as User ID).
- User ID (the User ID of a student in a course)
- Password (the password of the global user or the student)
- Global ID (this is old terminology and is supported for backward compatibility only. It has the same meaning as WebCT ID).
- WebCT ID (WebCT ID of a global user)
- First Name (first name of the global user or the student. It is one of the reserved columns in both the global and student databases)
- Last Name (last name of the global user or the student. It is one of the reserved columns in both the global and student databases)
- Courses (the list of WebCT courses for a global user). If you populate this field through the API, the course must already exist on the WebCT server.
- Registered Courses (the list of courses maintained by the registrar for a global user. These courses may or may not be a WebCT course.)
- Thumbprint (internal data and cannot be modified)
- LockPID (internal data and cannot be modified)
- #User Type (internal data. This can be modified through the API)
- #E-mail (internal data and cannot be modified)
- #Password Question (internal data and cannot be modified)
- #Password Answer (internal data and cannot be modified)

**Note**: Reserved words are case sensitive.

**Field Name:** `separator`

**Value:** Any alphanumeric string representing the separator between data pairs in the fieldsData_pair_list.

**Example:** "," (comma )

**Description:** Delimiter used to separate data items. You must declare what value you will be using as a delimiter for the operations add, delete, changeid, update, and find.

**Note**: For the global database, the colon and semi-colon are not allowed as separators.

**Field Name:** `user_type`

**Value:** `user_type`

**Example:** `user_type`

**Description:** Only used with the `find` operation on the global database; return value of user_type is one of three users types: S for student , D for designer, and TA for teaching assistant.

**Field Name:** `encrypted`
**Value:** `encrypted`
**Example:** `encrypted`
**Description:**
- Only used with the add, update, fileadd and fileupdate operations.
- The password must be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- Add to the end of the command line to indicate that the passwords are passing in encrypted form.

**Field Name:** `charset`
**Value:** A valid character set. See the *Appendix*.
**Example:** `"--charset=iso-8859-1"`
**Description:**
- If specified, charset will override the character set as defined on the administrator settings page as the file charset.
- **Note**: The default charset type is UTF-8

## FUNCTIONS

### ADDING USERS

Users can be added to the global database or student databases. However, you should add users to the global database and use the Courses field to add them to each course. This method is simpler and automatically links the WebCT ID to each User ID.

**Important:** If you are adding a user to a cross-listed course, see *Adding Users to Cross-Listed Courses*, page 95 for rules governing user roles.

**Warning:** WebCT strongly recommends you re-add previously deleted users through the administrator interface. Re-adding users through the Standard API may prevent preserved records from merging with the user. See the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 5, User Management.*

#### ADDING A SINGLE USER TO THE GLOBAL DATABASE

**Operation = `add`**

- The fieldsData_pair_list must include both the WebCT ID and Password fields.
- You can specify the user type: S for student, D for designer, TA for teaching assistant. If you don't specify a user type, the user type defaults to (S)tudent. Except in the case of a cross-listed course, if the user type is specified as (D)esigner and there is no existing designer, the user is added as the primary designer. In a cross-listed course, all designers are secondary, or shared access designers. In the case of a non-cross-listed course, if there is an existing designer, the user is added as secondary designer.

**Warning:** When a designer is assigned to one cross-listed course, they also have designer access to all courses in the set, with access to all student and TA records. Similarly, when a TA is assigned to one cross-listed course, they also have TA access to all courses in the set, with access to all

| Operation = add |
| --- |
| student records. |

### *Syntax*

**add**

```
<db> <course> <fieldsData_pair_list> <separator> [encrypted]
```

### Example

Add a user named `Justin Case` to the global database as a designer for `cs100`; a teaching assistant for `cs200`; and as a student in `cs810`:

Enter the command:

UNIX
```
./webctdb add global xxxx "WebCT ID=jcase,Password=1234,First
Name=Justin,Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S" ","
```

Windows
```
webctdb add global xxxx "WebCT ID=jcase,Password=1234,First
Name=Justin,Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S" ","
```

#### ADDING A SINGLE USER TO THE STUDENT DATABASE

| Operation = add |
| --- |
| • The fieldsData_pair_list must include both the User ID and Password fields. |

### *Syntax*

**add**

```
<db> <course> <fieldsData_pair_list> <separator> [encrypted]
```

### Example

Add a user named `Bailey Wick` to the student database for course `cs100`:

Enter the command:

UNIX
```
./webctdb add student cs100 "User ID=bwick,Password=1234,
First Name=Bailey,Last Name=Wick" ","
```

Windows
```
webctdb add student cs100 "User ID=bwick,Password=1234,
First Name=Bailey,Last Name=Wick" ","
```

#### ADDING MULTIPLE USERS TO THE GLOBAL DATABASE

| Operation = fileadd |
| --- |
| • filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is only the name of the file. A file extension, such as `.txt`, is recommended.<br>• The file must be in plain text. The first line of the file must be the field names separated by the separator value. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the separator value. There must be no spaces between |

**Operation = fileadd**

the data and the separators.

- If the user exists in the database, `fileadd` will send an error message to STDOUT. The user record will not be changed; the process will skip to the next record in the file.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- If you are adding users to a cross-listed course, be sure to review the rules about user roles in cross-listed courses in *Adding Users to Cross-Listed Courses*, page 95.

### *Syntax*

**fileadd**

```
<db> <course> <filename> <separator> [encrypted]
```

### Example

Add users to the global database from a text file named `users.txt`.

**SAMPLE USERS.TXT FILE:**

```
WebCT ID,Password,Last Name,First Name
jsmith,9876,Smith,John
jbrown,2345,Brown,Jane
bfawlty,8765,Fawlty,Basil
arigsby,5432,Rigsby,Arthur
```

Enter the command:

UNIX `./webctdb fileadd global xxxx users.txt ","`

Windows `webctdb fileadd global xxxx users.txt ","`

### ADDING MULTIPLE USERS TO THE STUDENT DATABASE

**Operation = fileadd**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is only the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the separator value. There must be no spaces between the data and the separators.
- If the user exists in the database, `fileadd` will send an error message to STDOUT. The user record will not be changed in the database; the process will skip to the next record in the file.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.

#### *Syntax*

**fileadd**

```
<db> <course> <filename> <separator> [encrypted]
```

#### Example

Add students whose records are stored in the file `class.txt` to the course `cs100`.

**SAMPLE CLASS.TXT FILE**

```
User ID,Password,Last Name,First Name
jsmith,9876,Smith,John
jbrown,2345,Brown,Jane
bfawlty,8765,Fawlty,Basil
arigsby,5432,Rigsby,Arthur
```

Enter the command:

UNIX `./webctdb fileadd student cs100 class.txt ","`

Windows `webctdb fileadd student cs100 class.txt ","`

### UPDATING USERS

**Important:** If you are adding a user to a cross-listed course, see *Adding Users to Cross-Listed Courses*, page 95 for rules governing user roles.

### UPDATING A SINGLE USER IN THE GLOBAL DATABASE

| Operation = update |
| --- |
| • The fieldsData_pair_list must include the WebCT ID.<br>• Empty fields are not changed.<br>• If the field value is "_DELETE_", the value will be set to null<br>• When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you supply and unlinked from any pre-existing courses that you do not supply.<br>• You can update and change a user type by specifying a different user type. For example, you can change a designer (D) into a student (S).<br>• `update` cannot be used to modify quiz scores.<br>• An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or updated users may not be able to access WebCT.<br>• If you are updating a user in a cross-listed course, be sure to review the rules about user roles in cross-listed courses in *Adding Users to Cross-Listed Courses* on page 95 in this guide. |

#### *Syntax*

**update**

```
<db> <course> <fieldsData_pair_list> <separator> [encrypted]
```

#### Example 1

For the student `Justin Case`, password `1234`, with the following courses: `cs100(D)` `cs200(TA)` `cs810(S)`, update the password in the global database and update the courses so that only `cs100` remains.

Enter the command:

UNIX
```
./webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100" ","
```

Windows
```
webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100" ","
```

#### Example 2

Using the previous example, update `Justin Case` so he is now a student in course `cs100`.

Enter the command:

UNIX
```
./webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100;S" ","
```

Windows
```
webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100;S" ","
```

### UPDATING A SINGLE USER IN THE STUDENT DATABASE

| Operation = update |
| --- |
| • The fieldsData_pair_list must include the User ID field. |

**Operation = update**

- Empty fields are not changed.
- If the field value is "_DELETE_", the value will be set to null
- When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you supply and unlinked from any pre-existing courses that you do not supply.
- You can update a user type by specifying a different one.
- `update` cannot be used to modify quiz scores.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the updated users may not be able to access WebCT.

### *Syntax*

**update**

```
<db> <course> <fieldsData_pair_list> <separator> [encrypted]
```

### Example

To update the student `Bailey Wick`, first name, last name, and password of `abcd`.

Enter the command:

UNIX
```
./webctdb update student cs100 "User ID=bwick,Password=abcd,
First Name=Bailie,Last Name=Wicke" ","
```

Windows
```
webctdb update student cs100 "User ID=bwick,Password=abcd,
First Name=Bailie,Last Name=Wicke" ","
```

## UPDATING MULTIPLE USERS IN THE GLOBAL DATABASE

**Operation = fileupdate**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the separator value. There must be no spaces between the data and the separators.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- `fileupdate` will add a user if they do not exist in the database.
- Empty fields will not be changed.
- For `fileupdate`, if the value field value is "_DELETE_", the value will be set to null.
- When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you supply and unlinked from any pre-existing courses that you do not supply.

| Operation = fileupdate |
| --- |
| • If you are updating a user in a cross-listed course, be sure to review the rules about user roles in cross-listed courses in *Adding Users to Cross-Listed Courses* on page 95 in this guide. |

### *Syntax*

**fileupdate**

```
<db> <course> <filename> <separator> [encrypted]
```

### Example

Change the names of a group of users whose updates are contained in the file `updates.txt`.

**SAMPLE UPDATES.TXT FILE:**

```
WebCT ID,Last Name,First Name
jsmith,Smith,Jerry
jbrown,Brown,Janet
bfawlty,Fawlty,Brian
arigsby,Rigsby,Alan
```

Enter the command:

UNIX `./webctdb fileupdate global xxxx updates.txt ","`

Windows `webctdb fileupdate global xxxx updates.txt ","`

#### UPDATING MULTIPLE USERS IN THE STUDENT DATABASE

| Operation = fileupdate |
| --- |
| • filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is the name of the file. A file extension, such as `.txt`, is recommended.<br>• The file must be in plain text. The first line of the file must be the field names separated by the separator. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. There must be no spaces between the data and the separators.<br>• An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.<br>• `fileupdate` will add a student or user if they do not already exist in the database.<br>• Empty fields will not be changed.<br>• For `fileupdate`, if the field value is "_DELETE_", the value will be set to null.<br>• `fileupdate` overwrites the data fields being changed; it does not append. |

### *Syntax*

**fileupdate**

```
<db> <course> <filename> <separator> [encrypted]
```

### Example

Change the names of students in the course `cs100` using updates contained in the file `updates.txt`.

**SAMPLE UPDATES.TXT FILE:**

```
User ID,Last Name,First Name
jsmith,Smith,Jerry
jbrown,Brown,Janet
bfawlty,Fawlty,Brian
arigsby,Rigsby,Alan
```

Enter the command:

UNIX `./webctdb fileupdate student cs100 updates.txt ","`

Windows `webctdb fileupdate student cs100 updates.txt ","`

## DELETING USERS

### DELETING A SINGLE USER FROM THE GLOBAL DATABASE

| Operation = delete |
| --- |
| • global_id is the ID of the user to be deleted from the global database.<br><br>**Note**: Depending on the *User Data* setting in the administrator interface, the student's data can also be deleted from the student database. |

#### *Syntax*

**delete**

```
<db> <course> <WebCT_ID | user_id>
```

### Example

Delete the global database record for the user whose WebCT ID is `jcase`. **Note**: The student will be denied access to all the courses listed in their global database record. Depending on the *User Data* setting in the administrator interface, the student's data can also be deleted from the student database.

Enter the command:

UNIX `./webctdb delete global xxxx jcase ","`

Windows `webctdb delete global xxxx jcase ","`

### DELETING A SINGLE USER FROM THE STUDENT DATABASE

| Operation = delete |
| --- |
| • user_id is the ID of the student to be deleted from the student database. |

### Example
Delete the record for the student in the `cs100` course whose User ID is `bwick`.

Enter the command:

UNIX
```
./webctdb delete student cs100 bwick ","
```

Windows
```
webctdb delete student cs100 bwick ","
```

#### DELETING MULTIPLE USERS FROM THE GLOBAL DATABASE

**Operation = filedelete**

- filename is the either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the file. A file extension, such as `.txt`, is recommended.
- If the user does not exist in the database, `filedelete` will send an error message to STDOUT. The process will skip to the next record in the file.

### *Syntax*

**filedelete**
```
<db> <course> <filename>
```

### Example
Delete users from the global database using a text file `deleteusers.txt`.

**SAMPLE DELETEUSERS.TXT FILE:**
```
jsmith
jbrown
bfawlty
arigsby
```

Enter the command:

UNIX
```
./webctdb filedelete global xxxx deleteusers.txt "," 
```

Windows
```
webctdb filedelete global xxxx deleteusers.txt "," 
```

#### DELETING MULTIPLE USERS FROM THE STUDENT DATABASE

**Operation = filedelete**

**Operation = filedelete**

- filename is the either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the file. A file extension, such as `.txt`, is recommended.
- If the user does not exist in the database, `filedelete` will send an error message to STDOUT The process will skip to the next record in the file.

### *Syntax*

**filedelete**

```
<db> <course> <filename>
```

### Example

Delete students whose records are stored in the file `delete.txt` from the course `cs100`.

**SAMPLE DELETE.TXT FILE:**

```
jsmith
jbrown
bfawlty
arigsby
```

Enter the command:

UNIX `./webctdb filedelete student cs100 delete.txt`

Windows `webctdb filedelete student cs100 delete.txt`

## FINDING USERS

### FINDING A USER IN THE GLOBAL DATABASE

**Operation = find**

- WebCT ID is the WebCT ID of the user in the global database.
- Separator is the separator of the output data, which is sent to STDOUT in the same format as the fieldsData_pair_list.
- If the field name user_type is specified in a global database query, the user type (S,D,TA) will be included in the result.

### *Syntax*

**find**

```
<db> <course> <WebCT_ID | user_id> <separator> [user_type]
```

### Example

Find the global database record, including user type, for the user with the WebCT ID of `jcase`.

Enter the command:

UNIX `./webctdb find global xxxx jcase "," user_type`

Windows `webctdb find global xxxx jcase "," user_type`

If the command is successfully executed:

```
Success: WebCT ID=jcase,First Name=Justin,
      Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S
```

### FINDING A USER IN THE STUDENT DATABASE

| Operation = find |
| --- |
| • user_id is the User ID of the student in the student database.<br>• Separator is the separator of the output data, which is sent to STDOUT in the same format as the fieldsData_pair_list. |

#### *Syntax*

**find**

```
<db> <course> <WebCT_ID | user_id> <separator> [user_type]
```

**Example**

Find the student in the course `cs100` whose User ID is `bwick`.

Enter the command:

UNIX `./webctdb find student cs100 bwick ","`

Windows `webctdb find student cs100 bwick ","`

If the command is successfully executed:

```
Success:First Name=Bailie,Last Name=Wicke,User ID=bwick
```

## CHANGING WEBCT IDS

### CHANGING A SINGLE USER'S WEBCT ID

| Operation = changeid |
| --- |
| • `changeid` can only be used on the global database.<br>• old_id is the WebCT ID to be changed.<br>• new_id is the new WebCT ID. |

#### *Syntax*

**changeid**

```
<db> <course> <fieldsData_pair_list> <separator>
```

**Example**
Change Justin Case's WebCT ID from `jcase` to `jicase`.

Enter the command:

UNIX `./webctdb changeid global xxxx "Old ID=jcase,New ID=jicase" ","`

Windows `webctdb changeid global xxxx "Old ID=jcase,New ID=jicase" ","`

### CHANGING MULTIPLE USERS' WEBCT IDS

| Operation = filechangeid |
|---|
| • filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the file. A file extension, such as `.txt`, is recommended.<br>• The first line of the data file should be the field names Old ID and New ID, separated by the separator_file string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the separator value. **Note**: The field name Old ID does not exist in the databases.<br>• If the user does not already exist in the database, `filechangeid` will send an error message to STDOUT. The process will skip to the next record in the file. |

#### *Syntax*

**filechangeid**
`<db> <course> <filename> <separator>`

**Example**
Change the WebCT IDs of a group of users contained in a file `changeusers.txt`.

**SAMPLE CHANGEUSERS.TXT FILE:**

```
Old ID,New ID
jsmith,jtsmith
jbrown,jkbrown
bfawlty,befawlty
arigsby,aurigsby
```

Enter the command:

UNIX `./webctdb filechangeid global xxxx changeusers.txt ","`

Windows `webctdb filechangeid global xxxx changeusers.txt ","`

## WEB-BASED INTERFACE (SERVE_WEBCTDB)

The Web-based Standard API allows data in the WebCT global database and student databases to be queried and manipulated by remote servers. For example, the Web-based interface can be used to make changes to global database records based on registration changes driven by events on another system. It can also be used to create a custom administrator interface.

**Important:**

- Some system administrators have created scripts that automatically import data files from outside systems to WebCT and export data file from WebCT to outside systems on a regular basis. By default, these scripts rely on the character set, a setting available through the system administrator interface, to determine the file character set of the file to be imported or exported. Imported files will be converted to this character set; exported files will be converted from this character set. You should change all automatic scripts to explicitly specify the character set of the files being imported to match the actual character set of the source data. This will ensure that background scripts import data smoothly, no matter the changes made to the setting for character set for imported and exported administrator files. For more information, see the *International Support* chapter of this guide.

Implementing the Web-based interface involves two steps.

1. Setting the API shared secret value
2. Developing a program to generate an HTTP request

Step 1 can be carried out by a WebCT administrator who has basic knowledge of the WebCT file system. Step 2 requires an experienced Web developer.

## 1. Setting the API Shared Secret Value

The shared secret value is used to ensure only authorized external servers are able to access the Web-based API. Once set, the shared secret value is used to create a Message Authentication Code (MAC) from the submitted data. When WebCT receives a request, it decodes the shared secret value from MAC using the submitted data. If the decoded shared secret value is the same as the one stored locally, the request is considered authentic and is processed. You can set the shared secret value by performing the following steps:

1. Using a text editor, open the file `<webct_install_directory>/webct/webct/generic/api/api_secret`
2. Change the first line of the file to your desired secret. (For security reasons, the default value *secret* does not work). You should note the following about the shared secret value.
   - It cannot exceed 256 characters.
   - It cannot contain tab, or other control characters.
   - It should not contain end-of-line characters. **Note**: By default, the UNIX text editors vi and pico automatically add end-of-line characters. Check the file size to ensure that the number of characters equals the number of bytes.
   - It is case-sensitive
3. Save the file.

Because the shared secret value has such a critical role, choose it carefully.

**Tips for Shared Secrets**

- Make your shared secret value difficult to guess by making it lengthy and by including a combination of numbers and upper and lower case characters.
- Change your shared secret value at regular intervals.
- On remote systems, place shared secret values in secure directories.

## 2. Developing a Program to Generate an HTTP Request

Developing a program to generate an HTTP request is the most substantive part of implementing the Web-based standard API. The program must:

- Generate a Message Authentication Code (MAC)
- Assemble a properly formatted HTTP request
- Process any data being returned

### Creating Message Authentication Codes

Because the Web interfaces to the Standard API reside in public directories, Message Authentication Codes (MACs) are required to ensure that only messages from trusted servers are processed.

WebCT provides three options to assist you in creating MACs:

1. A C function that you can integrate and compile into your C program
2. An executable file to which you make a system call from your program
3. Instructions for generating a MAC using a language of your choice

#### Option 1: Using the get_authentication C function

The get_authentication function generates a MAC from an array of data and a shared secret value.

The source code necessary to use the C function is located in `<webct_install_directory>/webct/webct/generic/api/security/`

A test program, which contains a Makefile for UNIX based systems, is also provided.

The file `api_security.c` contains the `get_authentication` function.

| | |
|---|---|
| get_authentication | Generates a MAC from an array of data and a shared secret value |
| Syntax | char* get_authentication (int i, char* data[], char* secret, char* encrypted_data) |
| Returns | 32-byte alphanumeric MAC |

| Parameter | Description |
|---|---|
| I | The number of elements in the array data[]. |
| Data | Array of all values to be used in generating the MAC. The data should not be URL encoded. |
| Secret | The shared secret value. |
| encrypted_data | The memory location of the MAC. It must be at least 32 bytes long. |

### OPTION 2: USING THE MESSAGE AUTHENTICATION CODE GENERATOR (GET_MD5 EXECUTABLE)

The Message Authentication Code generator generates a MAC from a shared secret value and a string consisting of the IMS ID, a timestamp, and a destination URL.

Use the Message Authentication Code generator (an executable called get_md5) if you are not working in the C language, or do not want to create a function to create the MAC. You can make a system call to get_md5 from your program and have the authentication string returned. The get_md5 executable has no dependencies on WebCT and can be copied to other servers as required. If you need a get_md5 executable for an operating system other than the one your WebCT server is running on, you can download several pre-compiled binaries for other operating systems from `http://download.webct.com`

| | |
|---|---|
| get_md5 | Generates a MAC from a shared secret value and a string to be encrypted (consisting of the IMS ID, a timestamp, and a destination URL). |
| Syntax | get_md5 <shared_secret_filename> <string_to_encrypt> |
| Returns | 32-byte alphanumeric MAC |

| Parameter | Description |
|---|---|
| shared_secret_filename | The filename where the shared secret value is stored. |
| string_to_encrypt | The string to be encrypted. The string should not be URL encoded. |

An example of using the get_md5 program to generate a MAC from a shared secret value and the data string described as follows:

Enter the command:

UNIX
```
./get_md5 api_secret  2A508D8EB5EB2D596DD937E2B8835100 982187291
http://webct.institution.edu:8900/ SCRIPT/ENGL100-
001/scripts/serve_home
```

Windows
```
get_md5 api_secret  2A508D8EB5EB2D596DD937E2B8835100 982187291
http://webct.institution.edu:8900/ SCRIPT/ENGL100-
001/scripts/serve_home
```

### OPTION 3: CREATE A MAC USING A LANGUAGE OF YOUR OWN CHOICE

If you want to create MACs within your code, (e.g. you are writing your code in Java and don't want to make a system call), you can create a MAC with the following procedure:

1. Calculate the total of the ASCII values of all the characters in the data.
2. Convert the total of the ASCII values into a string.
3. Append the shared secret value.
4. Encrypt the string into a 16-byte string using the MD5 algorithm.
5. Convert the 16-byte string into a 32-byte alphanumeric string to make it URL-friendly.

### HOW THE SERVE_WEBCTDB MAC IS GENERATED

The MAC is generated by using key/value pairs in the request. The serve_webctdb API uses the value from all key/value pairs except the ones listed below:

- AUTH
- ENCRYPTED
- USER_TYPE
- USER TYPE
- CHARSET

### ASSEMBLING THE HTTP REQUEST

There are several options for assembling an HTTP request to the Web-based standard API. The option you choose will be based on your programming language of choice and how you want to communicate with the Web server. You can issue API commands in several ways, including:

- Socket programming directly with the Web server
- Using a library which simulates a user agent
- Assembling a GET request and refreshing a browser window with the query string.

In Perl, you have the option of communicating directly with the Web server using the IO::Socket module included with most basic distributions, or installing and using a module such as LWP which simulates a user agent (e.g. a Web browser). Similar modules are available for most popular languages such as C or Java.

If you wish to refresh a user's browser window with a query string, you can do so using the "Location" HTTP header, HTML meta tags, or using JavaScript's location.replace method.

## SYNTAX

The general syntax for a Web-based request to the Standard API is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=<operation>&DB=<db>
      &COURSE=<course_id | placeholder>&AUTH=<32_byte_mac>
      [&User%20ID=<user_id> | &WebCT%20ID=<webct_id>][&IMS%20ID=<ims_id>]
      [&USER_TYPE=<1_or_0>][&ENCRYPTED=<1_or_0>][&field1=<field1>]
      [&fieldn=<fieldn>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | add | Adds a user to the global or student database.<br>If the user already exists, an error is returned. |
| | | |
| | update | Updates an existing user in the global or student database.<br>If the user does not exist, this operation returns an error. |
| | | |
| | delete | Deletes a single user from the global or student database. |
| | | |
| | find | Finds the user record based on the User ID (if searching the student database) or WebCT ID (if searching the global database). |
| | | |
| | changeid | Changes a WebCT ID. |
| | | |
| | homearea_xml | Exports a user's *myWebCT* in XML format. |
| | | |

**Notes**:

- The Standard API can accept GET or POST requests. POST requests can put their key/value pairs in the query string or in the body of the message in the appropriate format (see the W3C HTML 4.01 Specification at `http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4`
- Requests must be URL encoded (e.g. spaces should be replaced with %20)
- Key/value pairs can appear in any order
- Syntax examples represent HTTP requests directly to the Web server. If you are using a programming module to create your requests (such as LWP in Perl), many details of the request can be transparent to you.

## FUNCTIONS

## ADDING USERS

### ADDING A USER TO THE GLOBAL DATABASE

**Important:** If you are adding a user to a cross-listed course, see *Adding Users to Cross-Listed Courses*, page 95 for rules governing user roles.

Add operations have the following syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=add&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      &Password=<password>[&field1=<field1>][&fieldn=<fieldn>]
      [&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the Message Authentication Code (MAC) generator, or using custom code. |
| | | |
| WebCT ID | WebCT ID | The WebCT ID of the user being added.<br>WebCT IDs can contain alphanumeric strings, underscores, and periods. |
| | | |
| Password | Password | The password to be used for the user being added. Passwords can consist of any alphanumeric string. The API does not enforce minimum password lengths. |
| | | |
| Field1<br>…<br>Fieldn<br>(optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| ENCRYPTED (optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting. |
| | | |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Note:** The Courses field uses a colon as a delimiter between courses and a semicolon as a delimiter between user types. Thus the string "HKIN100;D:HKIN200;TA:HKIN300;S" indicates that a user is to be added to HKIN100 as a designer, HKIN200 as a teaching assistant and HKIN300 as a student. If no user type is specified, WebCT will default to adding the user as a student. Similarly, the Registered Courses field is colon delimited. For more information on the Courses and Registered courses field, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.

### Example

Add a user to the global database, and enroll them in the course ENGL100 as a designer, ENGL560 as a student, and ENGL477 as a teaching assistant.

```
GET  /webct/public/serve_webctdb?OPERATION=add&DB=global&COURSE=xxxx
      &AUTH= EB1A09F0BB299C23E99A5978587F49C1&WEBCT%20ID=pinto
      &PASSWORD=an1mal&FIRST%20NAME=Larry&LAST%20NAME=Kroger&
      COURSES=ENGL100;D:ENGL560:ENGL477;TA HTTP/1.0
```

## ADDING A USER TO THE STUDENT DATABASE

Students can be added to the student database using the following syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=add&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      &Password=<password>[&field1=<field1>][&fieldn=<fieldn>]
      [&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course to which the user will be added. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the user being added. |
| | | |
| Password | Password | The password to be used for the user being added. The API does not enforce minimum password lengths. |
| Field1<br>…<br>Fieldn<br>(optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| ENCRYPTED<br>(optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting |

| Key | Value | Description |
|---|---|---|
| | | |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear |

**Example**
Add a student to the student database of the course ENGL588. In addition, add data to a pre-existing column "StudentNumber" (This is a custom column created by the designer). Because this user is being added to the student database only, they are considered an "orphan user" until a WebCT ID is associated with this User ID:

```
GET  /webct/public/serve_webctdb?OPERATION=add&DB=student
      &COURSE=ENGL588&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=flounder&Password=an1mal&First%20Name=Kent
      &Last%20Name=Dorfman&StudentNumber=123456789 HTTP/1.0
```

## UPDATING USERS

### UPDATING A USER IN THE GLOBAL DATABASE

**Important:** If you are adding a user to a cross-listed course, see *Adding Users to Cross-Listed Courses*, page 95 for rules governing user roles.

Updating users in the global database is very similar to adding users. The syntax is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=update&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      [&FIELD1=<field1>][&FIELDN=<fieldn>][&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | Existing WebCT ID | The WebCT ID of the user being added.<br>WebCT IDs can contain alphanumeric strings, underscores, and periods. |
| | | |
| Password (optional) | Password | The password to be used for the user being updated. The API does not enforce minimum password lengths. |
| Field1<br>…<br>Fieldn (optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |

| Key | Value | Description |
|---|---|---|
| | _DELETE_ | The "_DELETE_" keyword deletes the data from the field specified in the key and sets it to undefined. |
| | | |
| ENCRYPTED (optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Notes:**

- The *User Data* setting in the WebCT administrator interface affects how updating the Courses column will modify the student database when unlinking WebCT IDs from User IDs. If the User Data setting is selected, user data is left in the student database.
- The Standard API always overwrites the Courses and Registered Course fields when updating. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you specify and unlinked from any pre-existing courses that you do not specify.
- The Courses field uses a colon as a delimiter between courses and a semicolon as a delimiter between user types. Thus the string "HKIN100;D:HKIN200;TA:HKIN300;S" indicates that a user is to be added to HKIN100 as a designer, HKIN200 as a teaching assistant and HKIN300 as a student. If no user type is specified, WebCT will default to adding the user as a student. Similarly, the Registered Courses field is colon delimited. For more information on the Courses and Registered courses field, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.

### Example

A user is currently enrolled in three courses: ENGL101 as a designer, ENGL560 as a student, and ENGL477 as a teaching assistant. This example unlinks the WebCT ID from the User ID for ENGL 560 and ENGL 477, and adds the WebCT ID to the course ENGL101 as designer.

```
GET  /webct/public/serve_webctdb?OPERATION=update&DB=global&COURSE=xxxx&AUTH=EB1A
      09F0BB299C23E99A5978587F49C1&WebCT%20ID=pinto&Courses=ENGL101;D:ENGL101;D
      HTTP/1.0
```

The user is unlinked from the two courses because API updates always overwrite fields.

#### UPDATING A USER IN THE STUDENT DATABASE

Updating students in the student database is very similar to adding students. The syntax is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=update&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      [&field1=<field1>][&fieldn=<fieldn>][&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course in which the user's data is updated. |

| Key | Value | Description |
|---|---|---|
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the user being added. |
| | | |
| Password (optional) | Password | The password to be used for the user being added. The API does not enforce minimum password lengths. |
| | | |
| Field1 … Fieldn (optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| | _DELETE_ | The "_DELETE_" keyword deletes the data from the field specified in the key and sets it to undefined. |
| ENCRYPTED (optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting. |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Example**

In the following example, a student record is updated with information for the instructor-added numeric columns "Student Participation" and "Bonus" in the course MATH100.

```
GET webct/public/serve_webctdb?OPERATION=update&DB=student&
      COURSE=MATH100&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=otter&Student%20Participation=100&Bonus=34 HTTP/1.0
```

## DELETING USERS

### DELETING A USER FROM THE GLOBAL DATABASE

The syntax for deleting a user from the global database is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=delete&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | WebCT ID | The WebCT ID of the user being deleted. |

**Note:** The *User Data* setting in the WebCT administrator interface affects whether user data is left in a course when a user record is deleted from the global database. If the *User Data* setting is selected, user data is left in the student database.

#### Example

In this example, the user record for the user with the WebCT ID `neidermeyer` is deleted from the global database:

```
GET  /webct/public/serve_webctdb?OPERATION=delete&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&WebCT%20ID=neidermeyer
      HTTP/1.0
```

### DELETING A USER FROM THE STUDENT DATABASE

The syntax for deleting a student from the student database is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=delete&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course from which the user will be deleted. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |

| | | |
|---|---|---|
| User ID | User ID | The User ID of the user being deleted. |

**Example**
In this example, the student with the User ID `stork` is deleted from the course PSYCH204-23:

```
GET  /webct/public/serve_webctdb?OPERATION=delete&DB=student
      &COURSE=PSYCH204-23&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=stork HTTP/1.0
```

## FINDING USERS

### FINDING A USER IN THE GLOBAL DATABASE

To find a user's global database record, the following syntax is used:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=find&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      [USER_TYPE=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | find | Finds the user record for a given WebCT ID |
| | | |
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | WebCT_ID | The WebCT ID of the record you want to display. |
| | | |
| USER_TYPE (optional) | 1 | With the User_Type option enabled, the global database record generated includes user type information that indicates whether a user is a designer, student, or teaching assistant for the course. |
| | 0 (default) | No user type information is generated. |

**Example**
In this example, the complete record including user type information is returned for the user with the WebCT ID `pinto`, who is enrolled in three courses.

```
GET  /webct/public/serve_webctdb?OPERATION=find&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&WebCT%20ID=pinto
      &USER_TYPE=1 HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success: WebCT ID=pinto,First Name=Larry,Last Name=Kroger,Courses=
      ENGL100;D:ENGL560;S:ENGL477;TA
```

### FINDING A USER IN THE STUDENT DATABASE

To find a student's student database record, the following syntax is used:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=find&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id> HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | find | Finds a user's record from a WebCT ID. |
| | | |
| COURSE | Any alphanumeric string | The course that you are searching. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the record you wish to display. |

**Example**
In this example, a complete student database record is displayed for the user with User ID `chip` in the course `HKIN455`:

```
GET  /webct/public/serve_webctdb?OPERATION=find&DB=student
      &COURSE=HKIN455=AUTH=EB1A09F0BB299C23E99A5978587F49C1&User%20ID=chip
      HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success: First Name=Chip,Last Name=Diller,User ID=chip,Quiz1=36,Assignment1=10
```

## CHANGING WEBCT IDS

### CHANGING A USER'S WEBCT ID

To change a WebCT ID for a user, use the syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=changeid&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&Old%20ID=<old_webct_id>
      &New%20ID=<new_webct_id> HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | changid | Changes the WebCT ID of a user. |
| | | |
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| Old ID | Old WebCT ID | The WebCT ID of the record you want to change. |
| | | |
| New ID | New WebCT ID | The WebCT ID that you want to assign to the user. |

**Example**

In this example, the WebCT ID `flounder` is changed to `dorfmank`:

```
GET  /webct/public/serve_webctdb?OPERATION=changeid&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&Old%20ID=flounder
      &New%20ID=dorfmank HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success:
```

# SECTION 2: CAMPUS EDITION INSTITUTION LICENSE

# CHAPTER 1 USER AUTHENTICATION

WebCT Campus Edition™ 4.0 provides two major methods for user authentication:

| | |
|---|---|
| Browser Based Authentication | • Users are authenticated through a browser dialog box that prompts for a username and password. The username and password are verified against WebCT internal databases. If the user is authorized, a Basic Authentication Header is provided. Subsequent page accesses to WebCT are authorized according to the browser header.<br>• This authentication method is used in previous versions of WebCT. |
| Ticket Based Authentication | • Users are authenticated through a login page that prompts for a username and password. The username and password are verified against either WebCT internal databases or against an external password database. If the user is authenticated, the user is issued a browser cookie that serves as a ticket. Subsequent page accesses to WebCT are authorized according to the ticket.<br>• Institutions that choose ticket-based authentication have the option of implementing automatic signon to WebCT. With this feature implemented, institutions that have portal solutions or other secure environments can create a seamless environment for users by pre-authenticating them into WebCT. |

## CHOOSING AN AUTHENTICATION METHOD

Many of the features of WebCT CE 4.0 require ticket-based authentication, including:

External password database authentication using LDAP, Kerberos, Windows 2000 Domain Controller, or a custom implementation

- Logout
- Server lockdown
- Automatic signon
- Session timeout
- Customizable login page

Browser based authentication is primarily provided in WebCT 4.0 as a legacy option. Choose this method of authentication if:

- Your institution has an information technology policy forbidding the use of applications that employ browser cookies.
- It is critical that the user interface of WebCT remain the same as previous versions.

### BROWSER BASED AUTHENTICATION PROCESS

Browser-based authentication has served as the standard authentication method for all previous versions of WebCT. When a user a requests a URL, authentication of the user occurs as follows:

1. The Web server checks to see if the requested URL requires authorization.
2. If none is required (e.g., a user requests a course Welcome Page), then the Web server delivers the page to the browser.
3. If authorization is required, the Web server checks to see if the user has already provided a username and password by checking to see if a valid Basic Authentication Header was provided in the request. If the header is valid, the page is delivered.
4. If the Basic Authentication Header is invalid, or no header is provided, the user is prompted with a username and password dialog box. The cycle is then repeated.

### TICKET BASED AUTHENTICATION PROCESS

When a user requests a URL, authentication of the user occurs as follows:

1. The Web server checks to see if the requested URL requires authorization.
2. If none is required, the Web server delivers the page to the browser.
3. If authorization is required, WebCT checks for a valid ticket.
4. If a valid ticket is found (i.e. the user has been authenticated and is authorized for the resource), the page is delivered to the browser.
5. If a ticket is not found, a login form is delivered to the browser. The user submits the form and WebCT authenticates the user, issuing their browser a cookie. The URL is re-requested and the cycle repeats.

#### HOW WEBCT GENERATES TICKETS

WebCT tickets (in the form of browser cookies) contain the following information:

- Username
- Encrypted Password (DES encryption)
- Timestamp (UNIX Epoch format)
- Message Authentication Code (MAC)

The MAC is generated in three steps:

1. The username, encrypted password, timestamp, user agent information (if sent), and a shared secret value are concatenated.
2. The concatenated string is encrypted with the MD5 algorithm.
3. The encrypted string is encrypted a second time with the MD5 algorithm.

## IMPLEMENTING TICKET BASED AUTHENTICATION

With ticket-based authentication, you can use one or more authentication sources. WebCT supports the following authentication sources:

- WebCT's internal database (default)
- LDAP
- Kerberos
- Windows 2000 Domain Controller
- a custom authentication source.

## CHOOSING AN AUTHENTICATION SOURCE

The authentication source(s) that you choose should be based on what your institution has already implemented. If your institution is using a centralized password management or single signon solution that is not directly supported, you can want to consider a custom implementation using WebCT's open source authentication code, written in C. For more information on custom authentication, see the section *Implementing Custom Authentication*.

The following table describes each type of authentication source.

| | |
|---|---|
| WebCT Internal Database | • This is the best option for institutions that do not have a single signon solution.<br>• This is the easiest option to deploy as there are no external systems to manage. |
| LDAP | • This is the open standard for providing directory services such as e-mail addresses, telephone numbers, addresses, etc. to the Internet.<br>• Many institutions have discovered that LDAP can also serve as an authentication database as part of a single signon environment.<br>• LDAP is not a true authentication source, so it lacks many of the features seen in purpose-built authentication sources. |
| Kerberos/Windows 2000 Domain Controller | • Kerberos is an authentication system that enables two parties to exchange private information across a network. A unique key, called a ticket, is assigned to each user who logs in to the network. |
| Custom Authentication | • If your institution uses a single signon solution that is not directly supported, (e.g. Radius, IMAP), your institution can modify WebCT's open source authentication code by using the WebCT Open Authentication Kit.<br>• To obtain the WebCT Open Authentication Kit (WOAK), contact your account representative.<br>• You will need an experienced C programmer to write the authentication function. |

### USING ONE AUTHENTICATION SOURCE

1. From the *Admin* toolbar, click **Server Mgmt**. The *Server Mgmt* toolbar appears.
2. From the *Server Mgmt* toolbar, click **Settings**. The *Administrator Settings* screen appears.
3. Under *User Authentication*, select *Use ticket based authentication*.
4. Choose whether the **Logout** link should appear in the course *Menu Bar*:
    - To display the **Logout** link, select *Display Logout link in course Menu Bar*.

- To hide the **Logout** link, deselect *Display Logout link in course Menu Bar*. **Note:** If you run WebCT in a framed environment (such as a portal) where a logout link or **Return to Portal** link already exists, you can hide the **Log Out** link.

5. In the *Ticket shared secret value* text box, either leave the shared secret value that was automatically generated by WebCT or enter a new shared secret value. For security reasons, the value *secret* does not work. The secret value
   - is case-sensitive
   - cannot exceed 256 characters
   - cannot contain tab or other control characters
   - should not contain end-of-line characters.
6. In the *Tickets remain valid for* text box, enter the number of minutes until ticket time-out. This value controls the expiry time of the ticket based on the user's last access and therefore affects how long a user can stay logged in while inactive. The default is 180 minutes.
7. If you want to specify a screen to display when users log out using WebCT's **Log Out** link, in the *URL to redirect users to after logging out* text box, enter the URL for the screen you want to display. **Note:** If you do not enter anything in this text box, WebCT's Entry Page will be displayed when users log out.
8. If you want to specify an authentication screen to appear when a session expires, in the *URL to redirect users to when authentication is required* text box, enter the URL for the authentication screen. **Note:** If you run WebCT in a framed environment (such as a portal), you can enter a URL to the authentication screen for the framed environment. If you do not specify a URL, WebCT's Entry Page will be displayed when a session expires and users will be prompted for a WebCT ID and password.
9. Choose whether to allow WebCT authentication across a domain. Authentication across a domain allows users to access all servers in the domain, without having to re-authenticate for each one.
   - To allow authentication across a domain:
     c) Select *Allow WebCT authentication across a domain*.
     d) In the *Please specify your domain* text box, enter the domain name. The domain name must have a period in front of it. Example: `.webct.com`
   - To disallow authentication across a domain, select *Do not allow WebCT authentication across a domain*.
10. Under *User is authenticated using*, from the drop-down list for the authentication source that you are using, select *First*.
11. For all other authentication sources, select *Never*.
12. Scroll to the bottom of the screen and click **Update**.

### USING MULTIPLE AUTHENTICATION SOURCES

You can integrate third-party authentication sources, such as LDAP, Kerberos, or a custom authentication source with WebCT. For example, use multiple authentication sources if your institution requires a failover authentication scheme to authenticate users who do not have an account in the primary authentication database. Users who are not authenticated by the primary authentication source can be authenticated by secondary sources, such as the internal WebCT database.

1. From the *Admin* toolbar, click **Server Mgmt**. The *Server Mgmt* toolbar appears.
2. From the *Server Mgmt* toolbar, click **Settings**. The *Administrator Settings* screen appears.

3. Under *User Authentication*, select *Use ticket based authentication*.
4. Choose whether the **Logout** link should appear in the course *Menu Bar*:
    - To display the **Logout** link, select *Display Logout link* in course Menu Bar.
    - To hide the **Logout** link, deselect *Display Logout link in course Menu Bar*. **Note:** If you run WebCT in a framed environment (such as a portal) where a logout link or **Return to Portal** link already exists, you can hide the **Logout** link.
5. In the *Ticket shared secret value* text box, either leave the shared secret value that was automatically generated by WebCT or enter a new shared secret value. For security reasons, the value *secret* does not work. The secret value
    - is case-sensitive
    - cannot exceed 256 characters
    - cannot contain tab or other control characters
    - should not contain end-of-line characters.
6. In the *Tickets remain valid* for text box, enter the number of minutes until ticket time-out. This value controls the expiry time of the ticket based on the user's last access and therefore affects how long a user can stay logged in while inactive. The default is 180 minutes.
7. If you want to specify a screen to display when users log out using WebCT's **Log Out** link, in the *URL to redirect users to after logging out* text box, enter the URL for the screen you want to display. **Note:** If you do not enter anything in this text box, WebCT's Entry Page will be displayed when users log out.
8. If you want to specify an authentication screen to appear when a session expires, in the *URL to redirect users to when authentication is required* text box, enter the URL for the authentication screen. **Note:** If you run WebCT in a framed environment (such as a portal), you can enter a URL to the authentication screen for the framed environment. If you do not specify a URL, WebCT's Entry Page will be displayed when a session expires and users will be prompted for a WebCT ID and password.
9. Choose whether to allow WebCT authentication across a domain. Authentication across a domain allows users to access all servers in the domain, without having to re-authenticate for each one.
    - To allow authentication across a domain:
        a) Select *Allow WebCT authentication across a domain*.
        b) In the *Please specify your domain* text box, enter the domain name. The domain name must have a period in front of it. Example: `.webct.com`
    - To disallow authentication across a domain, select *Do not allow WebCT authentication across a domain*.
10. Under *User is authenticated using*, specify when to use the authentication source(s):
    - If you are using the internal WebCT password database, from the corresponding drop-down list, select when it should be used in the authentication sequence. **Important:** If you are using the internal WebCT database in a failover authentication scheme, it is strongly recommended that you
        - use the WebCT database last in the authentication sequence.
        - do not use passwords that can be easily guessed (for example: *webct* or *password*).
    - If you are using LDAP:
        a) From the LDAP server drop-down list, select when it should be used in the authentication sequence.
        b) Specify the LDAP settings. See the *Specifying the LDAP Settings* section in this guide.

- If you are using Kerberos or Windows 2000 Domain Controller:
    a) From the *MIT Kerberos V5 KDC* or *Windows 2000 Domain Controller* drop-down list, select when it should be used in the authentication sequence.
    b) Specify the Kerberos settings or Windows 2000 Domain Controller settings. See the *Specifying the Kerberos Settings* section in this guide. If you are using a custom authentication source, from the corresponding drop-down list, select when it should be used in the authentication sequence.

11. Scroll to the bottom of the screen and click **Update**.

### SPECIFYING THE LDAP SETTINGS

1. Under *LDAP settings*, in the *LDAP Server Name* text box, enter the name of your LDAP server.
2. In the *LDAP Port* text box, enter the port of your LDAP server.
3. In the *Base DN* text box, enter the root directory on your LDAP server where your WebCT user records are stored. This directs the authentication program to search in the appropriate directory on your LDAP server.
4. In the *WebCT ID Attribute* text box, enter the attribute or field of the user record where the WebCT ID is stored.
5. In the *Manager DN* text box, enter the LDAP server manager's distinguished name.
6. In the *Manager Password* text box, enter the LDAP server manager's password.
7. Click **Update**.

**Important:** If you are using LDAP in a multiple authentication scheme, you must also specify the sequence in which it should be used.

### SPECIFYING THE KERBEROS SETTINGS

**Note**:

- Unix users: Kerberos requires a properly configured krb5.conf file in the generic/ticket directory.
- Windows users: Kerberos requires a properly configured krb5.ini file in the <webct_install_dir>\webct\webct\generic\ticket folder.

1. Under *Kerberos/Domain Controller* settings, in the *Realm/Domain Name* text box, enter the Kerberos Realm name. **Note:** Each entry in the KDC is called a principal and has the format:
   `username/instance@Kerberos Realm`
   Example: `johnsmith/admin@MYINSTITUTE.EDU`
   In this example, the Realm is `MYINSTITUTE.EDU`.
2. In the *Instance* text box, enter the Kerberos Instance name. In the example above, the instance is `admin`.
3. Click **Update**.

**Important:** If you are using Kerberos in a multiple authentication scheme, you must also specify the sequence in which it should be used.

### SPECIFYING THE WINDOWS 2000 DOMAIN CONTROLLER SETTINGS

1. Under *Kerberos/Domain Controller* settings, in the *Realm/Domain Name* text box, enter the Windows domain name.

2. Leave the *Instance* text box empty.
3. Click **Update**.

**Important:** If you are using *Windows 2000 Domain Controller* in a multiple authentication scheme, you must also specify the sequence in which it should be used.

## IMPLEMENTING CUSTOM AUTHENTICATION

For institutions that use external password databases that are not directly supported (e.g. Radius, IMAP), WebCT allows modification of the WebCT open source authentication code by using the WebCT Open Authentication Kit (WOAK). To obtain the WebCT Open Authentication Kit for your operating system, contact your account representative. In addition, you will need an experienced C programmer to write the authentication function.

### UNIX/LINUX

To compile the WOAK, ensure that you have the Free Software Foundation's GCC compiler installed `http://www.gnu.org/software/gcc/`. Other C compilers are not recommended.

It is beyond the scope of this guide to describe an exact procedure for code development. You should follow basic rules such as not developing on live servers, make appropriate backups of important files, and do as much testing as possible with your custom code. The following is provided as a general guide:

1. Extract the WebCT Open Authentication Kit to a working directory.
2. In the `[woak_directory]/compile` directory of the WOAK, modify the Makefile with a text editor so that your system architecture is uncommented (e.g. A Solaris developer should uncomment the `configure/solaris-sparc` line and make sure that both the `configure/linux-libc6` and `configure/aix lines` are commented out.) The Makefile is set up for Linux systems by default.
3. In the `[woak_directory]/ticket` directory, open the file `custom_auth.c`.
4. In the file `custom_auth.c`, implement the function user_is_authentic_other so that it returns AUTH_DECLINED if the user does not exist in the password database, AUTH_VALID if the password is valid, and AUTH_FAILED if the password is not valid.
5. Make any changes necessary to your Makefile in order for it to compile with your code.
6. Compile the `custom_auth.so` shared object by issuing the `make` command within the `custom_auth/compile` directory.
7. Install your custom authentication module `custom_auth.so` by copying `custom_auth.so` to the directory `[installdir]/webct/webct/generic/bin/` of your WebCT installation and enabling custom authentication in the administrator interface.

**Note:** Be sure to read the README_compile file included in the WOAK distribution. It will contain the latest instructions on how to build the custom authentication module.

## WINDOWS 2000

To compile the WebCT Open Authentication Kit (WOAK), ensure that you have Microsoft Visual Studio 6 installed and have applied the latest service packs. You can download these from `http://msdn.microsoft.com/vstudio`. Other C compilers are not recommended.

It is beyond the scope of this guide to describe an exact procedure for code development. The following is provided as a general guide.

1. Extract the WOAK into a working directory.
2. In the `<woak_directory>/ticket` directory, open the file `custom_auth.c`.
3. In the file `custom_auth.c`, implement the function `user_is_authentic_other` so that it returns AUTH_DECLINED if the user does not exist in the password database, AUTH_VALID if the password is valid, and AUTH_FAILED if the password is not valid.
4. Build the project with the command:

   ```
   msdev custom_auth.dsp /make "custom_auth - Win32 Release"
   ```

5. Install your custom authentication module by copying `custom_auth.dll` into the `<installdir>/webct/webct/generic/bin/` directory of your test installation of WebCT and enabling custom authentication in WebCT's administrator interface.

**Note:** Be sure to read the README_compile file included in the WOAK distribution. It will contain the latest instructions on how to build the custom authentication module.

# CHAPTER 2 AUTOMATIC SIGNON FROM OTHER SYSTEMS

Automatic signon allows institutions to create seamless computing environments. Users can move from an application where they are authenticated to WebCT without retyping usernames and passwords. For example, automatic signon can allow users to log in to their campus portal, browse campus events, and then click a link to their WebCT course, upon which they are automatically logged in, without being prompted for a username or password.

## AUTOMATIC SIGNON PROCESS

The automatic signon process will vary depending on the type of system with which you are integrating WebCT. For an institution that has a campus portal, the process can occur as follows:

1. A user accesses their campus portal account, using their portal username and password.
2. The portal obtains the user's *myWebCT* and/or WebCT course information either by
   - obtaining the information from an external source, such as a student information system.
   - executing a local program that makes a Standard API call to obtain the information. The local program can use one of the following API commands:
     - Standard API command `homearea_xml`, which uses the WebCT ID and server base address to export a user's *myWebCT* in XML format. See *Chapter 5: Standard API*.
     - Standard API command `find`, which uses the WebCT ID to find a user's global database record. See *Chapter 5: Standard API*.
3. The user clicks a link to the WebCT server either to their *myWebCT* or directly into a course.
4. The portal executes a local program that makes an IMS API call to find the user's IMS ID and IMS source.
5. WebCT returns the IMS ID and IMS source.
6. (Optional) The portal stores the IMS ID and IMS source locally with the user's record so that subsequent requests to the WebCT server are faster.
7. The portal executes a local program that creates a Message Authentication Code (MAC) from the data (the IMS ID and IMS source, a timestamp, and a destination URL) and the shared secret value. The local program assembles the data and MAC into an HTTP request and then sends the HTTP request via the user's browser.
8. WebCT verifies the validity of the HTTP request and issues a ticket in the form of a browser cookie.
9. The user is redirected to the URL provided in the request (e.g., the course *Homepage* or their *myWebCT*).

# IMPLEMENTING AUTOMATIC SIGNON

Implementing automatic signon involves two steps:

1. Setting shared secret values and enabling ticket based authentication
2. Developing a program to automatically authenticate a user

Step 1 can be accomplished by a WebCT administrator who has basic knowledge of the WebCT file system. Step 2 requires an experienced Web developer.

## SETTING SHARED SECRET VALUES AND ENABLING TICKET BASED AUTHENTICATION

Shared secret values are key security components for automatic signon as they are used for authenticating messages from external servers. Implementing automatic signon requires setting two shared secret values, which ensure that only messages from trusted servers are processed:

- the automatic signon secret
- the API secret

First, set the shared secret value for automatic signon:

1. Using a text editor, open the file `<webct_install_dir>/webct/webct/generic/autosignon/autosignon_secret`
2. Change the first line of the file to your desired secret. For security reasons, the value *secret* does not work.
   - It cannot exceed 256 characters.
   - It cannot contain tab or other control characters.
   - It should not contain end-of-line characters. **Note**: By default, the UNIX text editor vi and pico automatically add end-of-line characters. Check the file size to ensure that the number of characters equals the number of bytes.
   - It is case-sensitive
3. Change the first line of the file to your desired secret. For security reasons, the default value *secret* does not work. Save the file.

Now, set the API shared secret value:

4. Open the file `<webct_install_dir>/webct/webct/generic/api/api_secret`
5. Change the first line of the file to your desired secret, following the guidelines in step 2.
6. Save the file.

Now, log in to the administrator interface, and enable ticket-based authentication:

7. From the *Admin* toolbar, click **Server Mgmt**. The *Server Mgmt* toolbar appears.
8. From the *Server Mgmt* toolbar, click **Settings**. The *Administrator Settings* screen appears.
9. Under *User Authentication*, select *Use ticket based authentication.*

## DEVELOPING A PROGRAM TO AUTOMATICALLY AUTHENTICATE A USER

The most substantive part of implementing automatic signon is developing a program that automatically authenticates users into WebCT. The program must:

- find a user's IMS ID and IMS source via the IMS API
- make a request to the Automatic Signon CGI

Each of these requests requires the creation of a MAC to ensure the authenticity of the request.

### CREATING MESSAGE AUTHENTICATION CODES

Because the Web interfaces to the Standard API and automatic signon reside in public directories, Message Authentication Codes (MACs) are required to ensure that only messages from trusted servers are processed.

WebCT provides three options to assist you in creating MACs:

1. A C function that you can integrate and compile into your C program.
2. An executable file to which you make a system call from your program.
3. Instructions for generating a MAC using a language of your choice.

### OPTION 1: USING THE GET_AUTHENTICATION C FUNCTION

The get_authentication function generates a MAC from an array of data and a shared secret value.

The source code necessary to use the C function is located in `<webct_install_directory>/webct/webct/generic/api/security/`

A test program, which contains a Makefile for UNIX based systems, is also provided.

The file `api_security.c` contains the `get_authentication` function.

| | |
|---|---|
| get_authentication | Generates a MAC from an array of data and a shared secret value |
| Syntax | char* get_authentication (int i, char* data[], char* secret, char* encrypted_data) |
| Returns | 32-byte alphanumeric MAC |

| Parameter | Description |
|---|---|
| `i` | The number of elements in the array data. |
| `data` | Array of all values to be used in generating the MAC. The data should not be URL encoded. |
| `secret` | The shared secret value. |
| `encrypted_data` | The memory location of the MAC. It must be at least 32 bytes long. |

### OPTION 2: USING THE MESSAGE AUTHENTICATION CODE GENERATOR (GET_MD5 EXECUTABLE)

The Message Authentication Code (MAC) generator generates a MAC from a shared secret value and a string consisting of the IMS ID, a timestamp, and a destination URL.

If you are not working in the C language or do not want to create a function to create the MAC, use the Message Authentication Code (MAC) generator (an executable called get_md5). You can make a system call to the get_md5 executable from your program and have the authentication string returned. The get_md5 executable has no dependencies on WebCT and can be copied to other servers as required. If you need a get_md5 executable for an operating system other than the one your WebCT server is running on, you can download several pre-compiled binaries for other operating systems from `http://download.webct.com`

| | |
|---|---|
| get_md5 | Generates a MAC from a shared secret value and a string to be encrypted (consisting of the IMS ID, a timestamp, and a destination URL). |
| Syntax | get_md5 <shared_secret_filename> <string_to_encrypt> |
| Returns | 32-byte alphanumeric MAC |

| Attribute | Description |
|---|---|
| `shared_secret_filename` | The filename where the shared secret value is stored. |
| `data_to_encrypt` | The data to be encrypted. Data should not be URL encoded. |

An example of using the get_md5 program to generate a MAC from a shared secret value and the data string described as follows:

Enter the command:

UNIX
```
./get_md5 api_secret 2A508D8EB5EB2D596DD937E2B8835100 982187291
http://webct.institution.edu:8900/SCRIPT/ENGL100-
001/scripts/serve_home
```

Windows
```
get_md5 api_secret 2A508D8EB5EB2D596DD937E2B8835100 982187291
http://webct.institution.edu:8900/SCRIPT/ENGL100-
001/scripts/serve_home
```

### OPTION 3: CREATE A MAC USING A LANGUAGE OF YOUR CHOICE

If you want to create MACs within your code, (e.g. you are writing your code in Java and don't want to make a system call), you can create a MAC with the following procedure:

1. Calculate the total of the ASCII values of all the characters in the passed data.
2. Convert the total of the ASCII values into a string.
3. Append the shared secret value.

4. Encrypt the string into a 16-byte string using the MD5 algorithm.
5. Convert the 16-byte string into a 32-byte alphanumeric string to make it URL-friendly.

## FINDING THE IMS ID FOR AUTOMATIC SIGNON

To send a request to autosignon, the program must first find a user's IMS ID and IMS source. The program can find the IMS ID and IMS source by making an IMS API call using the `get_person_ims_info` operation. Optionally, after finding the IMS ID and IMS source, the program can store the IMS ID and IMS source locally so that the next time they are required, the program can read them locally and then pass them to the automatic signon CGI without having to make an API call.

### FINDING THE IMS ID USING THE WEB-BASED IMS API

The syntax for a Web-based request to the IMS API to find an IMS ID is as follows:

```
<GET | POST>  /webct/ims/serve_ep_api.pl?ACTION=configure&OPTION=get_person_ims_i
      nfo&GLOBALID=<WEBCTID>&TIMESTAMP=<unix_epoch_time>
      &AUTH=<32_byte_mac>
```

| | | |
|---|---|---|
| OPTION | get_person_ims_info | Finds a user's IMS ID and IMS source from a WebCT ID |
| | | |
| GLOBALID | An existing WebCT ID | An existing WebCT ID is required when using the `get_person_ims_info` option |
| | | |
| TIMESTAMP | UNIX epoch timestamp | Time stamp in UNIX epoch format (seconds since midnight GMT, Jan 1, 1970) |
| | | |
| AUTH | A valid MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |

When developing a program for a Web-based API, you should keep the following points in mind:

- You can use either GET or POST methods to submit requests
- Requests must be URL encoded (e.g. spaces should be replaced with %20)
- Key/value pairs must be separated by ampersands (&) signs
- Key/value pairs can appear in any order

### FINDING THE IMS ID USING THE COMMAND LINE IMS API

The IMS API executable `ep_api.pl` is in the following directory:
`<webct_install_dir>/webct/generic/ims`

The syntax for a command line request to the IMS API to find an IMS ID is as follows:

UNIX
```
./ep_api.pl configure get_person_ims_info <WEBCTID>
```

Windows
```
ep_api.pl configure get_person_ims_info <WEBCTID>
```

where:

| Argument | Input | Description |
|---|---|---|
| WEBCTID | An existing WebCT ID | A WebCT ID is required when using the `get_person_ims_info` option. |

### *Example*

Find the IMS ID for the WebCT ID jdoe:

UNIX `./ep_api.pl configure get_person_ims_info jdoe`

Windows `ep_api.pl configure get_person_ims_info jdoe`

The IMS API returns:

```
Success:

IMS id=2A508D8EB5EB2D596DD937E2B8835100

IMS source=WebCT
```

## FINDING WUUIS

### FINDING THE WUUI USING THE WEB-BASED STANDARD API

**Important**: Since the release of WebCT 3.6, the use of the `WebCT Unique Universal Identifier` (WUUI) for automatic signon and the `find_wuui` operation are deprecated. With WebCT moving towards the use of the IMS specifications, which are becoming standards in the learning community, the IMS ID and IMS source are now preferred over the WUUI. Although the use of the WUUI is deprecated, the functionality will still be supported for 4.0.

The syntax for a Web-based request to the Standard API is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=<operation>&DB=global
      &field1=<value>&COURSE=<placeholder>&AUTH=<32_byte_mac>
      HTTP/1.0
```

| Key | Value | Notes |
|---|---|---|
| `OPERATION` | `find_wuui` | Finds a user's WUUI for a given a WebCT ID |
| | | |
| | `find_ims_id_wuui` | Finds a user's WUUI for a given an IMS ID |
| | | |
| `DB` | `global` | Although this value can also be `student`, for the application of finding WUUIs, you will always use `global` |
| | | |
| `field1` | `WebCT ID` | Use if the operation is `find_wuui` |
| | `IMS ID` | Use if the operation is `find_ims_id_wuui` |

| Key | Value | Notes |
|---|---|---|
| | | |
| COURSE | Any alphanumeric string | This is a generic placeholder value. You can use any value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |

### FIND A USER'S WUUI FROM AN IMS ID

Find the WUUI for the IMS ID (person→sourcedid→id) 123456789:

```
GET  /webct/public/serve_webctdb?OPERATION=find_ims_id_wuui
     &DB=global&IMS%20ID=123456789&COURSE=xxxx
     &AUTH=EB1A09F0BB299C23E99A5978587F49C1 HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success: #WUUI = 6321BB2537BE7F1E26375D4E1687EE1F
```

### FINDING THE WUUI USING THE COMMAND LINE STANDARD API

**Important**: Since the release of WebCT 3.6, the use of the WUUI for automatic signon and the `find_wuui` operation are deprecated. With WebCT moving towards the use of the IMS specifications, which are becoming standards in the learning community, the IMS ID and IMS source are now preferred over the WUUI. Although the use of the WUUI is deprecated, the functionality will still be supported for 4.0.

The Standard API executable `webctdb` is in the following directory:
`<webct_install_dir>/webct/webct/generic/api`

The general syntax using the command line Standard API to find WUUIs is as follows:

UNIX `./webctdb <find_ims_id_wuui> global xxxx <WEBCTID | IMSID>`

Windows `webctdb <find_ims_id_wuui> global xxxx <WEBCTID | IMSID>`

Where:

- `find_ims_id_wuui` is the operation to find a WUUI using an IMS ID
- `global` is the name of the database you are accessing
- `xxxx` is a required placeholder
- `WEBCTID` is the WebCT ID of the user whose WUUI you are trying to find
- `IMSID` is the IMS ID of the user whose WUUI you are trying to find

## MAKING A REQUEST TO THE AUTOMATIC SIGNON CGI

Once the program has determined the IMS ID and IMS source, it must pass the IMS ID and IMS source to the Autosignon CGI, which then logs the user on to WebCT.

## HOW THE AUTOMATIC SIGNON MAC IS GENERATED

The MAC is generated by using key/value pairs in the request. The automatic signon CGI uses the value from the key/value pairs listed below:

- WUUI
- TIME STAMP
- URL (before URL encoding)
- IMS id (before URL encoding)
- IMS SOURCE (before URL encoding)

**Note:** The MAC will only use those keys provided in the request, as not all requests will include all keys.

The general syntax for an automatic signon request is as follows:

```
http://<webctserver>:<port>/webct/public/autosignon?IMS%20id=<IMS id>
      &Time%20Stamp=<unix_epoch_time>&URL=<url>&MAC=<32_byte_mac>
```

| Key | Value | Notes |
|---|---|---|
| IMS id | An IMS id | This is the IMS ID that has either been found using an IMS API call or is stored locally. |
| | | |
| Time Stamp | unix_epoch_time | Time stamp in UNIX epoch format (seconds since midnight GMT, Jan 1, 1970). |
| | | |
| URL | URL | The destination URL.<br>The URL parameter can be one of:<br>• a url<br>• “homearea”<br>• “coursepage://ims <sourcedid.id of course>”<br>• “courseid://webct courseid of course” |
| | | |
| MAC | 32-byte mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |

**Example 1:**

The following example describes a user being logged into the course *Homepage* of the course ID ENGL100-001:

```
http://webct.institution.edu:8900/webct/public/autosignon?
      IMS%20id=2A508D8EB5EB2D596DD937E2B8835100&Time%20Stamp=982187291
```

```
&URL=http://webct.institution.edu:8900/SCRIPT/ENGL100-
001/scripts/serve_home&MAC=0A0D776506D70AE16537560CBDE4EE1
```

The server responds by issuing the browser a cookie and sending the user to the URL, in this case the homepage for ENGL100-001.

**Example 2:**

Using the previous example, the URL consists of the `courseid`:

```
http://webct.institution.edu:8900/webct/public/autosignon?
IMS%20id=2A508D8EB5EB2D596DD937E2B8835100&Time%20Stamp=982187291
&URL=courseid://ENGL100-001&MAC=0A0D776506D70AE16537560CBDE4EE1
```

The server responds by issuing the browser a cookie and sending the user to the URL, in this case the homepage for ENGL100-001.

# CHAPTER 3 OVERVIEW OF THE APPLICATION PROGRAMMING INTERFACES

Application Programming Interfaces (APIs) allow users and other systems to directly interface with WebCT without the graphical user interface. WebCT 4.0 CE provides two APIs: the proprietary Standard API and the IMS Enterprise API. The IMS Enterprise API is exclusive to WebCT Campus Edition™ 4.0 and is compliant with the IMS Global Learning Consortium, Inc. (`http://www.imsproject.org`) enterprise API specification. Both APIs have two interfaces: a command line interface and a Web-based interface.

## DIFFERENCES BETWEEN THE IMS ENTERPRISE API AND THE STANDARD API

Some functions are available in both the IMS API and Standard API. The following table summarizes the functionality of each API.

### FUNCTIONAL DIFFERENCES

| Function | IMS API | | Standard API | |
|---|---|---|---|---|
| | Command Line | Web-Based | Command Line | Web-Based |
| Export Midterm/Final Grades from Course | √ | √ | | |
| | | | | |
| Add/Update/Delete Multiple Courses | √ | √ | | |
| | | | | |
| Add/Update/Delete Multiple Terms | √ | √ | | |
| | | | | |
| Add/Update/Delete Single Users | | | | |
| from/to global database | √ | √ | √ | √ |
| from/to the student database [1] (*Manage Students* in courses) | | | √ | √ |
| | | | | |
| Add/Update/Delete Multiple Users | | | | |
| from/to global database | √ | √ | √ | |
| from/to the student database [1] (*Manage Students* in courses) | | | √ | |
| | | | | |
| Set IMS IDs and IMS sources for users and courses | √ | √ | | |
| | | | | |
| Find Users | √ | √ | √ | √ |
| Find WUUIs (deprecated) | | | √ | √ |
| Find IMS IDs | √ | √ | | |

[1] Indirectly, the IMS API can modify the student database when importing membership objects. However, direct modification is only available via the Standard API.

| | | | | |
|---|---|---|---|---|
| Change WebCT IDs | | | √ | √ |
| Export *myWebCT* in XML format | | | √ | √ |

## OPERATIONAL DIFFERENCES

The IMS API relies on the exchange of XML data files to perform its functions. The Standard API relies on a variety of operations to carry out its functions, including the processing of delimited text files for batch operations, entering command line statements for single operations, and creating HTTP query-strings for Web-based operations.

For features unique to each API, the choice of which to use is obvious. However, for managing user accounts in WebCT, the choice is more difficult since there is an overlap in functionality. In general, whenever your goal is to communicate with an IMS-compliant application, you should use the IMS API. Some advantages and disadvantages of each API for user management are summarized below:

| | Advantages | Disadvantages |
|---|---|---|
| IMS API | • Allows you to set IMS IDs and IMS sources within WebCT<br>• Facilitates two way communication between IMS-compliant systems (which rely on the IMS ID and IMS source) | • XML files must be created |
| | | |
| Standard API | • Allows the modification of administrator-created columns<br>• Can be simpler to use for user management since it is command driven and does not require XML files | • Difficult to integrate with IMS-compliant systems |

## CHOOSING THE APPROPRIATE INTERFACE FOR YOUR REQUIREMENTS

WebCT provides two interfaces to each of its APIs: a command line interface and a Web-based interface. Choosing an interface is not a one-time decision; it will vary depending on the task that you need to accomplish. For example, if you want to process real-time updates from your institution's student information system (SIS), the Web-based interface may be more suitable. However, if you have to debug a problem, using the command line interface may be more suitable.

Use the following table as a guide for choosing the best interface.

| Task/ Situation | Suggested Interface |
|---|---|
| Processing multiple records simultaneously<br>(e.g. you want to populate the global database based on a batch extract from your institution's SIS) | command line |
| | |
| Processing a single record | command line |
| | |

| Task/ Situation | Suggested Interface |
|---|---|
| Integrating systems that are on the same physical server and run as the same user as WebCT | command line |
| | |
| Debugging | command line |
| | |
| Integrating external system with WebCT<br>(e.g., you want to integrate your institution's SIS with WebCT) | Web-based |

# CHAPTER 4 IMS ENTERPRISE API

The IMS Enterprise API in WebCT Campus Edition™ 4.0 complies with version 1.1 and 1.01 of the IMS Enterprise Specification (available at `http://www.imsproject.org/enterprise`), the latest version of the specification at the time of release. The IMS API allows you streamline integration of WebCT with other products that also conform to the IMS specification.

## IMS API ADAPTORS

The IMS API adaptors allow you to integrate your IMS-compliant systems with WebCT. These built-in adaptors take vendor-specific IMS Enterprise XML and translate it using Extensible Stylesheet Language Transformations (XSLT) for import into WebCT. The XSLT translations handle both inbound XML data and outbound grade exchange transformations.

Typical adaptor implementations include: integrating a student information system (SIS) with WebCT and importing content from earlier versions of WebCT CE. By selecting your vendor-specific adaptor, the IMS API translates your XML objects, courses and memberships, using the IMS Enterprise Specification, into WebCT-specific XML. The IMS API adaptors are described in the following table:

**IMS API Adaptors**

| Adaptor | Definition |
|---|---|
| `IMS` | The `IMS` adaptor is the default adaptor. If you do not specify an adaptor at the command line, then the `IMS` adaptor is applied. The `IMS` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. All inbound XML is run through XSLT normalization to ensure correct object order, removal of ignored elements, and conversion of deprecated extensions into the newly supported elements. |
| | |
| `webct38` | The `webct38` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. Specifying the `webct38` adaptor allows you to import WebCT 3.8 content into WebCT 4.0. |
| | |
| `datatel` | The `datatel` adaptor complies with Datatel specific XML. The adaptor provides XSLT transformation for Datatel XML and allows for Datatel specific responses. Outbound grade export (membership objects) complies with the required Datatel format. Administrators can disable grade exchange. |
| | |
| `SCT` | The `SCT` adaptor complies with SCT XSLT transformations and handles SCT-style imported data. Outbound grade export (membership objects) complies with the required `SCT` format. Administrators can disable grade exchange. |

For more information on Extensible Stylesheet Language Transformations, see `www.xslt.com`.

### SIS Grade Export

Only one adaptor can have grade export enabled at a time. You can enable and disable midterm and final grade export independent of each other.

## Functionality in the IMS Enterprise API

The IMS API has functionality that can be divided into three basic categories.

| | |
|---|---|
| Import | • Create, modify, and delete term and course instances. Both term and course instances are group objects in IMS terminology.<br>• Create, modify, and delete users. Users are person objects in IMS terminology.<br>• Register and deregister users from courses. Associates person with group objects using the membership object in IMS terminology, |
| Export | • Create an XML file of all user, course, and course membership information.<br>• Create an XML file containing basic information for a single student record.<br>• Create an XML file containing a list of users with midterm and/or final grades for a given course. |
| Configure | • Add or update the IMS source for a single course.<br>• Add or update the IMS id for a single course.<br>• Add or update the IMS source for a single user.<br>• Add or update the IMS id for a single user.<br>• Find the IMS id and IMS source for a single user. |

## Terminology

The IMS Enterprise Information Model describes data structures that are used to provide interoperability of instructional management systems like WebCT with other enterprise systems. The information model defines several data objects, and WebCT maps the appropriate internal data to these objects. You should become familiar with the IMS data objects and their relation to WebCT. The following table summarizes some of the important terminology relevant to WebCT.

**Important IMS Terminology for WebCT Users**

| Term | Definition |
|---|---|
| Group Object | An object describing a group such as a course or term instance. WebCT matches data in this object with data associated with courses such as the Course ID and Course Description. |
| | |
| Person Object | An object describing a user, such as a designer, student, or teaching assistant. IMS data elements map to WebCT elements as follows: User ID maps to WebCT ID; Family maps to Last Name; and Given maps to First Name. |
| | |
| Membership Object | An object describing the membership of a person or group within a group. WebCT uses the membership object to modify the Courses field within the global database and to add instructors, students, and teaching assistants to their appropriate course databases. |
| | |
| Properties Object | An object containing general bookkeeping information for an IMS-compliant XML file. WebCT has no equivalent to the properties object. |
| | |
| IMS id | The IMS id is a unique identifier for an IMS object. All group, person, and membership objects have an associated IMS id. Within an IMS-compliant XML file, the IMS id refers to the `<object>→sourcedid→id` |
| | |
| IMS source | The IMS source identifies the organization or system that assigned the IMS id to the object. group, person, and membership objects all have IMS sources<br>Within an IMS-compliant XML file, the IMS source refers to the `<object>→sourcedid→source`. |

## IMPLEMENTING THE IMS API

To implement the IMS API, consider the tasks that you need to perform and the systems that you want WebCT to interact with. A typical implementation of the IMS API with a student information system (SIS) might include the following steps.

1. Creation of an XML extract from a SIS that has group, person, and Member data objects within it.
2. Bulk population of the WebCT global database using the command line IMS API to import the XML extract. This creates the courses, creates the terms, creates the users, and assigns users to courses.
3. Setup of an interface between the SIS and WebCT that sends periodic updates to WebCT via the Web-based IMS API. Updates can include students dropping and adding courses, the addition of new students, and the creation of courses that have a WebCT component.
4. Transfer of midterm and/or final grades from WebCT to the SIS via the Web-based IMS API. The transfer can occur when an instructor fills out a Web form indicating that grades are ready to be released to the registrar.

## COMMAND LINE INTERFACE (EP_API.PL)

The *IMS Best Practice and Implementation Guide* describes a robust and easy-to-implement interface that involves the creation and passing of a complete "snapshot" of the person, group, and group membership data from one system to another. For example, at the beginning of a school year, an institution can export a snapshot of all student and course information from their SIS for the term. The "snapshot" can then be transferred and imported to the WebCT server.

The command line interface provides an effective way of importing "snapshots" into WebCT.

The command line interface also provides an effective way of exporting and configuring WebCT data without the development effort required for a Web-based implementation.

**Note**: The *IMS Best Practice and Implementation Guide* is available from `http://www.imsproject.org/enterprise/`

### SYNTAX

The command line interface has the following general syntax:

UNIX
```
./ep_api.pl <ACTION> <OPTION> <FILENAME | COURSEID | GLOBAL ID>
[--ims_id=<ID>] [--ims_source=<SOURCE>] [--ims_target=<TARGET>]
[--datasource=<DATASOURCE>] [--studentlist=<STUDENTLIST>]
[--charset=<characterset>][--adaptor=<ADAPTOR>]
```

Windows
```
ep_api.pl <ACTION> <OPTION> <FILENAME | COURSEID | GLOBAL ID>
[--ims_id=<ID>] [--ims_source=<SOURCE>] [--ims_target=<TARGET>]
[--datasource=<DATASOURCE>] [--studentlist=<STUDENTLIST>]
[--charset=<characterset>][--adaptor=<ADAPTOR>]
```

where:

| ACTION | OPTION |
|---|---|
| import | restrict<br>unrestrict |
| | |
| export | snapshot<br>person_record<br>group_record<br>group_final_grades<br>group_midterm_grades |
| | |
| configure | set_group_ims_info<br>import_group_ims_info<br>set_person_ims_info<br>import_group_ims_info<br>get_person_ims_info |

## FUNCTIONS

### IMPORT

**Important:** If you are adding a user to a cross-listed course, see *Adding Users to Cross-Listed Courses*, page 95, for rules governing user roles.

The syntax for an import is:

UNIX `./ep_api.pl import <OPTION> FILENAME [--adaptor=<ADAPTOR>]`

Windows `ep_api.pl import <OPTION> FILENAME [--adaptor=<ADAPTOR>]`

where:

| Argument | Input | Description |
|---|---|---|
| OPTION | `restrict` | With restrict mode on, the `sourcedid.source` and `sourcedid.id` supplied in the XML file are checked against the IMS `sourcedid` elements for similar objects to ensure they exist in the WebCT database. Objects can be updated or deleted only if the `sourcedid.source` element and `sourcedid.id` elements match. |
| | `unrestrict` | No checking of the `sourcedid.source` element is performed; only the `sourcedid.id` is checked. |
| FILENAME | `filename` | File to be imported into WebCT. |
| `--adaptor` | `IMS` | The `IMS` adaptor is the default adaptor. If you do not specify an adaptor at the command line, then the `IMS` adaptor is applied. The `IMS` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. All inbound XML is run through XSLT normalization to ensure correct object order, removal of ignored elements, and conversion of deprecated extensions into the newly supported elements. |
| | `webct38` | The `webct38` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. Specifying the `webct38` adaptor allows you to import WebCT 3.8 content into WebCT 4.0. |
| | `datatel` | The `datatel` adaptor complies with Datatel specific XML. The adaptor provides XSLT transformation for Datatel XML and allows for Datatel specific responses. |
| | `SCT` | The `SCT` adaptor complies with SCT XSLT transformations and handles SCT-style imported data. |
| `--charset` | A valid character set. See the *Appendix*. | The charset parameter overrides the character set element as defined in the XML header file. As some IMS compliant XML data is inconsistent with the character set element in the header file, it is necessary to override the element to ensure data is imported correctly. |

**Example 1**
Import the XML file `course.xml` in restrict mode using the `webct38` adaptor.

UNIX `./ep_api.pl import restrict courses.xml --adaptor=webct38`

Windows `ep_api.pl import restrict courses.xml --adaptor=webct38`

**Note:** For more examples of using import to work with cross-listed courses and to set homearea as required, see *XML File Format Guidelines* on page 90.

### Example 2

Import the XML file `isodata.xml`, containing ISO-8859-1 data and override the UTF-8 character set tag declaration as defined in the XML header. **Note:** The IMS adaptor is the default adaptor. If you do not specify an adaptor at the command line, then the IMS adaptor is applied.

| | |
|---|---|
| UNIX | `./ep_api.pl import unrestrict isodata.xml --charset=iso-8859-1` |
| Windows | `ep_api.pl import unrestrict isodata.xml --charset=iso-8859-1` |

## IMS IMPORT LOGGING

You can track the results of an IMS import by looking at log files written to during each import. A running log of all events processed can be found in `<webct_install_directory>/webct/generic/logs/ims_log.txt`. In addition, detailed log files for failed imports or imports that completed with warnings are created. The system administrator will also receive an e-mail for each failed import. The e-mail will refer to log files containing details required to reprocess an import if required. These detailed log files are referred to in this guide as "working" log files. More complete information about `ims_log.txt`, the working log files, and about automatically generated e-mail is provided in this section.

**Note:** Throughout this section, the import of multiple files will be termed an "IMS import process." Each file imported during an IMS import process will be considered an "IMS import event." An IMS import process, then, would consist of several import events.

### *ims_log.txt*

- Located in `$webct_root/webct/generic/logs/`
- Contains success and error messages related to each IMS import event, as well as export or configure events.
- Each message logged to the `ims_log.txt` file appears on a single line and has the following format:

  [<Timestamp>] [Type of call ("Web Interface" | "Console" | "IMSReceiver")>] [<Process ID number>] [<ClientMessageKey>]
  <Log message>

  **Example**
  [Fri Apr 12 09:49:39 2002] [Console] [26671] [WebCT_2002-04-12T09:49:39-0800_26670_0] Error: Cannot determine globalDB user existence. Global ID cannot be blank.

- Several messages can be related to one import event or to one import process. Each logged message falls into one of six categories, as shown below. Messages in categories 3 through 6 also generate an e-mail alert to the system administrator.

**Category 1: Informative message**
Contains comments related to an import, indicating the start of an import, import parameters, etc.

**Category 2: Success Message**
Starts with a *Success:* prefix. Success messages are logged when an operation is complete and usually mark the end of an event. Warnings can occur during a successful import, but would not interfere with completion.

**Category 3: Warning Message**
Starts with a *Warning:* prefix. Warning messages are logged when an import is successful, but did not complete as expected. In such a case, WebCT would continue processing, but may guess at the appropriate action and write to the log file so a user can check if the item was processed correctly. An example where a warning message would be logged is as follows: The XML for an import specifies that a course should be added to a non-existent term, and the course is instead added to "Default Term".

**Category 4: Error Message**
Starts with an *Error* prefix. Error messages are logged when an import process begins but an error occurs which results in an incomplete import. An example where an error message would be logged is as follows: If during the import of multiple students listed in an XML file, the Global ID of the third student in the list were not found, that particular student would not be imported. However, the import process can continue, skipping to the fourth student in the list.

**Category 5: Fatal Error Message**
Starts with a *Fatal Error* prefix. Fatal error messages are logged when an import begins but something is encountered during the process that makes it impossible to continue the processing to completion. In such a case, the import process would have begun but would immediately stop upon encountering the fatal error. An example where a fatal error message would be logged is as follows: If during an import a tag were encountered in the XML file that WebCT did not know how to deal with, the import process would stop.

**Category 6: Fatal Failure Message**
Starts with a *Fatal Failure* prefix. Fatal failure messages are logged when an import process can not begin for some reason. An example where a fatal failure message would be logged is as follows: In attempting to import a list of students, the XML file named in the import command can not be found, the processing of the XML file can not begin.

**Important:** The `ims_log.txt` file should be archived and deleted periodically so the file does not grow too large.

### *Working log files*

- Located in `generic/temp/ims_files/`
- Two log files are saved for each failed IMS import, containing details to allow reprocessing as required. An e-mail message is also sent to the system administrator as an alert. **Note:** These files are created only for import events, not export or configure events.
- The log files are named using the Client Message Key (CMK), a unique identifier for an IMS event (also known as the "Event ID"). The two files are:
  `CMK.xml`, which contains the XML data used in the import.

CMK.pairs, which contains multi-lines consisting of the processing parameters used in the import.

For example, the two files created for a failed event with the Client Message Key "WebCT_2002-04-03T17:05:02-0800_20496_0" would be:
WebCT__-2002-04-03T17_05_02-0800__-20496__-0_1.xml
WebCT__-2002-04-03T17_05_02-0800__-20496__-0_1.pairs

- These log files are known as "working" files because they are created automatically at the start of an import, named CMK.work_xml and CMK.work_pairs, and deleted automatically upon successful completion of the import. Only in case of process warnings or failure will the files be saved and renamed to <CMK>.xml and CMK.pairs.

## Content and Layout of Working Log Files

The content of the CMK.pairs log file differs depending upon if the event being logged is a console or Web interface event, or an IMS Receiver event.

### *Console or Web Interface*

Messages logged as a result of a failed console or Web Interface event run over multiple lines and have the format shown in the following example:

```
CLIENT_MESSAGE_KEY ::: WebCT_2002-04-12T11:15:16-0800_3195_0

INTERFACE_TYPE ::: Console

INLINEMODE ::: Off

INTERFACE_TYPE ::: IMS

ACTION ::: Import

IO_CHARSET :::

FILENAME ::: XML/out.xml

OPTION ::: Unrestrict
```

## E-mail Alerts

E-mail messages are generated to alert system administrators to check the working log files in three different cases:

***During or at the end of an import:***

1. If there is a warning during an import. A warning will not cause an import to fail; however, the import may not have completed as expected and so may require investigation by a system administrator to ensure the outcome is acceptable. For example, an administrator may want to reprocess an import after seeing a message such as "Warning: STATUS value is inappropriate for a deletion. Request is treated as an update for designer (John) in course (WebCT101)."
2. If an import does not complete because of an Error, a Fatal Error, or a Fatal Failure, as defined in the *ims_log.txt* section above.

***Upon system startup:***

3. If an import fails to complete for some reason other than described in cases 1 and 2.

In cases 1 and 2, the content of the e-mail message will be as follows, with the Subject line indicating a warning or an incomplete import:

```
Subject: IMS Import Process Incomplete
Importing file (<filename>) through <type of interface> failed.
(WebCT Installation on <server name>:<server port>)
The following errors (warnings) are reported:
<IMS error log for this process>
Please check the work files of that process at:
<work file path>
```

In case 3, the incomplete process will have been found upon system startup, when the working files directory is checked for temporary work files that are not owned by an active process. If such files are found, this indicates an import may not have completed. The system administrator is alerted with an e-mail. The content of the e-mail message are as follows:

```
Subject: Incomplete IMS Import Process Discovered
Possible incomplete process with Client Message Key (<Client
Message Key>) discovered.
Please check <ims_log file> for more details
(WebCT Installation on <server name>:<server port>)
Please check the work files of that process at:
<work file path>
```

In each case, after checking the work files and possibly reprocessing an import, the system administrator should delete the work files from `generic/temp/ims_files/` so they do not collect in this directory.

## EXPORT

This argument exports data from WebCT's global database. The syntax for an export is:

UNIX
```
./ep_api.pl export <OPTION> <FILENAME>
[--datasource=<DATASOURCE>][--ims_target=<TARGET>][--type=<TYPE>]
[--ims_id=<ID>][--studentlist=<STUDENTLIST>]
[--charset=<characterset>] [--adaptor=<ADAPTOR>]
```

Windows
```
ep_api.pl export <OPTION> <FILENAME>
[--datasource=<DATASOURCE>][--ims_target=<TARGET>][--type=<TYPE>]
[--ims_id=<ID>][--studentlist=<STUDENTLIST>]
[--charset=<characterset>] [--adaptor=<ADAPTOR>]
```

where:

| Argument | Input | Description |
|---|---|---|
| OPTION | Snapshot | Create an XML file of all person, group, and membership objects. |
| | person_record | Create an XML file containing basic information for a single student record. |
| | group_record | Create an XML file containing a list of users with midterm and final grade information for a given course. |
| | group_final_grades | Create an XML file containing a list of users with final grade information for a given course. |
| | group_midterm_grades | Create an XML file containing a list of users with midterm grade information for a given course. |
| | | |
| FILENAME | Any valid filename | Filename to be used for the XML file. |
| | | |
| --datasource | Any alphanumeric string up to 256 characters.<br>Enclose strings containing spaces in quotation marks. | Sets the datasource element within the properties element. Defaults to WebCT if none is specified. |
| | | |
| --ims_target | Any alphanumeric string up to 256 characters.<br>Enclose strings containing spaces in quotation marks. | Sets the target element within the properties element. |
| | | |
| --type | Any alphanumeric string up to 256 characters.<br>Enclose strings containing spaces in quotation marks. | Sets the type element within the properties object. |
| | | |
| --ims_id | Any person or group object id from the sourcedid data element. | When exporting with the person_record, group_record, group_final_grades, or group_midterm_grades options, use this optional field to specify the person or group object that you want to export. |
| | | |
| --studentlist | Any valid filename | When exporting using group_record, group_final_grades, or group_midterm_grades options, this optional file allows you to export a subset of the data. The file must be in plain text format with one IMS id per line. |

| Argument | Input | Description |
|---|---|---|
| | | |
| `--adaptor` | `webct38` | Specifying the `webct38` adaptor allows you to generate WebCT 3.8 compliant XML. |
| | `IMS` | The `IMS` adaptor is the default adaptor. If you do not specify an adaptor at the command line, then the `IMS` adaptor is applied. The `IMS` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. |
| | | |
| `--charset` | A valid character set. See the *Appendix*. | The charset parameter converts all export data into the specified character set. On export, the specified character set is tagged in the XML header file for correct identification. Character set conversion will be applied to all outgoing data.<br><br>If the charset parameter is not specified the administrator file character set will be used as the export default. Administrators can set a default through the WebCT administrator *Settings* screen. Under the *Set character set for administrator file upload/download as* option, select a default character set. Character set conversion will be applied to all outgoing data. |

**Example 1**
Export a snapshot of the WebCT global database to the file `dbsnap.xml` with the datasource set to `WebCT - Faber College` and the target set to `BigSIS`:

UNIX
```
./ep_api.pl export snapshot dbshot.xml --datasource="WebCT - Faber College" --ims_target="BigSIS"
```

Windows
```
ep_api.pl export snapshot dbshot.xml --datasource="WebCT - Faber College" --ims_target="BigSIS"
```

**Example 2**
Specifying the `webct38` adaptor, export the content from Example 1 to generate WebCT 3.8 compliant XML:

UNIX
```
./ep_api.pl export snapshot dbshot.xml --datasource="WebCT - Faber College" --ims_target="BigSIS" --adaptor=webct38
```

Windows
```
ep_api.pl export snapshot dbshot.xml --datasource="WebCT - Faber College" --ims_target="BigSIS" --adaptor=webct38
```

**Example 3**
Export a person_record to the file person.xml for the user with the IMS id `612`:

UNIX
```
./ep_api.pl export person_record person.xml --ims_id=612
```

Windows `ep_api.pl export person_record person.xml --ims_id=612`

### Example 4

Export a group_record to the file `group.xml` for a subset of students whose IMS ids are stored in the file `students.txt` for a course with the IMS id `C101`.

**Note:** The file containing the IMS ids must be in plain text format, with one IMS id per line.

UNIX
```
./ep_api.pl export group_record group.xml --
studentlist=students.txt --ims_id=C101
```

Windows
```
ep_api.pl export group_record group.xml --
studentlist=students.txt --ims_id=C101
```

### Example 5

Export a person_record to the file `person.xml` for the user with the IMS ID `612`. On export, use the charset parameter to set `person.xml` as an ISO-8859-1 file with the header <?xml version="1.0" encoding="iso-8859-1"?>:

**Note:** The same command with `-charset=utf-8` would set `person.xml` as a UTF-8 file with the header <?xml version="1.0" encoding="utf-8"?>

UNIX
```
./ep_api.pl export person_record person.xml --ims_id=612 --
charset=iso-8859-1
```

Windows
```
ep_api.pl export person_record person.xml --ims_id=612 --
charset=iso-8859-1
```

## IMS EXPORT LOGGING

You can track the results of an IMS export by looking at `ims_log.txt`, which contains a running log of all export events processed, as well as import and configure events.

**Note:** Throughout this section, the export of multiple files will be termed an "IMS export process." Each file exported during an IMS export process will be considered an "IMS export event." An IMS export process, then, would consist of several export events.

### *ims_log.txt*

- Located in `$webct_root/webct/generic/logs/`
- Contains success and error messages related to each IMS export event, as well as import or configure events.
- Several messages can be related to one export event, or to one export process. Each logged message falls into one of six categories: informative, success, warning, error, fatal error, or fatal failure. For details about each category of message, see *IMS Import Logging*, *ims_log.txt*, page 75.
- Each message logged to the `ims_log.txt` file appears on a single line and has the following format:

[<Timestamp>] [Type of call ("Web Interface" | "Console" | "IMSReceiver")>] [<Process ID number>] [<ClientMessageKey>]
<Log message>

**Example**
[Fri Apr 12 09:49:39 2002] [Console] [26671] [WebCT_2002-04-12T09:49:39-0800_26670_0] Error: Cannot determine globalDB user existence. Global ID cannot be blank.

**Important:** The `ims_log.txt` file should be periodically archived and then deleted so the file does not grow too large.

## CONFIGURE

Configure is used to set the IMS id for group objects and person objects. The syntax for a configure action is:

UNIX
```
./ep_api.pl configure <OPTION> <FILENAME | COURSEID | WEBCTID >
[--ims_id=<ID>] [--ims_source=<SOURCE>]
```

Windows
```
ep_api.pl configure <OPTION> <FILENAME | COURSEID | WEBCTID >
[--ims_id=<ID>] [--ims_source=<SOURCE>]
```

where:

| Argument | Input | Description |
|---|---|---|
| OPTION | set_group_ims_info | Sets the IMS id for a group object (course) |
| | import_group_ims_info | Sets the IMS id for multiple group objects from a file. |
| | set_person_ims_info | Sets the IMS id for a person object (user). |
| | import_person_ims_info | Sets the IMS id for multiple person objects from a file. |
| | get_person_ims_info | Finds a user's IMS id and IMS source with the WebCT ID. |
| | | |
| FILENAME | Any valid filename | A filename must be supplied for import_group_ims_info or import_person_ims_info. The file must be plain text, in the format: <webct_id>,<ims_id>,<ims_source_new> |
| | | |
| COURSEID | An existing Course ID | A WebCT Course ID is required when using the set_group_ims_info option. |
| | | |
| WEBCTID | An existing WebCT ID | A WebCT ID is required when using the set_person_ims_info option and the get_person_ims_info option. |
| | | |
| --ims_id | Any valid IMS id `sourcedid.id` | For the set_group_ims_info and set_person_ims_info options, this argument allows you to set the `group→sourcedid→id` or `person→sourcedid→id`, respectively |
| | | |
| --ims_source | Any valid IMS source | For the set_group_ims_info and set_person_ims_info |

| | | |
|---|---|---|
| | sourcedid.source | options, this argument allows you to specify the desired group→sourcedid→source or person→sourcedid→id, respectively. |

### *Example 1*

Set the sourcedid→id 0390-ENGL-101-2345 for a group object with the course ID ENGL101-2345:

UNIX
```
./ep_api.pl configure set_group_ims_info ENGL101-2345
--ims_id=0390-ENGL-101-2345
```

Windows
```
ep_api.pl configure set_group_ims_info ENGL101-2345
--ims_id=0390-ENGL-101-2345
```

### *Example 2*

Set the sourcedid→id's for three person objects contained in the file person_ims.txt.

**Note**: When viewed with a text editor, the person_ims.txt file looks like the following:

```
bluto,blutarsky123,InstitutionSIS
pinto,kroger34,InstitutionSIS
flounder,dorfman53,InstitutionSIS

ep_api.pl configure import_person_ims_info person_ims.txt
```

## IMS CONFIGURATION LOGGING

You can track the results of an IMS configure event by looking at <webct_install_directory>/webct/generic/logs/ims_log.txt, which contains a running log of all configure events processed, as well as import and export events.

**Note:** Throughout this section, the configuration of multiple files will be termed an "IMS configuration process." Each file exported during an IMS export process will be considered an "IMS configuration event." An IMS configuration process, then, would consist of several configuration events.

### *ims_log.txt*

- Located in $webct_root/webct/generic/logs/
- Contains success and error messages related to each IMS configure event, as well as import or export events.
- Several messages can be related to one configuration event or to one configuration process. Each logged message falls into one of six categories: informative, success, warning, error, fatal error, or fatal failure. For details about each category of message, see *IMS Import Logging*, *ims_log.txt* on page 75.
- Each message logged to the ims_log.txt file appears on a single line and has the following format:

  [<Timestamp>] [Type of call ("Web Interface" | "Console" | "IMSReceiver")>] [<Process ID number>] [<ClientMessageKey>]
  <Log message>

**Example**
[Fri Apr 12 09:49:39 2002] [Console] [26671] [WebCT_2002-04-12T09:49:39-0800_26670_0] Error: Cannot determine globalDB user existence. Global ID cannot be blank.

**Important:** The `ims_log.txt` file should be periodically archived and then deleted so the file does not grow too large.

## WEB-BASED INTERFACE (SERVE_EP_API.PL)

The *IMS Best Practices and Implementation Guide* defines an event driven interface as an effective method of communicating periodic updates. The event-driven interface, in which events trigger the transmission of IMS data objects to the target system (e.g. WebCT), can be implemented effectively through the Web-based interface. For example, a student adding a course can trigger an event on an SIS which then sends an IMS data object to WebCT, updating the WebCT database.

**Note**: *The IMS Best Practices and Implementation Guide* is available from `http://www.imsproject.org/enterprise`

**Important:**

- Some system administrators have created scripts that automatically import data files from outside systems to WebCT and export data file from WebCT to outside systems on a regular basis. By default, these scripts rely on character set for imported and exported administrator files, which is set in the WebCT administrator interface. Both imported and exported files will be converted to this character set. You should change all automatic scripts to explicitly specify the character set of the files being imported to match the actual character set of the source data. This will ensure that background scripts import data smoothly, no matter the changes made to the character set for imported and exported administrator files. See the *International Support* chapter of this guide for more information.
- For details about how to set the CHARSET parameter, see *Section 2: Campus Edition Institution License, Chapter 4 IMS Enterprise API,* Import, Example 2 in this guide.

Implementing the Web-based interface involves two steps.

1. Setting the API shared secret value
2. Developing a program to generate an HTTP request

Step 1 can be accomplished by a WebCT administrator who has basic knowledge of the WebCT file system. Step 2 requires an experienced Web developer.

### SETTING THE API SHARED SECRET VALUE

The shared secret value is a key component of allowing external servers to automatically sign on users to WebCT. The shared secret is used to create a Message Authentication Code (MAC) from the submitted data. When WebCT receives a request, it decodes the shared secret from MAC using the submitted data. If the decoded shared secret is the same as the one stored locally, the request is considered authentic and is processed. Because the shared secret value has such a critical role, choose it carefully. You can set the shared secret by performing the following steps:

1. Using a text editor, open the file `<webct_install_directory>/webct/webct/generic/api/api_secret`
2. Change the first line of the file to your desired secret. For security reasons, the default value *secret* does not work. The shared secret value
   - is case-sensitive
   - cannot exceed 256 characters.
   - cannot contain tab, or other control characters.
   - should not contain end-of-line characters. **Note**: By default, the UNIX text editor vi and pico automatically add end-of-line characters. Check the file size to ensure that the number of characters equals the number of bytes.
3. Save the file.

**Tips for Shared Secrets**

- Make your shared secret value difficult to guess by making it lengthy and by including a combination of numbers and upper and lower case characters.
- Change your shared secret value at regular intervals.
- On remote systems, place shared secret values in secure directories.

## Developing a Program to Generate an HTTP Request

Developing a program to generate an HTTP request is the most substantive part of implementing the Web-based IMS API. The program must:

- generate a Message Authentication Code (MAC)
- generate a checksum for submitted XML extracts
- assemble a properly formatted HTTP request
- process any data being returned

**Note:** Log files are generated for all IMS import, export and configure events. For log file details, see *Section2, Chapter 4 IMS Enterprise API, Implementing the IMS API, Command Line Interface: IMS Import Logging.*

### Creating Message Authentication Codes

Because the Web interfaces to the Standard API and automatic signon resides in public directories, Message Authentication Codes (MACs) are required to ensure that only messages from trusted servers are processed.

WebCT provides three options to assist you in creating MACs:

1. A C function that you can integrate and compile into your C program
2. An executable file to which you make a system call from your program
3. Instructions for generating a MAC using a language of your choice

#### Option 1: Using the get_authentication C Function

The get_authentication function generates a MAC from an array of data and a shared secret value.

The source code necessary to use the C function is located in `<webct_install_directory>/webct/webct/generic/api/security/`

A test program, which contains a Makefile for UNIX based systems, is also provided.

The file `api_security.c` contains the `get_authentication` function.

| | |
|---|---|
| get_authentication | Generates a MAC from an array of data and a shared secret value |
| Syntax | char* get_authentication (int i, char* data[], char* secret, char* encrypted_data) |
| Returns | 32-byte alphanumeric MAC |

| Parameter | Description |
|---|---|
| `i` | The number of elements in the array data. |
| `data` | Array of all values to be used in generating the MAC. The data should not be URL encoded. |
| `secret` | The shared secret value. |
| `encrypted_data` | The memory location of the MAC. It must be at least 32 bytes long. |

### OPTION 2: USING MESSAGE AUTHENTICATION CODE GENERATOR (GET_MD5 EXECUTABLE)

The Message Authentication Code (MAC) generator generates a MAC from a shared secret value and a string consisting of the IMS id, a timestamp, and a destination URL.

If you are not working in the C language or do not want to create a function to create the MAC, use the Message Authentication Code (MAC) generator (an executable called get_md5). You can make a system call to get_md5 from your program and have the authentication string returned. The get_md5 executable has no dependencies on WebCT and can be copied to other servers as required. If you need a get_md5 executable for an operating system other than the one your WebCT server is running on, you can download several pre-compiled binaries for other operating systems from `http://download.webct.com`

| | |
|---|---|
| get_md5 | Generates a MAC from a shared secret value and a line of data |
| Syntax | get_md5 <shared_secret_filename> <data_to_encrypt> |

| Returns | 32-byte alphanumeric MAC |
|---|---|
| **Attribute** | **Description** |
| `shared_secret_filename` | The filename where the shared secret value is stored. |
| `string_to_encrypt` | The string to be encrypted. The string should not be URL encoded. |

### OPTION 3: CREATE A MAC USING A LANGUAGE OF YOUR OWN CHOICE

If you want to create MACs within your code, (e.g. you are writing your code in Java and don't want to make a system call), you can create a MAC with the following procedure:

1. Calculate the total of the ASCII values of all the characters in the data.
2. Convert the total of the ASCII values into a string.
3. Append the shared secret value.
4. Encrypt the string into a 16-byte string using the MD5 algorithm.
5. Convert the 16-byte string into a 32-byte alphanumeric string to make it URL-friendly.

**Note**: The FILENAME field should not be passed as data.

### HOW THE SERVE_EP_API.PL MAC IS GENERATED

The MAC is generated by using key/value pairs in the request. The serve_ep_api.pl API uses the value from the key/value pairs listed below:

- ACTION
- OPTION
- TIMESTAMP
- TYPE
- SCTMODE
- DATASOURCE
- TARGET
- ID
- SOURCE
- COURSEID
- GLOBALID
- INSTRUCTOR
- INLINEMODE
- CHECKSUM (of the imported data itself)

**Note:** Because not all requests will include all keys, the MAC will only use those keys provided in the request.

### GENERATING A CHECKSUM

To ensure the integrity of XML files being transferred, WebCT uses a checksum. The checksum is created by summing the ASCII values of each character in the file (including line feeds and other control characters). As with other data, the checksum is used in the generation of the MAC. However, unlike other data, the checksum is not passed in the request to the Web server.

For example, the string `a dog\n` (where `\n` represents a line feed) has ASCII values of 97, 16, 100, 111, 103, and 10. The checksum for this string is 437, calculated by summing the values.

If you are programming in Perl, you can use the `ord()` function to get the ASCII value for a single character and then loop through the entire file. In C, you can accomplish the same task by casting each character to an integer.

### ASSEMBLING THE HTTP REQUEST

Your choice of language will determine the method that you use to assemble your HTTP request. In general, you have two basic options:

- Socket programming with the Web server
- Using a library which simulates a user agent

In Perl, you have the option of communicating directly with the Web server using the IO::Socket module included with most basic distributions, or installing and using a module such as LWP which simulates a user agent (e.g. a Web browser). Similar modules are available for most popular languages such as C or Java. Although no language is recommended over others, the examples in this guide use Perl and the IO::Socket module to communicate with the Web server.

The serve_ep_api.pl CGI will accept both GET and POST requests. However, anytime that you need to submit a file to the server, a POST request will be necessary in since GET requests are limited in length. Actions that require you to use POST are:

- Import actions
- Export actions that utilize the STUDENTLIST option
- The import_group_ims_info and import_person_ims_info actions.

The following example, written in Perl, imports an XML extract into WebCT. It generates a MAC for the data, generates a checksum for the XML extract, posts an HTTP request to the serve_ep_api.pl CGI via sockets, and prints out the response from the Web server.

```
#!/usr/bin/perl

use strict;

use LWP;
use HTTP::Request::Common;


my $remote_host = 'http://webct.institution.com';
my $SECRET_FILE = 'api_secret';
```

```
my %params;

# Uncommon to hard code the following values. This
# example simply demonstrates communication with the API.
$params{'FILENAME'} = 'export.xml';
$params{'ACTION'} = 'import';
$params{'OPTION'} = 'restrict';

# Calculate a timestamp
$params{'TIMESTAMP'} = time();

# Generate a checksum
$params{'CHECKSUM'} = &calculate_checksum($params{'FILENAME'});

# Concatenate the data into a single string so we can
# create the MAC
my $data_string = $params{'ACTION'};
$data_string .= $params{'OPTION'};
$data_string .= $params{'TIMESTAMP'};
$data_string .= $params{'CHECKSUM'};


# Make a system call to the program that generates MACs
$params{'AUTH'} = `./get_md5 $SECRET_FILE $data_string`;

# Send the request
my $ua = LWP::UserAgent->new;
my $request = POST "$remote_host/webct/systemIntegrationApi.dowebct",
    Content_Type => 'form-data',
    Content      => [ ACTION  => $params{'ACTION'},
               OPTION => $params{'OPTION'},
               TIMESTAMP => $params{'TIMESTAMP'},
               AUTH => $params{'AUTH'},
               FILENAME => [ $params{'FILENAME'} ] ];

my $response = $ua->request($request);
my $content = $response->content();
if (!$content)
{
    print "Connection to $remote_host failed\n";
}
else
{
    print $content;
}

#
#
#

sub read_file
{
    my ($filename) = @_;
    my $file_content = '';

    # Read the entire contents of the requested file
    local(*FH);
```

```
    open(FH, $filename) || return undef;
    while (my $line = <FH>)
    {
      $file_content .= $line;
    }
    close(FH);

    return $file_content;
}

sub calculate_checksum
{
    my ($filename) = @_;
    my $checksum = 0;

    # Read in the XML extract and assign it to a variable
    my $data = &read_file($filename);

    # Sum up the ASCII values of all the characters in $data
    while (defined($data) && $data ne '')
    {
      $checksum = $checksum + ord($data);
      $data = substr($data, 1);
    }

    return $checksum;
}
1;
```

## XML File Format Guidelines

The IMS Enterprise API is based on the exchange of XML files between IMS Enterprise-compliant systems. Each XML file contains one or more data objects, which each represent an operation that should occur (e.g. add a user to the database or create a new course instance). All IMS Enterprise documents have the following general structure:

```
<?xml version="1.1" encoding="UTF-8"?>

<!doctype enterprise system "ims_epv1p1.dtd" >

<enterprise>

      <object1>
      </object1>

      <object2>
      </object2>

      <objectn>
      </objectn>

</enterprise>
```

This example demonstrates a "shell" file that does not perform any actions. In this example, the three <object> tags represent placeholders that would be filled with properties, group, person, or membership data objects. You can place as many data objects as you want within the file.

## IMS OBJECTS AND WEBCT RELATIONSHIPS

All objects in an IMS-compliant XML file are based on the IMS Enterprise Information Model (available at http://www.imsproject.org/enterprise). Because the IMS Information Model is very broad and can cover a wide range of needs, WebCT only uses a subset of information from the model. The following sections outline the relationship between the IMS data objects and WebCT data**.** The IMS data object in each title in this section is followed by the WebCT data object of that type.

### PROPERTIES OBJECT: SYSTEM IDENTIFIER

The properties object contains some general packaging and control data for use by the target system (WebCT). The following is a fragment from an XML file showing a properties object:

```
<properties>
  <datasource>Faber College SIS</datasource>
  <datetime>2000-12-21</datetime>
</properties>
```

The following table describes data elements relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| datasource | Yes | The properties→datasource element is the identifier of the system that generated the XML file. |
| | | |
| datetime | Yes | Although not used by WebCT, a datetime element with date and time in ISO 8601 standard format is required for IMS-compliance. |

### GROUP OBJECT: COURSE

The following XML example describes a group object:

```
<group recstatus="1">
      <sourcedid>
        <source>Faber College SIS</source>
        <id>0390COMPSCI697CSec1-1164</id>
      </sourcedid>
      <description>
        <short>Security In Computing</short>
      </description>
      <extension>
        <template>Blank</template>
      </extension>
</group>
```

The following table describes data elements that are relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| recstatus | No | This describes the type of action to be performed on an object. Numbers are used for language independence:<br>1 = Add, 2 = Update, and 3 = Delete.<br>If no recstatus is supplied, the API will default to 1 (Add) if the record does not already exist, or 2 (Update) if the record does exist. |
| | | |
| source | Yes | The group→sourcedid→source element is used when importing in restrict mode. This value is compared against the IMS sourcedid.source element for the item on the WebCT server to determine if an object will be processed. |
| | | |
| id | Yes | The group→sourcedid→id element is used as the WebCT Course ID. |
| | | |
| short | Yes | The group→description→short element is used as the Course Title within WebCT. |
| | | |
| template | No | The group→extension→template element allows you to specify a course template or course ID as the basis for another course. It is processed in the following order:<br>1. If no template element is specified, use your adaptor-defined template.<br>2. If this element has the text "Blank," a blank template is used.<br>3. If a valid Course ID is supplied, the course is created based on it.<br>4. If a valid course template ID is supplied, it is used. Valid course template IDs are "blank", "photo_basic"(basic), "photo"(intermediate), and "photo_comprehensive"(advanced)<br>5. If an invalid value is supplied, use the default template. |

### ORGUNIT OBJECT: CATEGORIES

The IMS API provides a means to add a course to a category.

#### *Assigning a course to a category*

The following XML example describes the placement of a course in a category:

```
<group>
   <sourcedid>
      <source>Faber College SIS</source>
      <id>0390COMPSCI697CSec1-1164</id>
   </sourcedid>
   <description>
      <short>CS-697</short>
      <long>Security In Computing</long>
```

```
    </description>
    <org>
       <orgunit>Physics</orgunit>
    </org>
 <group>
```

The following table describes data elements that are relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| `source` | | The `group→sourcedid→source` element is used when importing in restrict mode. This value is compared against the IMS `sourceid.source` element for the item on the WebCT server to determine if an object will be processed. |
| `id` | | The `group→sourcedid→id` element is used as the WebCT ID and is stored internally. The WebCT ID can be overridden by the `userid` element. |
| `short` | Yes | This is the course title. |
| `long` | No, but recommended | This is the course description. |
| `org` | No | Needed when assigning a course to a category. |
| `orgunit` | Yes | The course will be assigned to the Category with this name. If your institution is organized by departments, the `orgunit` can be Physics or Psychology. |

### RELATIONSHIP OBJECT: CROSS-LISTED COURSES

A cross-listed course is associated with another course or other courses and is taught at the same time, with the same instructor, and in the same location. One course is designated as the master course and it contains all of the course content and user information. All course activity takes place in the master course. The other course is designated as an alias course and it contains only a reference to the master course. For example, Math 211 can be listed as Math 217 and Math 245. The courses would be taught as one by the same instructor. When users access a cross-listed course, WebCT will re-direct them to the master course. The IMS treats cross-listed courses as *also known as* (*aka*) relationships.

In the administrator interface, a master course is marked with an "M" and an alias course is marked with an "A". Each course has a different Course ID. It is recommended that you keep a list of cross-listed courses. The list allows you to keep track of cross-listed courses and distinguish them from standalone courses.

On *myWebCT*, designers will see courses in which they are registered as follows:

- If the designer is registered in only one course in a cross-listed set, that course (whether the master course or an alias course) is displayed.
- If the designer is registered in more than one course in a cross-listed set, the course in which the designer was first registered (whether the master course or an alias course) is displayed with the alias courses listed below it. The alias courses are marked *Also known as*.

You can use the IMS API to create, delete, and get information about cross-listed courses.

### *Creating Cross-Listed Courses*

To create cross-listed courses, you must first create the courses, cross list them, and then add students to the courses. If you want to use content from a course backup, you must first restore the backup file into what will become the master course before you cross-list it. **Important:** You must cross list courses before students start any course activities, such as taking a quiz or submitting an assignment. You must also place all courses in the same cross-listed set in the same term.

There are two methods for creating cross-listed courses:

- Create the courses through the WebCT administrator interface, and cross list the courses through the IMS API. Example XML to cross list courses is provided in this section.
- Create and cross list the courses through the IMS API only. Example XML to create courses is found in this guide on page 74. Example XML to cross list courses is provided in this section.

We recommend that you create cross-listed courses in the following order as it allows information to be processed faster:

1. Create the courses
2. If applicable, restore the course backup into the course that will become the master course.
3. Cross list the courses
4. Add students to the courses.

The following XML example describes how to cross list three courses: Math 211, Math 217, and Math 245. The first cross-listed course in the XML, which is Math 211, becomes the master course.

```
<group>
    <sourcedid>
      <source>crosslist test</source>
      <id>MATH211</id>
    </sourcedid>
    <description>
      <short>MATH 211</short>
    </description>
    <relationship relation="3">
      <sourcedid>
            <source>crosslist test</source>
            <id>MATH217</id>
      </sourcedid>
    </relationship>
    <relationship relation="3">
      <sourcedid>
            <source>crosslist test</source>
            <id>MATH245</id>
      </sourcedid>
    </relationship>
  </group>
```

### *Data Elements Relevant to WebCT*

The following table describes data elements that are relevant to WebCT.

| Data Element | Required | Description |
|---|---|---|
| recstatus | No | This describes the type of action to be performed on an object. Numbers are used for language independence:<br>1 = Add, 2 = Update, and 3 = Delete.<br>If no recstatus is supplied, the API will default to 1 (Add) if the record does not already exist, or 2 (Update) if the record does exist. |
| source | Yes | The group→sourcedid→source element is used when importing in restrict mode. This value is compared against the IMS sourceid.source element for the item on the WebCT server to determine if an object will be processed.<br><br>If the course was created through the WebCT administrator interface, the IMS source of the course will be WebCT. To find the IMS source of a WebCT course, export the group_record for the course and examine the resulting XML. |
| id | Yes | The group→relationship→sourcedid→id element refers to the group→sourcedid→id of this course.<br><br>The IMS ID is required to identify the course to be cross-listed. To find the IMS ID, export the group_record for the course and examine the resulting XML. |
| short | Yes | The sourceid→description→short element is the course title and is a mandatory field in the group object. |
| relationship | Yes | The group→relationship element defines when a section is related to another section or term.<br><br>Attributes:<br>• relation (optional): defines the nature of the relationship. Where, relation="3" indicates that the course will become a member of a cross-listed group. |

### *Adding Users to Cross-Listed Courses*

You can add users to cross-listed courses through the WebCT administrator interface, the Standard API, or the IMS API. We strongly recommend that you cross list the courses before you add users to them.

**Important:** The following rules apply to user roles in cross-listed courses:

- A student can be added to one and only one course in a cross-listed set.
- A teaching assistant (TA) who is added to one or more courses in a cross-listed set cannot be a student or designer in any other course in the set. When a TA is added to one cross-listed course, they also have TA access to all courses in the set, with access to all student records.

- A designer who is assigned to one or more courses in a cross-listed set cannot be a student or teaching assistant in any other course in the set. When a designer is assigned to one cross-listed course, they also have designer access to all courses in the set, with access to all student and TA records.

Instructions for adding users to courses are available in the following:

- To use the WebCT administrator interface, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.
- To use the Standard API, see this guide, Section 1, Chapter 2,Functions: Adding users.
- To use the IMS API, see this guide, Section 2, Chapter 4, Functions: Import

### Deleting Cross-Listed Courses

You can delete either a master course or an alias course from a set of cross-listed courses either through the WebCT administrator interface or through the IMS API. If you delete an alias course, it will no longer be associated with the master course or other alias courses. If you delete a master course, all course content and user information is moved to an alias course, which becomes the master course. **Important:** If you want to delete all the courses in a cross-listed set, it is strongly recommended that you delete alias courses before the master course, as this requires less data migration. Use the *Course Profile* screen in the administrator interface to determine the master course.

Instructions for deleting a course and its students are available in the following:

- To use the WebCT administrator interface, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.
- To use the IMS API, see this guide, Section 2, Chapter 4, XML File Format Guideline: Group object: Course.

### Deleting Cross-listed Relationships

If you want to delete the cross-listed relationship between a course and other courses in a cross-listed stand maintain the course as a standalone course, you use the IMS API. **Warning:** Do not delete cross-listed relationships after students have started course activities, such as taking a quiz or submitting an assignment.

The following XML examples show the deletion of cross-listed relationships in a set consisting of master course Math 211 and alias courses Math 217 and Math 245:

#### Example 1

The following XML describes the deletion of Math 217 from the cross-listed set. Math 217 becomes a standalone course.

```
<group>
   <sourcedid>
     <source>crosslist test</source>
     <id>MATH217</id>
   </sourcedid>
   <description>
     <short>Mathematics 217</short>
   </description>
   <extension>
     <deleted_relationships>
      <relationship relation="3">
        <sourcedid>
            <source>crosslist test</source>
```

```
            <id>MATH217</id>
          </sourcedid>
        </relationship>
      </deleted_relationships>
    </extension>
</group>
```

**Example 2:**

The following XML fragment describes the deletion of Math 217 and Math 245 from the cross-listed set. Math 217 and Math 245 become standalone courses.

```
<group>
   <sourcedid>
     <source>crosslist test</source>
     <id>MATH211</id>
   </sourcedid>
   <description>
     <short>Mathematics 211</short>
   </description>
   <extension>
     <deleted_relationships>
      <relationship relation="3">
        <sourcedid>
            <source>crosslist test</source>
            <id>MATH217</id>
        </sourcedid>
      <relationship relation="3">
        <sourcedid>
            <source>crosslist test</source>
            <id>MATH245</id>
        </sourcedid>
      </relationship>
     </deleted_relationships>
   </extension>
</group>
```

### *Data Elements Relevant to WebCT*

The following table describes data elements that are relevant to WebCT.

| Data Element | Required | Description |
|---|---|---|
| `recstatus` | No | This describes the type of action to be performed on an object. Numbers are used for language independence:<br>1 = Add, 2 = Update, and 3 = Delete.<br>If no `recstatus` is supplied, the API will default to 1 (Add) if the record does not already exist, or 2 (Update) if the record does exist. |
| `source` | Yes | The `group`→`sourcedid`→`source` element is used when importing in restrict mode. This value is compared against the IMS `sourceid.source` element for the item on the WebCT server to determine if an object will be processed. |

| | | |
|---|---|---|
| | | If the course was created through the WebCT administrator interface, the IMS source of the course will be `WebCT`. To find the IMS source of a WebCT course, export the group_record for the course and examine the resulting XML. |
| `id` | Yes | The `group→sourcedid→id` element refers to the `group→sourcedid→ims id` of this course. |
| `short` | Yes | The `sourceid→description→short` element is the course title and is a mandatory field in the group object. |
| `relationship` | Yes | The `relationship` element must contain the `source` and `id` of the course to be cross-listed.<br><br>The IMS ID is required to identify the course to be cross-listed. To find the IMS ID, export the group_record for the course and examine the resulting XML. |
| `deleted_relationships` | No | The `group→extension→deleted_relationships` element is used to specify the course relationships to be deleted. **Note:** `deleted_relationships` is a new WebCT extension to the IMS specification.<br><br>Attributes:<br>`relation` (optional): defines the nature of the relationship. Where `relation="3"` indicates that the course will become a member of a cross-listed group. |

### *Backing up and Restoring Cross-Listed Courses*

For information on backing up and restoring cross-listed courses, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 3, Course Management*, "*Backing Up and Restoring Courses.*"

## PERSON OBJECT: USER

The following XML example describes a person object:

```
<person recstatus="1">
      <sourcedid>
        <source>Faber College SIS</source>
        <id>39450210223</id>
      </sourcedid>
      <userid password="ToughPassword">HooverR</userid>
      <name>
      <fn>Robert Hoover</fn>
        <n>
              <family>Hoover</family>
              <given>Robert</given>
        </n>
```

```
      </name>
</person>
```

The following table describes data elements relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| recstatus | No | This describes the type of action to be performed on an object. Numbers are used for language independence.<br>1 = Add, 2 = Update, and 3 = Delete.<br>If no recstatus is supplied, the API will default to 1 (Add) if the record does not already exist, or 2 (Update) if the record does exist. |
| | | |
| source | Yes | The person→sourcedid→source element is used when importing in restrict mode. This value is compared against the IMS sourceid.source element for the item on the WebCT server to determine if an object will be processed. |
| | | |
| id | Yes | The person→sourcedid→id element is used as the WebCT ID and is stored internally. The WebCT ID can be overridden by the userid element. |
| | | |
| userid | No | The person→userid element specifies a WebCT ID.<br><br>Attributes:<br>• password (optional). The password assigned to the user for accessing WebCT.<br>• pwencryptiontype (optional). DES encryption is applied to the password. |
| | | |
| fn | Yes | The person→name→fn (Formatted Name) element is required by the IMS Specification and is stored internally. It is only used for compliance. |
| | | |
| family | No | The person→name→n→family element maps to the Last Name field within the WebCT global database. |
| | | |
| given | No | The person→name→n→given element maps to the First Name field within the WebCT global database. |

### MEMBERSHIP OBJECT: USER TYPE

The following XML example describes a membership object with two member elements underneath it (one student and one designer).

To set the status of a user (designer, TA, or student), use the `STATUS` tag. If the user's status is inactive, enter a value of `0`. If the user's status is active, enter a value of `1`. The default status of a user is active. The status of a user can be set only through the IMS API.

Setting the status of a user to inactive results in the following:

- The course does not display in the user's *myWebCT*.
- The user's data remains in the course.
- The ID relationship is kept.
- The user is not counted by the license server as only actives users are counted.
- The user in a cross-listed course will be made inactive in all courses in the cross-listed set.

Students or TAs who are denied access to a course by the designer retain an active status.

```
<membership>
  <sourcedid>
        <source>Faber College SIS</source>
        <id>0390COMPSCI697CSec1-1164</id>
  </sourcedid>
  <member>
        <sourcedid>
              <source>Faber College SIS</source>
              <id>39450210223</id>
        </sourcedid>
        <idtype>1</idtype>
        <role recstatus="1" roletype="01">
              <userid>39450210223</userid>
              <status>1</status>
              <finalresult>
                    <result>B</result>
              </finalresult>
              <interimresult>
                    <result>A</result>
              <interimresult>
        </role>
  </member>
  <member>
        <sourcedid>
              <source>Faber College SIS</source>
              <id>012345678910</id>
        </sourcedid>
        <idtype>1</idtype>
        <role recstatus="1" roletype="02">
              <userid>12345678910</userid>
              <subrole>Primary</subrole>
              <status>1</status>
        </role>
  </member>
</membership>
```

The following table describes data elements relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| membership→sourcedid→id | Yes | This membership→sourcedid→id should match an existing group→sourcedid→id element. Users will be manipulated in the WebCT course instance that this matches. |
| | | |
| member | No | The membership→member element can be repeated multiple times to associate multiple users with the group specified in membership→sourcedid→id. |
| | | |
| membership→member →sourcedid→source | Yes | The membership→member→sourcedid→source element is used when importing in restrict mode. This value is compared against the IMS sourceid.source element for the item on the WebCT server to determine if an object will be processed. |
| | | |
| membership→member →sourcedid→id | Yes | This id should match an existing person→sourcedid→id. This is the person object being added, updated, or deleted within the course. |
| | | |
| idtype | Yes | The membership→member→idtype element is required according to the IMS specification and indicates if the member is a person (indicated by a "1") or a group object (indicated by a "2"). WebCT only supports person objects. |
| | | |
| recstatus | No | This describes the type of action to be performed on an object. Numbers are used for language independence.<br>1 = Add, 2 = Update, and 3 = Delete.<br>If no recstatus is supplied, the API will default to 1 (Add) if the record does not already exist, or 2 (Update) if the record does exist. |
| | | |
| roletype | Yes | The membership→member→role→roletype determines what type of user this person object should be. 01 = Learner/Student, 02 = Instructor. |

| Data Element | Required | Description |
|---|---|---|
| | | Other numbers listed in the IMS Specification are not currently supported. |
| | | |
| `userid` | Conditional | The `membership→member→role→userid` is a required element if the associated person object has a `userid` associated with it. The `userid` in both objects must match or an error will be returned. |
| | | |
| `subrole` | No | For instructors/designers, a `membership→member→role→subrole` can be specified as "Primary" or "Subordinate" that maps to primary and secondary designers in WebCT. The IMS specifications also suggest that "teaching assistant" can be used as a `subrole`. However, WebCT's IMS API does not yet support teaching assistants. |
| | | |
| `status` | Yes | This `membership→member→role→status` element affects all users (and must be included for IMS-compliance). "1" indicates that a user is active, and "0" indicates inactive. Users with inactive status will not see the course listed in their *myWebCT* but their data will remain in the course. The license server only counts users with an active status. |
| | | |
| `membership→member→role →finalresult→result` | No | The `membership→member→role→finalresult →result` element maps to the *Final Grade* column in *Manage Students*. |
| `membership→member →role →interimresult→result` | No | This element maps to the *Midterm* column in *Manage Students*. |

### SPECIFIC GROUP OBJECT: TERMS

The IMS API provides a means to add, update, and delete terms, as well as to add a course to a term. Functionality related to terms is also provided through the administrator interface. Note that terms initially created through the IMS API cannot be deleted through the administrator interface; deletion must be performed using the IMS API.

#### *Adding a Term*

The following XML example describes two term objects. To add a term, prepare an XML file using these guidelines and add to the database using the 'import' command.

```
<group>
   <sourcedid>
      <source>Faber College SIS</source>
      <id>2002-Summer</id>
   </sourcedid>
   <grouptype>
      <typevalue level="2">Term</typevalue>
   </grouptype>
   <description>
      <short>2002 Term 2</short>
      <long>Summer 2002</long>
   </description>
</group>
<group>
   <sourcedid>
      <source>Faber College SIS</source>
      <id>2002-Fall</id>
   </sourcedid>
   <grouptype>
      <typevalue level="2">Term</typevalue>
   </grouptype>
   <description>
      <short>2002 Term 3</short>
      <long>Fall 2002</long>
   </description>
</group>
```

The following table describes data elements that are relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| `source` | Yes | The `group` →`sourcedid`→`source` element is used when importing in restrict mode. This value is compared against the IMS `sourceid.source` element for the item on the WebCT server to determine if an object will be processed. |
| `id` | | The `group`→`sourcedid`→`id` element is used as the WebCT ID and is stored internally. The WebCT ID can be overridden by the `userid` element. |
| `typevalue` | Yes | The `group`→`typevalue` element is required to indicate that the group object is a term. Each `typevalue` element requires a level to be defined. Level="1" specifies this is an instructional group object. Level="2" further specifies the instructional group object (in the example given) as a term object. |
| `short` | Yes | Sorting key for term element. For example, Spring, Summer, Fall, Winter terms can be assigned values 1, 2, 3, and 4 respectively to ensure the terms are displayed in the correct order in the WebCT interface. |
| `long` | No, but recommended | Term title that will appear in WebCT |

### *Assigning a course to a term*

The following XML example describes the placement of a course in a term.

```
<group>
  <sourcedid>
      <source>Faber College SIS</source>
      <id>0390COMPSCI697CSec1-1164</id>
  </sourcedid>
  <description>
      <short>CS-697</short>
      <long>Security In Computing</long>
  </description>
  <relationship relation="1">
    <sourcedid>
      <source>Faber College SIS</source>
      <id>2002-Summer</id>
    </sourcedid>
  </relationship>
<group>
```

The following table describes data elements that are relevant to WebCT:

| Data Element | Required | Description |
|---|---|---|
| `source` | | The `group→sourcedid→source` element is used when importing in restrict mode. This value is compared against the IMS `sourceid.source` element for the item on the WebCT server to determine if an object will be processed. |
| `id` | | The `group→sourcedid→id` element is used to generate the Course ID and is stored internally. |
| `short` | Yes | This is the course title. |
| `long` | No, but recommended | This is the course description. |
| `relationship` | Yes | The `group→relationship` element defines when a section is related to another section or term.<br><br>Attributes:<br>• `relation` (optional): defines the nature of the relationship. Where, `relation="1"` indicates the course specified should be related to the term indicated. |
| `source` | | The `group→relationship→source` element is used when importing in restrict mode. This value is compared against the IMS `sourceid.source` element for the term to which the course is being assigned. |
| `id` | | The `group→relationship→sourcedid→id` element refers to the `group→sourcedid→id` of the term this course is being assigned to. |

### COMPLETE SPECIFICATIONS

A complete discussion of the IMS Enterprise Information Model is beyond the scope of this guide. You should refer to the IMS Enterprise Information Model, Version 1.1, available at `http://www.imsproject.org/enterprise`.

### OTHER XML CONSIDERATIONS

In addition to following the IMS specification for XML, WebCT requires that documents are well formed and follow XML convention. Any error that causes WebCT's XML parser to fail will result in the entire API action failing with errors generated to standard error (on screen for command line operations and to the Apache error logs for Web-based requests). Applications that generate XML markup should ensure that they are following the XML 1.0 specification (available from the W3C at `http://www.w3.org/TR/REC-xml`).

### SYNTAX

The general syntax for a Web-based request to the IMS API is as follows:

```
<GET | POST> /webct/systemIntegrationApi.dowebct
      ?ACTION=<action>&OPTION=<option>
      &TIMESTAMP=<unix_epoch_time>&AUTH=<32_byte_mac>
      [&INLINEMODE=<inlinemode>][&COURSE=<course>]
      [&DATASOURCE=<datasource>][&TARGET=<target>][&ID=<id>]
      [&CHARSET=<characterset>] [&SOURCE=<source>][&ADAPTOR=<adaptor>] HTTP/1.0
```

**Note:** WebCT recommends you update your scripts to reflect the new Web-based IMS API URL. The former URL `/webct/ims/serve_ep_api.pl` still functions as documented previously.

```
[--Boundary_Value_Of_Your_Choosing]
[Content-Disposition: form-data; name="FILENAME"; filename="<filename>"
Content-Type: text/xml | Content-Type: text/plain

<file_content>]
[--Boundary_Value_Of_Your_Choosing]
[Content-Disposition: form-data; name="STUDENTLIST";
      filename="<studentlistfilename>"
Content-Type: text/plain

<studentlist_file_content>]
```

where:

| ACTION | Description |
|---|---|
| import | Imports an XML extract into the WebCT global database. |
| | |
| export | Exports an XML extract from the WebCT global database. |
| | |
| configure | Configures the IMS ID for a person or group object |

**Notes**:

- Although either GET or POST methods can be used, any request that requires the transfer of a file requires you to use POST
- Requests using the POST method can pass the key/value pairs to the Web server using the application/x-www-form-urlencoded content type or via the multipart/form-data content type within the body of the message. If you are uploading a file, the multipart/form-data content-type is required.
- Syntax examples represent HTTP requests directly to the Web server. If you are using a programming module to create your requests (such as LWP in Perl), many details of the request can be transparent to you.
- Because POSTing in multipart/form-data is extremely verbose, syntax examples have been consolidated so that only files to be uploaded appear in full syntax. Other key/value pairs are presented in the query string. In practice, your requests must send all data in the message body when POSTing.

## FUNCTIONS

### IMPORT

This action imports data to the WebCT global database. The syntax for an import is:

```
POST  /webct/systemIntegrationApi.dowebct?ACTION=import&OPTION=<option>
      &TIMESTAMP=<unix_epoch_time>&AUTH=<32_byte_mac>
      &CHECKSUM=<checksum>[&ADAPTOR=<adaptor>] HTTP/1.0

--Boundary_Value_Of_Your_Choosing
Content-Disposition: form-data; name="FILENAME" filename="<filename>"
Content-Type: text/xml

<file_content>
--Boundary_Value_Of_Your_Choosing--
```

where:

| Key/ Parameter | Value | Description |
|---|---|---|
| OPTION | restrict | With restrict mode on, the `sourcedid.source` and `sourcedid.id` supplied in the XML file are checked against the IMS `sourcedid` elements for similar objects to ensure they exist in the WebCT database. Objects can be updated or deleted only if the `sourcedid.source` element and `sourcedid.id` elements match. |
| | unrestrict | No checking of the `sourcedid.source` element is performed; only the `sourcedid.id` is checked. |
| | | |
| TIMESTAMP | UNIX epoch timestamp | Time stamp in UNIX epoch format (seconds since midnight GMT, Jan 1, 1970) |
| | | |
| AUTH | A valid MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |

| Key/ Parameter | Value | Description |
|---|---|---|
| | | |
| ADAPTOR | IMS | The IMS adaptor is the default adaptor. If you do not specify an adaptor at the command line then the IMS adaptor is applied. The IMS adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. All inbound XML is run through XSLT normalization to ensure correct object order, removal of ignored elements, and conversion of deprecated extensions into the newly supported elements. |
| | webct38 | The webct38 adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. Specifying the webct38 adaptor allows you to import WebCT 3.8 content into WebCT 4.0. |
| | datatel | The datatel adaptor complies with Datatel specific XML. The adaptor provides XSLT transformation for Datatel XML and allows for Datatel specific responses. |
| | SCT | The SCT adaptor complies with SCT XSLT transformations and handles SCT-style imported data. |
| | | |
| <filename> | A valid filename | Filename to be imported into WebCT. |
| | | |
| <file_content> | Contents of XML extract. | The contents of an IMS-compliant XML file. |

**Example**

This example imports an XML file that includes a single person object.

```
POST /webct/systemIntegrationApi.dowebct HTTP/1.0
Content-length: 1253
Content-type: multipart/form-data; boundary=WebCT_Enterprise_API_Boundary

--WebCT_Enterprise_API_Boundary
Content-Disposition: form-data; name="ACTION"

import
--WebCT_Enterprise_API_Boundary
Content-Disposition: form-data; name="OPTION"

restrict
--WebCT_Enterprise_API_Boundary
Content-Disposition: form-data; name="TIMESTAMP"

984693507
--WebCT_Enterprise_API_Boundary
Content-Disposition: form-data; name="AUTH"

68056FBF19C2C5FE3B7AB63A06B9A009
--WebCT_Enterprise_API_Boundary

Content-Disposition: form-data; name="FILENAME"; filename="event.xml"
Content-Type: text/xml
```

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE ENTERPRISE SYSTEM "IMS-EP01.dtd" >

<enterprise>
      <properties lang="en">
        <datasource>Faber College SIS</datasource>
        <datetime>2001-03-04</datetime>
      </properties>

      <person>
        <sourcedid>
              <source>Faber College SIS</source>
              <id>Hoover26</id>
        </sourcedid>
        <userid password=”ToughPassword”>rhoover</userid>
        <name>
              <fn>Robert Hoover</fn>
              <n>
                    <family>Hoover</family>
                    <given>Robert</given>
              </n>
        </name>
        <datasource>Faber College SIS</datasource>

      </person>
</enterprise>

--WebCT_Enterprise_API_Boundary—
```

The Web server returns (not including HTTP headers):

```
Success: Data successfully imported.
Success: Import complete.
```

### EXPORT

This action exports data from the WebCT global database. The syntax for an export is:

```
<GET | POST>  /webct/systemIntegrationApi.dowebct?ACTION=export&OPTION=<option>
      &TIMESTAMP=<unix_epoch_time>&AUTH=<32_byte_mac>
      [&DATASOURCE=<datasource>]
      [&TARGET=<target>][&TYPE=<type>][&ID=<id>]
      [&STUDENTLIST=<studentlist>]
      [&STUDENTLISTCHECKSUM=<studentlistchecksum>]
      [&CHARSET=<characterset>]
      [&ADAPTOR=<adaptor>] HTTP/1.0

[--Boundary_Value_Of_Your_Choosing
Content-Disposition: form-data; name=”STUDENTLIST”;
      filename=”<studentlist_filename>”
Content-Type: text/plain

<studentlist_file_content>
--Boundary_Value_Of_Your_Choosing--]
```

where:

| Key/Parameter | Input | Description |
|---|---|---|
| option | snapshot | Creates an XML file of all person, group and membership objects. |
| | person_record | Creates an XML file containing basic information for a single student record. |
| | group_record | Creates an XML file containing a list of users with midterm and final grade information for a given course. |
| | group_final_grades | Creates an XML file containing a list of users with final grade information for a given course. |
| | group_midterm_grades | Creates an XML file containing a list of users with midterm grade information for a given course. |
| | | |
| timestamp | UNIX epoch timestamp | Time stamp is in UNIX epoch format (seconds since midnight GMT, Jan 1, 1970). |
| | | |
| auth | A valid MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| adaptor | `webct38` | Specifying the `webct38` adaptor allows you to generate WebCT 3.8 compliant XML. |
| | `IMS` | The `IMS` adaptor is the default adaptor. If you do not specify an adaptor at the command line then the `IMS` adaptor is applied. The `IMS` adaptor complies with version 1.1 and 1.01 of the IMS Enterprise Specification. |
| | | |
| datasource | Any alphanumeric string up to 256 characters. Strings containing spaces can be enclosed in quotation marks. | Sets the `datasource` element within the `properties` element. Defaults to “WebCT” if none is specified. |
| | | |
| target | Any alphanumeric string up to 256 characters. Strings containing spaces can be enclosed in quotation marks. | Sets the `target` element within the `properties` object |
| | | |
| type | Any alphanumeric string up to 256 characters. Strings containing spaces can | Sets the `type` element within the `properties` object |

| Key/Parameter | Input | Description |
|---|---|---|
| | be enclosed in quotation marks. | |
| id | Any person or group object `sourcedid→id`. | When exporting with the person_record, group_record, group_final_grades, or group_midterm_grades options, you use this optional field to specify the person or group object that you want to export |
| | | |
| <studentlist_filename> | Any valid filename | This optional element is provided when using the group_record, group_final_grades, or group_midterm_grades options when you wish to export a subset of data. |
| | | |
| <studentlist_file_content> | Contents of file | This optional element is provided when using the group_record, group_final_grades, or group_midterm_grades options when you wish to export a subset of data.<br>The file must be in plain text, with one IMS id per line. |

**Example**

The following request generates an XML file for a person object with the `person→sourcedid→id` of "Hoover26":

```
GET  /webct/systemIntegrationApi.dowebct?ACTION=export&OPTION=person_record
      &TIMESTAMP=984694373&ID=Hoover26
      &AUTH=1F2E7CA6B6EBCE62D3FC089CA42E80FB HTTP/1.0
```

The server returns the following (not including HTTP headers):

```
<?xml version="1.1" encoding="UTF-8"?>

<enterprise>
  <properties>
    <datasource>WebCT</datasource>
    <datetime>2001-03-15T14:15:18-0800</datetime>
  </properties>
  <person>
    <sourcedid>
       <source>Faber College SIS</source>
       <id>Hoover26</id>
    </sourcedid>
    <userid>rhoover</userid>
    <name>
      <fn>Robert Hoover</fn>
      <n>
        <family>Hoover</family>
        <given>Robert</given>
      </n>
    </name>
    <datasource>WebCT</datasource>
  </person>
</enterprise>
```

## CONFIGURE

Configure sets the IMS ID for group objects and person objects. The syntax for configure is:

```
<GET | POST> /webct/systemIntegrationApi.dowebct?ACTION=configure&OPTION=<option>
      &TIMESTAMP=<unix_epoch_time>&AUTH=<32_byte_mac>
      [&COURSE=<course>][&GLOBALID=<WebCTID>][&ID=<id>][&SOURCE=<source>]

[--Boundary_Value_Of_Your_Choosing
Content-Disposition: form-data; name="FILENAME"; filename="<filename>"
Content-Type: text/plain

<file_content>
--Boundary_Value_Of_Your_Choosing--]
```

where:

| Argument | Input | Description |
|---|---|---|
| option | set_group_ims_info | Sets the IMS id for a group object (course). |
| | import_group_ims_info | Sets the IMS id for multiple group objects (courses) from a file. |
| | set_person_ims_info | Sets the IMS id for a person object (user). |
| | import_person_ims_info | Sets the IMS id for multiple person objects (users) from a file. |
| | get_person_ims_info | Finds the IMS id and IMS source for a single user. |
| | | |
| timestamp | UNIX epoch timestamp | Time stamp in UNIX epoch format (seconds since midnight GMT, Jan 1, 1970). |
| | | |
| auth | A valid MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| course | An existing Course ID | An existing WebCT Course ID is required when using the set_group_ims_info option. |
| | | |
| globalid | An existing WebCT ID | An existing WebCT ID is required when using the `set_person_ims_info` option and the `get_person_ims_info` option. |
| | | |
| id | Any valid IMS id | This argument sets the `group→sourcedid→id` or `person→sourcedid→id` for the set_group_ims_info and `set_person_ims_info` options, respectively. |
| | | |
| source | Any valid IMS source | This argument sets the `group→sourcedid→source` or `person→sourcedid→source` for the `set_group_ims_info` and `set_person_ims_info` options, respectively. |
| <filename> | A valid filename | A filename must be supplied for |

| Argument | Input | Description |
|---|---|---|
| | | `import_group_ims_info` or `import_person_ims_info`. |
| | | |
| <file_content> | Contents of text file | The file must be plain text, in the format: <webct_id>,<ims_id>,<ims_source_new>. |

**Example**

This example sets the `person→sourced→id` to "Pepperidge23" and the `person→sourcedid→source` to "Faber College SIS" for the WebCT ID "mpepperidge":

```
GET  /webct/systemIntegrationApi.dowebct?ACTION=configure
      &OPTION=set_person_ims_info&TIMESTAMP=984701570
      &GLOBALID=mpepperidge&ID=Pepperidge23
      &SOURCE=Faber%20College%20SIS
      &AUTH=95F5858984AB4D1571DC5BE9BD8E21DB HTTP/1.0
```

The Web server returns (not including HTTP headers):

```
<RESPONSE responsetext="optional">SUCCESS</RESPONSE>
Success: IMS info updated for mpepperidge.
```

# CHAPTER 5 STANDARD API

The Standard API gives administrators and developers access to the WebCT databases via command line or Web-based interfaces. The Standard API can be used to integrate external applications with WebCT. For example, it can be used for integrating WebCT with a student information system (SIS).

## FUNCTIONALITY IN THE STANDARD API

The Standard API allows you to manipulate two separate databases within WebCT: the global database and the student database.

The global database contains the central listing of users for all users on the WebCT server. By default, the global database contains the WebCT ID, Password, First Name, Last Name, Courses, and Registered Courses fields. All users must have an entry in this database in order to access a course.

The student database is a term for a collection of databases specific to a course. Every WebCT course has its own student database that contains, by default, the User ID, Password, First Name, and Last Name fields. The information in the student database can be viewed in the designer interface of the *Manage* Students feature in each WebCT course.

Generally, a WebCT ID is linked to a User ID for each course that a user is enrolled in. Users can have different User IDs from their WebCT IDs, as well as different First Name and Last Name data in the student and global databases.

The functionality of the Standard API can be divided into the following basic categories:

| | |
|---|---|
| Adding Users | • Adding a single user to the global database or student database.<br>• Adding multiple users to the global database or student database. |
| Updating Users | • Updating a single user in the global database or student database.<br>• Updating multiple users in the global database or student database.<br>• Updating user types . |
| Deleting Users | • Deleting a single user from the global database or student database.<br>• Deleting multiple users from the global database or student database. |
| Finding WUUIs | • Finding the WebCT Unique Universal Identifier in the global database, either by WebCT ID or IMS ID.<br>**Important**: Since the release of WebCT 3.6, the use of the WUUI for automatic signon and the `find_wuui` operation are deprecated. With WebCT moving towards the use of the IMS specifications, which are becoming standards in the learning community, the IMS ID and IMS source are now preferred over the WUUI. Although the use of the WUUI is deprecated, the functionality remains. |
| Finding Users | • Finding a user in the global database or student database. |
| Changing | • Changing a single user's WebCT ID. |

| | |
|---|---|
| WebCT IDs | • Changing multiple users' WebCT IDs. |
| Exporting *myWebCT* in XML format | • Exporting a user's *myWebCT* in XML format. |

# IMPLEMENTING THE STANDARD API

## COMMAND LINE INTERFACE (WEBCTDB)

The command line interface to the standard API provides a simple interface to the WebCT API. The executable file `webctdb`, is located in the directory `<webct_install_dir>/webct/webct/generic/api/`.

**Important:** WebCT strongly recommends you run the Standard API as the WebCT user. Operating the API as the Root user can prevent students and designers from logging into WebCT.

### SYNTAX

The general syntax for each of the Standard API operations is as follows:

| Operation | Field Names |
|---|---|
| add | <db> <course> <fieldsData_pair_list> <separator> [encrypted] [charset] |
| | |
| delete | <db> <course> <WebCT_ID \| user_id> [charset] |
| | |
| changeid | <db> <course> <fieldsData_pair_list> <separator> [charset] |
| | |
| update | <db> <course> <fieldsData_pair_list> <separator> [encrypted] [charset] |
| | |
| find | <db> <course> <WebCT_ID \| user_id> <separator> [user_type] [charset] |
| | |
| find_wuui | <db> <course> <WebCT_ID> [charset] |
| | |
| find_ims_id_wuui | <db> <course> <IMS_ID> [charset] |
| | |
| fileadd | <db> <course> <filename> <separator> [encrypted] [charset] |
| | |
| fileupdate | <db> <course> <filename> <separator> [encrypted] [charset] |
| | |
| filedelete | <db> <course> <filename> [charset] |
| | |
| filechangeid | <db> <course> <filename> <separator> [charset] |
| | |
| homearea_xml | <db> <course> <WebCT ID> <separator pair> <server base address> [charset] |

**Field Name:** `db`
**Value:** `global or student`
**Example:** `global`
**Description:** This is the name of the database, either global database or student database.

**Field Name:** `course`
**Value:** `Course ID`
**Example:** `cs100`
**Description:** - Required for student database operations.
- For global database operations, enter the placeholder value xxxx.

**Field Name:** `fieldsData_pair_list`
**Value:** A double quote-enclosed list of field-data pairs in the form: field_name=data_value.
**Example:** `WebCT ID=student1`
**Description:**
- The field names must exist in the WebCT global database or student database. The *separator* must be inserted between each of the field-data pairs.
- For the global database, the optional fields Courses and Registered Courses are available for adding and/or modifying courses and registered courses to which a global user belongs. The values for these fields can be a colon-separated list of course IDs for Courses or course names for Registered Courses. For example, `Courses=cs100:psyc100:math100`. If you also specify a user type with the course, this is separated from the course ID by a semicolon, for example, `Courses=cs100;D:psyc100;TA`. **Note**: The default user type is (S)tudent.
- A user can be added as a primary designer or as a secondary designer. The first WebCT ID added to the course as a designer becomes the primary designer; every subsequent designer becomes a secondary designer.

**Field Name:** fieldsData_pair_list (cont.)

**Description:**

- **Note**: The following are reserved words in the fieldsData_pair_list:
- Login ID (this is old terminology and is supported for backward compatibility only. It has the same meaning as User ID).
- User ID (the User ID of a student in a course)
- Password (the password of the global user or the student)
- Global ID (this is old terminology and is supported for backward compatibility only. It has the same meaning as WebCT ID).
- WebCT ID (WebCT ID of a global user)
- First Name (first name of the global user or the student. It is one of the reserved columns in both the global and student databases)
- Last Name (last name of the global user or the student. It is one of the reserved columns in both the global and student databases)
- Courses (the list of WebCT courses for a global user). If you populate this field through the API, the course must already exist on the WebCT server.
- Registered Courses (the list of courses maintained by the registrar for a global user. These courses may or may not have a WebCT course.)
- Thumbprint (internal data and cannot be modified)
- LockPID (internal data and cannot be modified)
- #User Type (internal data. This can be modified through the API)
- #E-mail (internal data and cannot be modified)
- #Password Question (internal data and cannot be modified)
- #Password Answer (internal data and cannot be modified)

**Note**: The reserved words are case sensitive.

**Field Name:** `separator`

**Value:** Any alphanumeric string representing the separator between data pairs in the fieldsData_pair_list.

**Example:** "," (comma)

**Description:** Delimiter used to separate data items. You must declare what value you will be using as a delimiter for the operations add, delete, changeid, update, and find.

**Note**: For the global database, the colon and semi-colon are not allowed as separators.

**Field Name:** `user_type`

**Value:** `user_type`

**Example:** `user_type`

**Description:** Only used with the `find` operation on the global database; return value of user_type is one of three users types: S for student, D for designer, and TA for teaching assistant.

**Field Name:** `encrypted`
**Value:** `encrypted`
**Example:** `encrypted`
**Description:**
- Only used with the add, update, fileadd and fileupdate operations.
- The password must be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- Add to the end of the command line to indicate that the passwords are passing in encrypted form.

**Field Name:** `charset`
**Value:** A valid character set. See the *Appendix*.

**Example:** `"--charset=iso-8859-1"`
**Description:**
- If specified, charset will override the character set as defined on the administrator settings page as the file charset.
- **Note**: The default charset type is UTF-8.

## FUNCTIONS

### ADDING USERS

Users can be added to the global database or student databases. However, in general, you should add users to the global database and use the Courses field to add them to each course. This method is simpler and automatically links the WebCT ID to each User ID.

**Warning:** WebCT strongly recommends you re-add previously deleted users through the administrator interface. Re-adding users through the Standard API can prevent preserved records from merging with the user. See the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™, Chapter 5, User Management.*

#### ADDING A SINGLE USER TO THE GLOBAL DATABASE

| **Operation = add** |
| --- |
| • The fieldsData_pair_list must include both the WebCT ID and Password fields.<br>• You can specify the user type: S for student, D for designer, TA for teaching assistant. If you don't specify a user type, the user type defaults to (S)tudent. If the user type is specified as (D)esigner and there is no existing designer, the user is added as the primary designer. If there is an existing designer, the user is added as secondary designer. |

**Example**
Add a user named Justin Case to the global database as a designer for cs100; a teaching assistant for cs200; and as a student in cs810:

Enter the command:

UNIX
```
./webctdb add global xxxx "WebCT ID=jcase,Password=1234,First
Name=Justin,Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S" ","
```

Windows
```
webctdb add global xxxx "WebCT ID=jcase,Password=1234,First
Name=Justin,Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S" ","
```

### ADDING A SINGLE USER TO THE STUDENT DATABASE

**Operation = add**

- The fieldsData_pair_list must include both the User ID and Password fields.

**Example**

Add a user named Bailey Wick to the student database for course cs100:

Enter the command:

UNIX
```
./webctdb add student cs100 "User ID=bwick,Password=1234,
First Name=Bailey,Last Name=Wick" ","
```

Windows
```
webctdb add student cs100 "User ID=bwick,Password=1234,
First Name=Bailey,Last Name=Wick" ","
```

### ADDING MULTIPLE USERS TO THE GLOBAL DATABASE

**Operation = fileadd**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is only the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. There must be no spaces between the data and the separators.
- If the user exists in the database, `fileadd` will send an error message to STDOUT (prints it to the screen). The user record will not be changed; the process will skip to the next record in the file.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.

**Example**

Add users to the global database from a text file named `users.txt`.

**SAMPLE USERS.TXT FILE:**

```
WebCT ID,Password,Last Name,First Name
jsmith,9876,Smith,John
jbrown,2345,Brown,Jane
bfawlty,8765,Fawlty,Basil
```

```
arigsby,5432,Rigsby,Arthur
```

Enter the command:

UNIX `./webctdb fileadd global xxxx users.txt ","`

Windows `webctdb fileadd global xxxx users.txt ","`

### ADDING MULTIPLE USERS TO THE STUDENT DATABASE

**Operation = fileadd**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is only the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. There must be no spaces between the data and the separators.
- If the user exists in the database, `fileadd` will send an error message to STDOUT. The user record will not be changed in the database; the process will skip to the next record in the file.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added users may not be able to access WebCT.

## Example

Add students whose records are stored in the file `class.txt` to the course cs100.

#### SAMPLE CLASS.TXT FILE

```
User ID,Password,Last Name,First Name
jsmith,9876,Smith,John
jbrown,2345,Brown,Jane
bfawlty,8765,Fawlty,Basil
arigsby,5432,Rigsby,Arthur
```

Enter the command:

UNIX `./webctdb fileadd student cs100 class.txt ","`

Windows `webctdb fileadd student cs100 class.txt ","`

## UPDATING USERS

### UPDATING A SINGLE USER IN THE GLOBAL DATABASE

**Operation = update**

- The fieldsData_pair_list must include the WebCT ID.
- Empty fields are not changed.
- If the field value is "_DELETE_", the value will be set to null
- When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you supply and unlinked from any pre-existing courses that you do not supply.
- You can update and change a user type by specifying a different user type. Example, you can change a designer (D) into a student (S).
- `update` cannot be used to modify quiz scores.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or updated users may not be able to access WebCT.

**Example**

For the student Justin Case, password 1234, with the following courses: cs100(D) cs200(TA) cs810(S), update the password in the global database and update the courses so that only cs100 remains.

Enter the command:

UNIX
```
./webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100" ","
```

Windows
```
webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100" ","
```

**Example 2**

Using the previous example, update Justin Case so he is now a student in course cs100.

```
Enter the command:
```

UNIX
```
./webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100;S" ","
```

Windows
```
webctdb update global xxxx "WebCT ID=jcase,Password=abcd,
Courses=cs100;S" ","
```

### UPDATING A SINGLE USER IN THE STUDENT DATABASE

**Operation = update**

- The fieldsData_pair_list must include the User ID field.
- Empty fields are not changed.
- If the field value is "_DELETE_", the value will be set to null.
- When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the

**Operation = update**

courses that you supply and unlinked from any pre-existing courses that you do not supply.

- You can update a user type by specifying a different one.
- `update` cannot be used to modify quiz scores.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the updated users may not be able to access WebCT.

### Example

To update the student Bailie Wicke, first name, last name, and password of password "abcd".

Enter the command:

UNIX
```
./webctdb update student cs100 "User ID=bwick,Password=abcd,
First Name=Bailie,Last Name=Wicke" ","
```

Windows
```
webctdb update student cs100 "User ID=bwick,Password=abcd,
First Name=Bailie,Last Name=Wicke" ","
```

## UPDATING MULTIPLE USERS IN THE GLOBAL DATABASE

**Operation = fileupdate**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. There must be no spaces between the data and the separators.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- `fileupdate` will add a user if they do not exist in the database.
- Empty fields will not be changed.
- For `fileupdate`, if the value field value is "_DELETE_", the value will be set to null.
- When updating the Courses and Registered Courses field, the Standard API always overwrites the field. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you supply and unlinked from any pre-existing courses that you do not supply.

### Example

Change the names of a group of users whose updates are contained in the file `updates.txt`.

**SAMPLE UPDATES.TXT FILE:**

```
WebCT ID,Last Name,First Name
jsmith,Smith,Jerry
jbrown,Brown,Janet
bfawlty,Fawlty,Brian
arigsby,Rigsby,Alan
```

Enter the command:

UNIX `./webctdb fileupdate global xxxx updates.txt ","`

Windows `webctdb fileupdate global xxxx updates.txt ","`

### UPDATING MULTIPLE USERS IN THE STUDENT DATABASE

**Operation = fileupdate**

- filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is the name of the file. A file extension, such as `.txt`, is recommended.
- The file must be in plain text. The first line of the file must be the field names separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. There must be no spaces between the data and the separators.
- An optional `encrypted` argument can be added at the end of the command line to indicate that the passwords are passing in an encrypted form. The passwords should be encrypted using the standard UNIX DES encryption method or the newly added or modified users may not be able to access WebCT.
- `fileupdate` will add a student or user if they do not already exist in the database.
- Empty fields will not be changed.
- For `fileupdate`, if the field value is "_DELETE_", the value will be set to null
- `fileupdate` overwrites the data fields being changed; it does not append.

## Example

Change the names of students in the course cs100 using updates contained in the file `updates.txt`.

**SAMPLE UPDATES.TXT FILE:**

```
User ID,Last Name,First Name
jsmith,Smith,Jerry
jbrown,Brown,Janet
bfawlty,Fawlty,Brian
arigsby,Rigsby,Alan
```

Enter the command:

UNIX `./webctdb fileupdate student cs100 updates.txt ","`

Windows `webctdb fileupdate student cs100 updates.txt ","`

## DELETING USERS

### DELETING A SINGLE USER FROM THE GLOBAL DATABASE

| Operation = delete |
| --- |
| • global_id is the ID of the user to be deleted from the global database.<br><br>**Note**: Depending on the *User Data* setting in the administrator interface, the student's data can also be deleted from the student database. |

**Example**
Delete the global database record for the user whose WebCT ID is jcase. **Note**: The student will be denied access to all the courses listed in their global database record. Depending on the *User Data* setting in the administrator interface, the student's data can also be deleted from the student database.

Enter the command:

UNIX `./webctdb delete global xxxx jcase`

Windows `webctdb delete global xxxx jcase`

### DELETING A SINGLE USER FROM THE STUDENT DATABASE

| Operation = delete |
| --- |
| • user_id is the ID of the student to be deleted from the student database. |

**Example**
Delete the record for the student in the cs100 course whose User ID is bwick.

Enter the command:

UNIX `./webctdb delete student cs100 bwick`

Windows `webctdb delete student cs100 bwick`

### DELETING MULTIPLE USERS FROM THE GLOBAL DATABASE

| Operation = filedelete |
| --- |
| • filename is the either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the file. A file extension, such as `.txt`, is recommended.<br>• If the user does not exist in the database, `filedelete` will send an error message to STDOUT. The process will skip to the next record in the file. |

### Example

Delete users from the global database using a text file `deleteusers.txt`.

**SAMPLE DELETEUSERS.TXT FILE:**

```
jsmith
jbrown
bfawlty
arigsby
```

Enter the command:

UNIX `./webctdb filedelete global xxxx deleteusers.txt ","`

Windows `webctdb filedelete global xxxx deleteusers.txt ","`

#### DELETING MULTIPLE USERS FROM THE STUDENT DATABASE

**Operation = filedelete**

- filename is the either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the file. A file extension, such as `.txt`, is recommended.
- If the user does not exist in the database, `filedelete` will send an error message to STDOUT The process will skip to the next record in the file.

### Example

Delete students whose records are stored in the file `delete.txt` from the course cs100.

**SAMPLE DELETE.TXT FILE:**

```
jsmith
jbrown
bfawlty
arigsby
```

Enter the command:

UNIX `./webctdb filedelete student cs100 delete.txt`

Windows `webctdb filedelete student cs100 delete.txt`

## FINDING WUUIS

**Important**: Since the release of WebCT 3.6, the use of the WUUI for automatic signon and the `find_wuui` operation are deprecated. With WebCT moving towards the use of the IMS specifications, which are becoming standards in the learning community, the IMS ID and IMS source are now preferred over the WUUI. Although the use of the WUUI is deprecated, the function will still be supported for 4.0.

### FINDING WUUIS USING IMS IDS

**Operation = find_ims_id_wuui**

- The WUUI (WebCT Unique Universal Identifier) is a 32-character alphanumeric string that identifies a global user in WebCT.
- The `find_ims_id_wuui` operation is for the global database only.
- This operation is similar to the `find_wuui` operation, except that your campus portal passes the user's IMS ID, not their WebCT ID. WebCT returns the user's WUUI.
- This operation can be used only when WebCT's global database has been populated using the IMS Enterprise API, because only in those cases would an IMS ID be present for each user in the WebCT global database.
- IMS ID is the IMS ID of the user in the global database.
- The WUUI is sent to STDOUT.

**Example**
Find the WUUI for the user whose IMS ID is jcase.

Enter the command:

UNIX
```
./webctdb find_ims_id_wuui global xxxx jcase
```

Windows
```
webctdb find_ims_id_wuui global xxxx jcase
```

The server returns:

```
Success: WUUI=abcdefghijklmnopqrstuvwxyz123456
```

## FINDING USERS

### FINDING A USER IN THE GLOBAL DATABASE

**Operation = find**

- WebCT ID is the WebCT ID of the user in the global database.
- Separator is the separator of the output data, which is sent to STDOUT in the same format as the fieldsData_pair_list.
- If the field name user_type is specified in a global database query, the user type (S,D,TA) will be included in the result.

**Example**
Find the global database record, including user type, for the user with the WebCT ID jcase.

Enter the command:

UNIX
```
./webctdb find global xxxx jcase "," user_type
```

Windows
```
webctdb find global xxxx jcase "," user_type
```

If the command is successfully executed:

```
Success: WebCT ID=jcase,First Name=Justin,
      Last Name=Case,Courses=cs100;D:cs200;TA:cs810;S
```

### FINDING A USER IN THE STUDENT DATABASE

| Operation = find |
| --- |
| • user_id is the User ID of the student in the student database.<br>• Separator is the separator of the output data, which is sent to STDOUT in the same format as the fieldsData_pair_list. |

**Example**
Find the student in the cs100 course whose User ID is bwick.

Enter the command:

UNIX `./webctdb find student cs100 bwick ","`

Windows `webctdb find student cs100 bwick ","`

If the command is successfully executed:

```
Success:First Name=Bailie,Last Name=Wicke,User ID=bwick
```

## CHANGING WEBCT IDS

### CHANGING A SINGLE USER'S WEBCT ID

| Operation = changeid |
| --- |
| • `changeid` can only be used on the global database.<br>• old_id is the WebCT ID to be changed.<br>• new_id is the new WebCT ID. |

**Example**
Change Justin Case's WebCT ID from jcase to jicase.

Enter the command:

UNIX `./webctdb changeid global xxxx "Old ID=jcase,New ID=jicase" ","`

Windows `webctdb changeid global xxxx "Old ID=jcase,New ID=jicase" ","`

### CHANGING MULTIPLE USERS' WEBCT IDS

| Operation = filechangeid |
| --- |
| • filename is either a full absolute path or a relative path from the current directory to the file. For example, if the file is located in the current directory, then filename is simply the name of the |

**Operation = filechangeid**

file. A file extension, such as `.txt`, is recommended.

- The first line of the data file should be the field names Old ID and New ID, separated by the separator string. The rest of the file contains the data, one record per line. Data should be in the same order as the field names, separated by the value of the separator. **Note**: The field name Old ID does not exist in the databases.
- If the user does not already exist in the database, `filechangeid` will send an error message to STDOUT. The process will skip to the next record in the file.

**Example**

Change the WebCT IDs of a group of users contained in a file `changeusers.txt`.

**SAMPLE CHANGEUSERS.TXT FILE:**

```
Old ID,New ID
jsmith,jtsmith
jbrown,jkbrown
bfawlty,befawlty
arigsby,aurigsby
```

Enter the command:

UNIX `./webctdb filechangeid global xxxx changeusers.txt ","`

Windows `webctdb filechangeid global xxxx changeusers.txt ","`

## EXPORTING MYWEBCT IN XML FORMAT

This Standard API command exports a user's *myWebCT* in XML format, which allows *myWebCT* information to be modified and redisplayed in a desired format. For example, the information can be integrated with a campus portal.

This command can be used in conjunction with automatic signon, allowing for a single point of authentication, see *Chapter 2: Automatic Signon From Other Systems*.

**Operation = homearea_xml**

- `homearea_xml` can only be used on the global database.
- WebCT ID is the WebCT ID of the user whose *myWebCT* you want to export in XML format.
- Separator is the separator of the output data, which is sent to STDOUT in the same format as the fieldsData_pair_list.
- server base address is the address of the WebCT server.

The returned XML conforms to the DTD located in `<install_dir>/webct/webct/generic/api/xml/mywebct.dtd`.

The XML can be parsed to extract the required elements. Link elements that require authentication by WebCT contain the attribute "secure" with a value of TRUE.

**Example**
Export *myWebCT* in XML format for the user whose WebCT ID is `jsmith` and whose server base address is `http://webctserver:port`.

Enter the command:

UNIX
```
./webctdb homearea_xml global xxxx jsmith http://webctserver:port
```

Windows
```
webctdb homearea_xml global xxxx jsmith http://webctserver:port
```

## WEB-BASED INTERFACE (SERVE_WEBCTDB)

The Web-based Standard API allows data in the WebCT global database and student databases to be queried and manipulated by remote servers. For example, the Web-based interface can be used to make changes to global database records based on registration changes driven by events on another system. It can also be used to create a custom administrator interface.

Log files are created during each import, export and configuration event. E-mail alerts are also generated for import events that are not processed correctly. See Section 2, *Logging, page* 75 for details.

**Important:**

- Some system administrators have created scripts that automatically import data files from outside systems to WebCT and export data file from WebCT to outside systems on a regular basis. By default, these scripts rely on the character set for imported and exported administrator files, which is set in the WebCT administrator interface. Both imported and exported files will be converted to this character set. You should change all automatic scripts to explicitly specify the character set of the files being imported to match the actual character set of the source data. This will ensure that background scripts import data smoothly, no matter the changes made to the character set for imported and exported administrator files. See the *International Support* chapter of this guide for more information.
- See the chapter *International Support, Importing data into WebCT* section for directions to specify the file character set for the administrator interface.
- See *Section 2: Campus Edition Institution License, Chapter 4 IMS Enterprise API,* Import, Example 2 page 75 in this guide for details about how to set the CHARSET parameter.

Implementing the Web-based interface involves two steps.

1. Setting the API shared secret value
2. Developing a program to generate an HTTP request

Step 1 can be accomplished by a WebCT administrator who has basic knowledge of the WebCT file system. Step 2 requires an experienced Web developer.

### SETTING THE API SHARED SECRET VALUE

The shared secret value is a key component of allowing external servers to automatically sign on users to WebCT. The shared secret value is used to create a Message Authentication Code (MAC) from the submitted data. When WebCT receives a request, it decodes the shared secret value from MAC using the submitted data.

If the decoded shared secret value is the same as the one stored locally, the request is considered authentic and is processed. Because the shared secret value has such a critical role, choose it carefully. You can set the shared secret value by performing the following steps:

1. Using a text editor, open the file `<webct_install_directory>/webct/webct/generic/api/api_secret`
2. Change the first line of the file to your desired secret. For security reasons, the default value *secret* does not work. The shared secret value
   - is case-sensitive
   - cannot exceed 256 characters.
   - cannot contain tab, or other control characters.
   - should not contain end-of-line characters. **Note**: By default, the UNIX text editor vi and pico automatically add end-of-line characters. Check the file size to ensure that the number of characters equals the number of bytes.
3. Save the file.

**Tips for Shared Secrets**

- Make your shared secret value difficult to guess by making it lengthy and by including a combination of numbers and upper and lower case characters.
- Change your shared secret value at regular intervals.
- On remote systems, place shared secret values in secure directories.

## Developing a Program to Generate an HTTP Request

Developing a program to generate an HTTP request is the most substantive part of implementing the Web-based standard API. The program must:

- generate a Message Authentication Code (MAC)
- assemble a properly formatted HTTP request
- process any data being returned

### Creating Message Authentication Codes

Because the Web interfaces to the Standard API and automatic signon reside in public directories, Message Authentication Codes (MACs) are required to ensure that only messages from trusted servers are processed.

WebCT provides three options to assist you in creating MACs:

1. A C function that you can integrate and compile into your C program.
2. An executable file to which you make a system call from your program.
3. Instructions for generating a MAC using a language of your choice.

#### Option 1: Using the get_authentication C Function

The get_authentication function generates a MAC from an array of data and a shared secret value.

The source code necessary to use the C function is located in `<webct_install_directory>/webct/webct/generic/api/security/`

A test program, which contains a Makefile for UNIX based systems, is also provided.

The file `api_security.c` contains the `get_authentication` function.

| get_authentication | Generates a MAC from an array of data and a shared secret value |
|---|---|
| Syntax | char* get_authentication (int i, char* data[], char* secret, char* encrypted_data) |
| Returns | 32-byte alphanumeric MAC |
| **Parameter** | **Description** |
| `i` | The number of elements in the array data[]. |
| `data` | Array of all values to be used in generating the MAC. The data should not be URL encoded. |
| `secret` | The shared secret value. |
| `encrypted_data` | The memory location of the MAC. It must be at least 32 bytes long. |

### OPTION 2: USING THE MESSAGE AUTHENTICATION CODE GENERATOR (GET_MD5 EXECUTABLE)

The Message Authentication Code (MAC) generator generates a MAC from a shared secret value and a string consisting of the IMS ID, a timestamp, and a destination URL.

If you are not working in C or do not want to create a function to create the MAC, use the Message Authentication Code generator (an executable called get_md5) You can make a system call to get_md5 from your program and have the authentication string returned. The get_md5 executable has no dependencies on WebCT and can be copied to other servers as required. If you need a get_md5 executable for an operating system other than the one your WebCT server is running on, you can download several pre-compiled binaries for other operating systems on `www.webct.com`.

| get_md5 | Generates a MAC from a shared secret value and a line of data |
|---|---|
| Syntax | get_md5 <shared_secret_filename> <data_to_encrypt> |
| Returns | 32-byte alphanumeric MAC |
| **Attribute** | **Description** |
| `shared_secret_filename` | The filename where the shared secret value is stored. |

| | |
|---|---|
| `string_to_encrypt` | The string to be encrypted. The string should not be URL encoded. |

### OPTION 3: CREATE A MAC USING A LANGUAGE OF YOUR OWN CHOICE

If you want to create MACs within your code, (e.g. you are writing your code in Java and don't want to make a system call), you can create a MAC with the following procedure:

1. Calculate the total of the ASCII values of all the characters in the data.
2. Convert the total of the ASCII values into a string.
3. Append the shared secret value.
4. Encrypt the string into a 16-byte string using the MD5 algorithm.
5. Convert the 16-byte string into a 32-byte alphanumeric string to make it URL-friendly.

## HOW THE SERVE_WEBCTDB MAC IS GENERATED

The MAC is generated by using key/value pairs in the request. The serve_webctdb API uses the value from all key/value pairs except the ones listed below.

- AUTH
- ENCRYPTED
- USER_TYPE
- USER TYPE
- CHARSET

## ASSEMBLING THE HTTP REQUEST

There are several options for assembling an HTTP request to the Web-based standard API. The option you choose will be based on your programming language of choice and how you want to communicate with the Web server. You can issue API commands in several ways, including:

- Socket programming directly with the Web server
- Using a library which simulates a user agent
- Assembling a GET request and refreshing a browser window with the query string.

In Perl, you have the option of communicating directly with the Web server using the IO::Socket module included with most basic distributions, or installing and using a module such as LWP which simulates a user agent (e.g. a Web browser). Similar modules are available for most popular languages such as C or Java.

If you wish to refresh a user's browser window with a query string, you can do so using the "Location" HTTP header, HTML meta tags, or using JavaScript's location.replace method.

## SYNTAX

The general syntax for a Web-based request to the Standard API is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=<operation>&DB=<db>
      &COURSE=<course_id | placeholder>&AUTH=<32_byte_mac>
      [&User%20ID=<user_id> | &WebCT%20ID=<webct_id>][&IMS%20ID=<ims_id>]
```

```
[&USER_TYPE=<1_or_0>][&ENCRYPTED=<1_or_0>][&field1=<field1>]
[&fieldn=<fieldn>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | add | Adds a user to the global or student database.<br>If the user already exists, an error is returned. |
| | | |
| | update | Updates an existing user in the global or student database.<br>If the user does not exist, this operation returns an error. |
| | | |
| | delete | Deletes a single user from the global or student database. |
| | | |
| | find_wuui | Finds the WUUI for a user using their WebCT ID as the key. |
| | find_ims_id_wuui | Finds the WUUI for a user using their IMS id (`person→sourcedid→id`) as the key. |
| | | |
| | find | Finds the user record based on the User ID (if searching the student database) or WebCT ID (if searching the global database). |
| | | |
| | changeid | Changes a WebCT ID. |
| | | |
| | homearea_xml | Exports a user's *myWebCT* in XML format. |

**Notes**:

- The Standard API can accept GET or POST requests. POST requests can put their key/value pairs in the query string or in the body of the message in the appropriate format (see the W3C HTML 4.01 Specification at `http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4`)
- Requests must be URL encoded (e.g. spaces should be replaced with %20)
- Key/value pairs can appear in any order
- Syntax examples represent HTTP requests directly to the Web server. If you are using a programming module to create your requests (such as LWP in Perl), many details of the request can be transparent to you.

## FUNCTIONS

## ADDING USERS

### ADDING A USER TO THE GLOBAL DATABASE

Add operations have the following syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=add&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      &Password=<password>[&field1=<field1>][&fieldn=<fieldn>]
      [&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | WebCT ID | The WebCT ID of the user being added. WebCT IDs can contain alphanumeric strings, underscores, and periods. |
| | | |
| Password | Password | The password to be used for the user being added. Passwords can consist of any alphanumeric string. The API does not enforce minimum password lengths. |
| | | |
| Field1<br>…<br>Fieldn<br>(optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| ENCRYPTED<br>(optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting. |
| | | |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Note:** The Courses field uses a colon as a delimiter between courses and a semicolon as a delimiter between user types. Thus the string “HKIN100;D:HKIN200;TA:HKIN300;S” indicates that a user is to be added to HKIN100 as a designer, HKIN200 as a teaching assistant and HKIN300 as a student. If no user type is specified, WebCT will default to adding the user as a student. Similarly, the Registered Courses field is colon

delimited. For more information on the Courses and Registered courses field, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.

### Example 1

Add a user to the global database, and enroll them in the course ENGL100 as a designer, ENGL560 as a student, and ENGL477 as a teaching assistant.

```
GET  /webct/public/serve_webctdb?OPERATION=add&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&WEBCT%20ID=pinto
      &PASSWORD=an1mal&FIRST%20NAME=Larry&LAST%20NAME=Kroger&
      COURSES=ENGL100;D:ENGL560:ENGL477;TA HTTP/1.0
```

### Example 2

Use UTF-8 characters to add a user to the global database, with WebCT ID "jpena", password "1234", first name "Juán", and last name "Peña".

```
GET  /webct/public/serve_webctdb?OPERATION=add&DB=global&COURSE=xxxx
      &WebCT+ID=jpena&Password=1234&First+Name=Ju%C3%A1n&Last+Name=Pe%C3%B1a
      &AUTH=E9895DF02FDEE859C8BDF7E0EA0E696F
```

**Note:** Accented characters are converted to UTF-8 bytes then are URL-escaped. The accented character "á" is represented using the byte sequence "C3 A1" in UTF-8, which is then URL-escaped in the string "%C3%A1".

## ADDING A USER TO THE STUDENT DATABASE

Students can be added to the student database using the following syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=add&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      &Password=<password>[&field1=<field1>][&fieldn=<fieldn>]
      [&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course to which the user will be added. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the user being added. |
| | | |
| Password | Password | The password to be used for the user being added. The API does not enforce minimum password lengths. |

| Key | Value | Description |
|---|---|---|
| Field1<br>…<br>Fieldn<br>(optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| ENCRYPTED<br>(optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting |
| | | |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear |

**Example**

Add a student to the student database of the course ENGL588. In addition, add data to a pre-existing column "StudentNumber" (This is a custom column created by the designer). Because this user is being added to the student database only, they are considered an "orphan user" until a WebCT ID is associated with this User ID:

```
GET  /webct/public/serve_webctdb?OPERATION=add&DB=student
      &COURSE=ENGL588&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=flounder&Password=an1mal&First%20Name=Kent
      &Last%20Name=Dorfman&StudentNumber=123456789 HTTP/1.0
```

## UPDATING USERS

### UPDATING A USER IN THE GLOBAL DATABASE

Updating users in the global database is very similar to adding users. The syntax is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=update&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      [&FIELD1=<field1>][&FIELDN=<fieldn>][&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | Existing WebCT ID | The WebCT ID of the user being added.<br>WebCT IDs can contain alphanumeric strings, underscores, and periods. |
| | | |
| Password | Password | The password to be used for the user being updated. The |

| Key | Value | |
|---|---|---|
| (optional) | | API does not enforce minimum password lengths. |
| Field1<br>…<br>Fieldn<br>(optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| | _DELETE_ | The "_DELETE_" keyword deletes the data from the field specified in the key and sets it to undefined. |
| | | |
| ENCRYPTED<br>(optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Notes:**

- The *User Data* setting in the WebCT administrator interface affects how updating the Courses column will modify the student database when unlinking WebCT IDs from User IDs. If the User Data setting is selected, user data is left in the student database.
- When updating, the Standard API always overwrites the Courses and Registered Course fields. If you supply a Courses field in your update, the user's WebCT ID will be linked to the courses that you specify and unlinked from any pre-existing courses that you do not specify.
- The Courses field uses a colon as a delimiter between courses and a semicolon as a delimiter between user types. Thus the string `HKIN100;D:HKIN200;TA:HKIN300;S` indicates that a user is to be added to `HKIN100` as a designer, `HKIN200` as a teaching assistant, and `HKIN300` as a student. If no user type is specified, WebCT will default to adding the user as a student. Similarly, the Registered Courses field is colon delimited. For more information on the Courses and Registered courses field, see the appropriate version of the *System Administrator's Guide: WebCT Campus Edition™*.

## Example

A user is currently enrolled in three courses: `ENGL101` as a designer, `ENGL560` as a student, and `ENGL477` as a teaching assistant. This example unlinks the WebCT ID from the User ID for `ENGL 560` and `ENGL 477`, and adds the WebCT ID to the course `ENGL101` as designer.

```
GET  /webct/public/serve_webctdb?OPERATION=update&DB=global&COURSE=xxxx&AUTH=EB1A
      09F0BB299C23E99A5978587F49C1&WebCT%20ID=pinto&Courses=ENGL101;D:ENGL101;D
      HTTP/1.0
```

The user is unlinked from the two courses because API updates always overwrite fields.

### UPDATING A USER IN THE STUDENT DATABASE

Updating students in the student database is very similar to adding students. The syntax is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=update&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      [&field1=<field1>][&fieldn=<fieldn>][&ENCRYPTED=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course in which the user's data is updated. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the user being added. |
| | | |
| Password (optional) | Password | The password to be used for the user being added. The API does not enforce minimum password lengths. |
| | | |
| Field1 … Fieldn (optional) | Data associated with the column specified as a key. | Any of the default columns within the global database (First Name, Last Name, Courses, Registered Courses) can be modified using the syntax ColumnName=Value. Administrator-created columns can also be modified using this method. |
| | | |
| | _DELETE_ | The "_DELETE_" keyword deletes the data from the field specified in the key and sets it to undefined. |
| ENCRYPTED (optional) | 1 | Enables pre-encrypted password support. With the encrypted argument set to 1, you should pass passwords encrypted with the standard UNIX DES method when using this setting. |
| | 0 (default) | Disables pre-encrypted password support (default). In this mode, passwords should be submitted as clear-text. |

**Example**

In the following example, a student record is updated with information for the instructor-added numeric columns *Student Participation* and *Bonus* in the course MATH100.

```
GET /webct/public/serve_webctdb?OPERATION=update&DB=student&
      COURSE=MATH100&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=otter&Student%20Participation=100&Bonus=34 HTTP/1.0
```

## DELETING USERS

### DELETING A USER FROM THE GLOBAL DATABASE

The syntax for deleting a user from the global database is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=delete&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | WebCT ID | The WebCT ID of the user being deleted. |

**Note:** The *User Data* setting in the WebCT administrator interface affects whether user data is left in a course when a user record is deleted from the global database. If the *User Data* setting is selected, user data is left in the student database.

#### Example

In this example, the user record for the user with the WebCT ID `neidermeyer` is deleted from the global database:

```
GET  /webct/public/serve_webctdb?OPERATION=delete&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&WebCT%20ID=neidermeyer
      HTTP/1.0
```

### DELETING A USER FROM THE STUDENT DATABASE

The syntax for deleting a student from the student database is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=delete&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id>
      HTTP/1.0
```

where:

| Key | Value | |
|---|---|---|
| COURSE | WebCT Course ID | The WebCT course from which the user will be deleted. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the user being deleted. |

### *Example*

In this example, the student with the User ID `stork` is deleted from the course PSYCH204-23:

```
GET  /webct/public/serve_webctdb?OPERATION=delete&DB=student
      &COURSE=PSYCH204-23&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      &User%20ID=stork HTTP/1.0
```

## FINDING WUUIS

**Important**: Since the release of WebCT 3.6, the use of the WUUI for automatic signon and the find_wuui operation are deprecated. With WebCT moving towards the use of the IMS specifications, which are becoming standards in the learning community, the IMS ID and IMS source are now preferred over the WUUI. Although the use of the WUUI is deprecated, the functionality remains.

The syntax for finding WUUIs is as follows:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=<operation>
      &DB=global&field1=<field1>&COURSE=<placeholder>
      &AUTH=<32_byte_mac> HTTP/1.0
```

| Key | Value | |
|---|---|---|
| OPERATION | find_wuui | Finds a user's WUUI from a WebCT ID. |
| | find_ims_id_wuui | Finds a user's WUUI from an IMS ID. |
| | | |
| AUTH | 32-byte MAC | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| `field1` | WebCT ID | Use if the operation is find_wuui. |
| | IMS ID | Use if the operation is find_ims_id_wuui. |
| | | |
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |

### EXAMPLES

#### FINDING A USER'S WUUI FROM A WEBCT ID

Find the WUUI for the WebCT ID `jdoe`.

```
GET  /webct/public/serve_webctdb?OPERATION=find_wuui&DB=global
      &WebCT%20ID=jdoe&COURSE=xxxx&AUTH=EB1A09F0BB299C23E99A5978587F49C1
      HTTP/1.0
```

The Web server returns the following (not including HTTP headers):

```
Success: #WUUI = 6321BB2537BE7F1E26375D4E1687EE1F
```

### FINDING A USER'S WUUI FROM AN IMS ID

Find the WUUI for the IMS id (person→sourcedid→id) 123456789:

```
GET  /webct/public/serve_webctdb?OPERATION=find_ims_id_wuui&DB=global
      &IMS%20ID=123456789&COURSE=xxxx&
      AUTH=EB1A09F0BB299C23E99A5978587F49C1 HTTP/1.0
```

The Web server returns the following (not including HTTP headers):

```
Success: #WUUI = 6321BB2537BE7F1E26375D4E1687EE1F
```

## FINDING USERS

### FINDING A USER IN THE GLOBAL DATABASE

To find a user's global database record, the following syntax is used:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=find&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&WebCT%20ID=<webct_id>
      [USER_TYPE=<1_or_0>]HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | find | Finds the user record for a given WebCT ID. |
| | | |
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| WebCT ID | WebCT_ID | The WebCT ID of the record you want to display. |
| | | |
| USER_TYPE (optional) | 1 | With the User Type option enabled, the global database record generated includes user type information that indicates whether a user is a designer, student, or teaching assistant for the course. |
| | 0 (default) | No user type information is generated. |

**Example**
In this example, the complete record including user type information is returned for the user with the WebCT ID “pinto”, who is enrolled in three courses.

```
GET  /webct/public/serve_webctdb?OPERATION=find&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&WebCT%20ID=pinto
      &USER_TYPE=1 HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success: WebCT ID=pinto,First Name=Larry,Last Name=Kroger,Courses=
      ENGL100;D:ENGL560;S:ENGL477;TA
```

### FINDING A USER IN THE STUDENT DATABASE

To find a student’s student database record, the following syntax is used:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=find&DB=student
      &COURSE=<course_id>&AUTH=<32_byte_mac>&User%20ID=<user_id> HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | find | Finds a user’s record for a given WebCT ID. |
| | | |
| COURSE | Any alphanumeric string | The course that you are searching. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| User ID | User ID | The User ID of the record you wish to display. |

**Example**
In this example, a complete student database record is displayed for the user with User ID “chip” in the course “HKIN455”:

```
GET  /webct/public/serve_webctdb?OPERATION=find&DB=student
      &COURSE=HKIN455=AUTH=EB1A09F0BB299C23E99A5978587F49C1&User%20ID=chip
      HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success: First Name=Chip,Last Name=Diller,User ID=chip,Quiz1=36,Assignment1=10
```

## CHANGING WEBCT IDS

### CHANGING A USER'S WEBCT ID

To change a WebCT ID for a user, use the syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=changeid&DB=global
      &COURSE=<placeholder>&AUTH=<32_byte_mac>&Old%20ID=<old_webct_id>
      &New%20ID=<new_webct_id> HTTP/1.0
```

where:

| Key | Value | Description |
|---|---|---|
| OPERATION | changid | Changes the WebCT ID of a user. |
| | | |
| COURSE | Any alphanumeric string | This is a required placeholder value. You can use any alphanumeric value, but ensure that you use it in the calculation of the MAC. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |
| | | |
| Old ID | Old WebCT ID | The WebCT ID of the record you want to change. |
| | | |
| New ID | New WebCT ID | The WebCT ID that you want to assign to the user. |

**Example**

In this example, the WebCT ID "flounder" is changed to "dorfmank":

```
GET  /webct/public/serve_webctdb?OPERATION=changeid&DB=global&COURSE=xxxx
      &AUTH=EB1A09F0BB299C23E99A5978587F49C1&Old%20ID=flounder
      &New%20ID=dorfmank HTTP/1.0
```

The Web server returns the following, not including HTTP headers:

```
Success:
```

## EXPORTING MYWEBCT IN XML FORMAT

This Standard API command exports a user's *myWebCT* in XML format, which allows *myWebCT* information to be modified and redisplayed in a desired format. For example, the information can be integrated with a campus portal.

This command can be used in conjunction with automatic signon, allowing for a single point of authentication, see *Chapter 2: Automatic Signon From Other Systems* in this guide.

To export a user's *myWebCT* in XML format, use the syntax:

```
<GET | POST>  /webct/public/serve_webctdb?OPERATION=homearea_xml&DB=global
      &WebCT%20ID=<WebCT ID>&AUTH=<32_byte_mac>
```

where:

| Key | Value | Notes |
|---|---|---|
| OPERATION | homearea_xml | Exports a user's *myWebCT* in XML format. |
| | | |
| WebCT ID | WebCT ID | The WebCT ID of the user whose *myWebCT* you want to export in XML format. |
| | | |
| AUTH | 32_byte_mac | This is the 32-byte hexadecimal string generated using the get_authentication C code, the get_md5 program, or using custom code. |

The returned XML conforms to the DTD located in `<install_dir>/webct/webct/generic/api/xml/mywebct.dtd`.

The XML can be parsed to extract the required elements. Link elements that require authentication by WebCT contain the attribute "secure" with a value of `TRUE`.

# RESOURCES

## LDAP RESOURCES

### WEB SITES

*LDAP Guru*
`http://www.ldapguru.com`
Provides a large database of articles and resources

*Open LDAP*
`http://www.openldap.org`
Home of the OpenLDAP Project

*iPlanet Directory Server*
`http://developer.iplanet.com/tech/directory/`
Information on iPlanet, one of the most commonly used LDAP servers.

## KERBEROS RESOURCES

### WEB SITES

*Kerberos: The Network Authentication Protocol*
`http://web.mit.edu/kerberos/www/`
The home of the free MIT Kerberos implementation

*The Kerberos FAQ*
`http://www.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html`
Updated monthly and has answers to many common questions

*Windows 2000 Kerberos Authentication*
`http://www.microsoft.com/windows2000/techinfo/howitworks/security/kerberos.asp`
Detailed document about how Kerberos works in a Windows 2000 environment

## IMS RESOURCES

### WEB SITES

*The IMS Enterprise Specifications Site*
`http://www.imsproject.org/enterprise/`
The authoritative resource for the IMS Enterprise Information model, the XML Binding Specification and the Best Practices and Implementation Guide.

# APPENDIX

## SUPPORTED CHARACTER SETS

The following character sets are supported by both the Standard API and the IMS Enterprise API:

- Arabic (Windows-1256)
- Baltic (Windows-1257)
- Central European (ISO-8859-2)
- Central European (Windows-1250)
- Chinese Simplified (GB2312)
- Chinese Traditional (Big5)
- Cyrillic (KO18-R)
- Cyrillic (Windows-1251)
- Greek (Windows-1253)
- Hebrew (Windows-1255)
- Japanese (EUC-JP)
- Japanese (Shift_JIS)
- Thai (Windows-874)
- Turkish (Windows-1254)
- Unicode (UNICODE2)
- Unicode (UTF-8)
- Vietnamese (Windows-1258)
- Western European (ISO-8859-1)
- Western European (Windows-1252)