

# ADOBE® CONNECT™ ENTERPRISE

BUILDING CUSTOM TRAINING REPORTS



© 2007 Adobe Systems Incorporated. All rights reserved.

Adobe® Connect™ Enterprise: Building Custom Training Reports

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, and Adobe Connect are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Java is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

RealDuplex™ Acoustic Echo Cancellation is Copyright © 1995-2004 SPIRIT.

This document has accompanying code sample files.

NOTICE: Adobe permits you to use, modify, and distribute the code files in accordance with the terms of the Adobe license agreement accompanying it. If you have received a file from a source other than Adobe, then your use, modification, or distribution of it requires the prior written permission of Adobe. THIS CODE AND INFORMATION IS PROVIDED "AS-IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. THIS CODE IS NOT SUPPORTED BY Adobe Systems Incorporated.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Building a custom training report

If you use Adobe® Connect™ Enterprise to deliver training, you most likely have used the built-in training reports. You may also have wished for a training report of your own design.

This article and its sample code (trainingreport.zip) show you how to create a custom training report for Connect Enterprise 6. You can find instructions for installing and running the sample in its ReadMe file.

The sample report delivers information about users who have taken a specific training and the cost center they belong to, like this:

user	id	training	date taken	cost center
jazz doe	10544	Intro to Marketing	2007-03-16T09:45:37.493-08:00	44444
jamm doe	10551	Intro to Marketing	2007-03-16T09:49:55.233-08:00	11111
amelie doe	22129	Intro to Marketing	2007-03-16T09:44:57.467-08:00	22222
jim jones	25103	Intro to Marketing	2007-02-28T16:30:42.717-08:00	33333
jennifer jones	25107	Intro to Marketing	2007-02-28T16:29:49.863-08:00	34567

You can combine data returned by existing Connect Enterprise reports to make a custom report. However, to create a custom report, you must write code that calls the Connect Enterprise Web Services XML API.

You can also add your own data to a report by creating a custom field. The sample report above has a custom field named *costcenter* and also combines response data from *report-quiz-takers* and *report-bulk-users*.

To write your code, you can use any development environment that supports XML over HTTP. The sample code is written in Java™ and JSP, runs on Tomcat, and uses JDOM and XPath to parse XML responses from Connect Enterprise.

## Create a custom field

The best way to associate a user with a cost center in Connect Enterprise is to create a custom field. You create the custom field once for all users, with either Enterprise Manager or the XML API.

After defining the custom field, you need to give it values. You can add one value at a time in Enterprise Manager or many values at once using the XML API. In addition, you can create the custom field in Enterprise Manager and upload values using the XML API, or create the field using the XML API and upload values using Enterprise Manager.

To build a custom report, you only need to enter custom field values for the users included in the report. You do not need to enter custom field values for all users.

### Define a custom field in Enterprise Manager

- 1 Navigate to Administration, then Users and Groups, then Customize User Profile.
- 2 Click New Field, then enter the field information. Name the field **costcenter**.
- 3 Click Save.

### Define a custom field using the XML API

- 1 Call `custom-field-update` with `object-type=object-type-principal`:

```
http://example.com/api/xml?action=custom-field-update
  &object-type=object-type-principal&permission-id=manage&name=costcenter
  &field-type=text&is-required=false&is-primary=false&display-seq=3
```

The value `object-type-principal` shows that the custom field applies to users, rather than content.

- 2 Parse the `field` element in the response for the `field-id` value, and store it.

The custom field is now defined for all users, and you can add values to it.

### Enter custom field values in Enterprise Manager

- 1 Navigate to Administration, then Users and Groups.
- 2 Click a user name, then Information.
- 3 Click New Information, and enter the custom field value.
- 4 Click Save.

### Enter custom field values using CSV files

❖ Follow the instructions for uploading comma-separated value (CSV) files from Enterprise Manager in *Adobe Connect Enterprise User Guide*.

### Enter custom field values using the XML API

- 1 Display information about the custom field:

```
https://example.com/api/xml?action=custom-fields&filter-name=costcenter
```

- 2 Parse the response for the `field-id` value.

For a custom field, the `field-id` value might start with `x-`.

- 3 Call `acl-field-update` with `acl-id`, `field-id`, and `value` parameters:

```
https://example.com/api/xml?action=acl-field-update&acl-id=2006258745
  &field-id=2007396975&value=33333
```

- Use the `principal-id` of a user as the value of `acl-id`.
- You can use multiple trios of `acl-id`, `field-id`, and `value`. Be sure to use an HTTP `POST` method, rather than a `GET` method, to upload a long list of parameters. If you use a `GET` method, a long parameter list might be truncated.

## View the reports in a browser

Before you write any code, it is helpful to run the XML reports in a browser to see how they work.

- 1 First, call `report-quiz-takers` with the `sco-id` of a training course that has a quiz:

```
http://example.com/api/xml?action=report-quiz-takers&sco-id=12005
```

The response has a `row` element for each user who has taken the course:

```
<?xml version="1.0" encoding="utf-8" ?>
<results>
  <status code="ok" />
  <report-quiz-takers>
    <row transcript-id="12048" sco-id="12005" principal-id="10532"
      status="completed" score="0" max-score="0" asset-id="12032"
      permission-id="" attempts="1" time-taken="4506"
      certificate="12048" answered-survey="1" version="2">
      <name>Take This Job and Love It</name>
      <login>jazz@example.com</login>
      <date-created>2007-01-17T13:10:14.907-08:00</date-created>
      <principal-name>Jazz Doe</principal-name>
      <override>>false</override>
    </row>
  </report-quiz-takers>
</results>
```

The `row` element contains the user's unique `principal-id`, as well as the course name, the user's name, the date the training was taken, and other data.

**2** Choose any `principal-id` value from the `report-quiz-takers` response.

**3** Call `report-bulk-users`, with `custom-fields` set to `true` and a filter on the `principal-id`:

```
http://example.com/api/xml?action=report-bulk-users&custom-fields=true
&filter-principal-id=10532
```

This is the most efficient way to call `report-bulk-users`. It returns one row of information about the user you chose, including the value of `costcenter`:

```
<?xml version="1.0" encoding="utf-8" ?>
<results>
  <status code="ok" />
  <report-bulk-users>
    <row principal-id="10532" type="user">
      <login>joy@example.com</login>
      <name>Joy Smith</name>
      <email>joy@example.com</email>
      <costcenter>12345</costcenter>
    </row>
  </report-bulk-users>
</results>
```

These are the two reports you need to merge to create the custom report. You need to write code to do this, but you have the sample code to guide you.

## Write the code

To generate the custom report, take these main steps in your code:

- 1 Log a user in to Connect Enterprise.
- 2 Obtain the unique ID of a training course, called the `sco-id`.

3 Call `report-quiz-takers` and `report-bulk-users`. The call to `report-bulk-users` returns data from the custom field you have created.

4 Merge data from the two responses and display it.

## Log the user in

After collecting the user's login credentials, write code that makes this call to `login` in the XML API:

```
https://example.com/api/xml?action=login&login=joy@example.com&password=nothing
&account-id=123456
```

The call uses an `account-id` parameter, so that you can run the sample in a hosted environment with more than one account. In the sample code (`XMLApiAdapter.java`), the `login` method makes the call with the user's credentials:

```
protected void login() throws XMLApiException {
    try {
        if (breezesession != null)
            logout();

        URL loginUrl = breezeUrl("login", "login=" + getLogin()
            + "&password=" + getPassword()
            + "&account-id=" + getAccountId() );

        // code omitted
    }
}
```

When the user is successfully logged in, Connect Enterprise returns a status code of `OK` and a `BREEZESSESSION` cookie. Get the value of `BREEZESSESSION` from the HTTP header and store it so that you can send it back to Connect Enterprise as you continue to make calls for the user:

```
String breezesessionString = (String) (conn.getHeaderField("Set-Cookie"));

StringTokenizer st = new StringTokenizer(breezesessionString, "=");
String sessionName = null;

if (st.countTokens() > 1)
    sessionName = st.nextToken();

if (sessionName != null && sessionName.equals("BREEZESSESSION")) {
    String breezesessionNext = st.nextToken();
    int semiIndex = breezesessionNext.indexOf(';');
    breezesession = breezesessionNext.substring(0, semiIndex);
}
```

## Get the sco-id of a course

Next, you need to get the `sco-id` of the course to pass to `report-quiz-takers`. The technique used in the example is to list all training courses in a certain folder. You could also have the user enter a `sco-id` value in an HTML form. The user can get the `sco-id` by navigating to a course in Enterprise Manager and looking for `sco-id=xxx` in the browser URL.

To list courses, write code that calls `sco-shortcuts`, parses the response for the `sco-id` of a folder, and then calls `sco-contents` to list the folder's contents.

The example lists courses at the top level of the Shared Training folder, but you can also list courses in another folder or in subfolders of Shared Training.

First, call `sco-shortcuts` to get the `sco-id` of the Shared Training folder:

```
Element results = request("sco-shortcuts", null);

// specifies the Shared Training folder
XPath shared = XPath.newInstance( "//sco[@type='courses']" );
Element myFolder = (Element) shared.selectSingleNode(results);
XPath scoId = XPath.newInstance("./@sco-id");
String id = scoId.valueOf(myFolder);
```

To use a top-level folder other than Shared Training, change `@type='courses'` to use a different type attribute value returned by `sco-shortcuts`.

Next, call `sco-contents` with the `sco-id` to list the top-level contents of Shared Training:

```
Element contents = request("sco-contents", "sco-id=" + id);
List courses = XPath.selectNodes(contents, "//sco[@type='content']");
Iterator i = courses.iterator();

while ( i.hasNext() ) {
    Element course = (Element) i.next();
    XPath pathToCourseName = XPath.newInstance("./name");
    String name = pathToCourseName.valueOf(course);

    XPath pathToScoId = XPath.newInstance("./@sco-id");
    String courseId = pathToScoId.valueOf(course);

    TrainingInfo thisTraining = new TrainingInfo();
    thisTraining.setCourseName(name);
    thisTraining.setScoId(courseId);
    trainingList.add(thisTraining);
}
```

These lines look for `sco` elements that have the attribute `type="content"`, indicating a course rather than a curriculum.

Notice that the code extracts both the `sco-id` and name of the course and stores them in an instance of the `TrainingInfo` class. `TrainingInfo` has get methods for extracting the values and displaying them in a user interface.

To list the contents of subfolders within Shared Training, you can call `sco-expanded-contents` instead of `sco-contents`. However, creating courses at the top level of Shared Training and calling `sco-contents` often gives better performance.

## Build the report

The next step is to call the two reports and extract values. The `buildReport` method (in `XMLApiAdapter.java`) first makes the call to `report-quiz-takers`, passing it the `sco-id` of the course:

```
Element results = request("report-quiz-takers", "sco-id=" + scoId);
```

Next, it parses the XML response to extract values for the custom report:

```
List quiztakers = XPath.selectNodes(results, "//row");
Iterator i = quiztakers.iterator();

while (i.hasNext() ) {
    Element row = (Element) i.next();

    XPath principalId = XPath.newInstance("./@principal-id");
    String id = principalId.valueOf(row);

    XPath name = XPath.newInstance("./name");
    String tname = name.valueOf(row);

    // code omitted
}
```

These lines get all `row` elements from `report-quiz-takers`, iterate through them, and extract data from certain child elements, starting with `principal-id` and `name`.

Next, for each user returned by `report-quiz-takers`, call `report-bulk-users` with `custom-fields` set to `true` and a filter on the `principal-id`:

```
Element users = request("report-bulk-users",
    "custom-fields=true&filter-principal-id=" + id );
```

This call returns one `row` element that describes the user. Once you have the matching row, extract the value of `costcenter`:

```
XPath cost = XPath.newInstance("./costcenter");
String costcenter = cost.valueOf(onerow);
```

At this point, you have the values you want from each report, and you can store them in data objects. In the sample code, data objects are instances of Java classes that store data and provide get and set methods. The `buildReport` method uses a class named `UserReportInfo` to store data for each user:

```
UserReportInfo thisUser = new UserReportInfo();
thisUser.setCostCenter(costcenter);
thisUser.setTrainingName(tname);
thisUser.setDateTaken(tdate);
thisUser.setUserName(pname);
thisUser.setPrincipalId(id);
```

The method then returns an array of `UserReportInfo` objects, each containing information about one user extracted from two reports. Once you have the array, you can display values in a user interface.

Your code also needs to handle a special case — when a user has taken a course in the past and is returned by `report-quiz-takers`, but no longer exists on Connect Enterprise, so is not returned by `report-bulk-users`.

In this case, the call to `report-bulk-users` does not return a matching row element, so you need to set a string value for `costcenter`:

```
if (matchingUser.size() > 0) {
    // get the matching row from the list
    Element onerow = (Element) matchingUser.get(0);
    XPath cost = XPath.newInstance("./costcenter");
    costcenter = cost.valueOf(onerow);
} else {
    costcenter = "none";
}
```

Users who are returned by `report-quiz-takers` but not by `report-bulk-users` are displayed with the cost center value `none`. You can also define another string.

## Display the report

The example uses a model-view-controller architecture, which allows you to change the view layer while leaving the model and controller the same. In the example, the view is implemented as simple JSP files that retrieve data from data objects on the model layer (specifically, from `UserReportInfo` and `TrainingInfo` objects) using get methods.

The JSP pages (`displayreport.jsp` and `welcome.jsp`) use expressions to call the get methods and display the data, for example:

```
<!-- one table row for each user -->
<tr class="t_row_2">
    <td> <%= userInfo.getUserName() %> </td>
    <td> <%= userInfo.getPrincipalId() %> </td>
    <td> <%= userInfo.getTrainingName() %> </td>
    <td> <%= userInfo.getDateTaken() %> </td>
    <td> <%= userInfo.getCostCenter() %> </td>
</tr>
```

You can also implement the view layer using another user interface technology of your choice.