



macromedia® white paper

Macromedia MX: Creating Learning Objects

by Ann Gallenson, Jay Heins and Tanya Heins

December 2002

Copyright © 2002 Macromedia, Inc. All rights reserved.

The information contained in this document represents the current view of Macromedia on the issue discussed as of the date of publication. Because Macromedia must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Macromedia, and Macromedia cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for information purposes only. MACROMEDIA MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Macromedia may have patents, patent applications, trademark, copyright or other intellectual property rights covering the subject matter of this document. Except as expressly provided in any written license agreement from Macromedia, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

Flash Player, Flash MX, Flash Communications Server MX, Flash Remoting, ColdFusion MX, Dreamweaver MX are either trademarks or registered trademarks of Macromedia, Inc. in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Macromedia, Inc.
600 Townsend Street
San Francisco, CA 94103
415-252-2000

Contents

Creating Dynamic Learning Objects.....	2
A Brief History of the LO Demo	2
Anatomy of a Learning Object.....	2
Audience	3
Enhancing User Experience.....	3
Architecture	5
Tools	5
Database	6
Data “Glue”: Client/Server/Database Communication.....	9
The Learning Object Interface	11
The Learning Area	11
Adaptive Navigation.....	13
Data Transfer	14
The Learning Object Workbench.....	16
Variations on a Theme: Deleting and Creating Objects.....	20
Uploading Media	23
Inside the Workbench: Screen Level - Edit.....	25
Editing Tasks – Preview and Delete Screens	25
Form View – Add and Change Screens	26
Server/Database Communications	27
Conclusion	29
Next Steps	29
Macromedia Learning Objects Development Center.....	30

This white paper outlines the anatomy of a learning object, offers design and development strategies, and suggests technical best practices for building and deploying dynamic learning objects using the Macromedia MX product suite. This document describes the LO Demo application that is used to create dynamic Learning Objects. It is suggested that readers download and install the accompanying demo files prior to reading this paper. The demo files are available for download from the Macromedia Learning Object Development Center at <http://www.macromedia.com/resources/elearning/objects/>.

The Learning Object Demo application provides an example of reusable content maintained in an open-source architecture and is branded using the fictional Darby University theme. In its second version, the LO Demo is constructed using a Rich Internet Application (RIA) approach. Where previous versions of the LO Demo used a RIA for the presentation layer of the actual Learning Object and a HTML/CFMX presentation for the development environment, the revised LO Demo uses a RIA for both the presentation and the LO assembly interfaces, thereby enhancing the user experience.

The Learning Object Demo provides a means to build and display dynamic Learning Objects. It doesn't contextualize them in a course or learning context. Each organization has its own, unique and specific methodology and technical parameters for contextualizing LOs. Because of this, related issues like metatagging choices, integration into a LMS, administrative permissions and controls and asset management are not discussed in this document or addressed within the Learning Object Demo application.

Please Note: The approach, working files and database structure for version 2 have been advanced from the first version of the Learning Object Demo. Before installing version 2 of the LO Demo please be sure to **save earlier versions of the files** should you want to continue developing Learning Objects using version 1 of the LO Demo (that is, using the HTML/CFMX Workbench format).

Creating Dynamic Learning Objects

A Brief History of the LO Demo

The Learning Object Demo project was initiated by Macromedia's Solutions Marketing team to illustrate the integration of MX suite of products, specifically: Flash MX, ColdFusion MX, Flash Communication Server and Dreamweaver MX as the IDE. Originally intended for demonstration purposes only, the LO Demo is used by the Macromedia Sales Teams as a presentation tool. Clients were interested, not only with the company's products, but with the Learning Object application itself. It was recognized that a dynamic Learning Object application fulfilled a greater need in the eLearning and higher education communities. This demo has therefore been made publicly available as an example of dynamic LO implementation with the MX product line.

It is the hope of the Learning Object demo team, that eLearning developers will adapt the LO application to their own particular needs. Only by a process of deconstructing, analyzing and re-assembling the application – in short, by breaking it and rebuilding it – will developers create a model that works within their organizations' specific instructional design framework and related taxonomies. In this way, we aim to advance the technology, encourage adoption and improve implementation of reusable content.

Anatomy of a Learning Object

For a moment, let us review the pedagogical structure of a Learning Object as described in the *Macromedia MX: Strategies and Architectures for E-Learning Content* whitepaper: it is a unit of instructionally sound content centered on a learning objective or outcome intended to teach a focused concept. A LO may contain opportunities for practice, simulation, collaborative interaction, assessment, and educational resources.

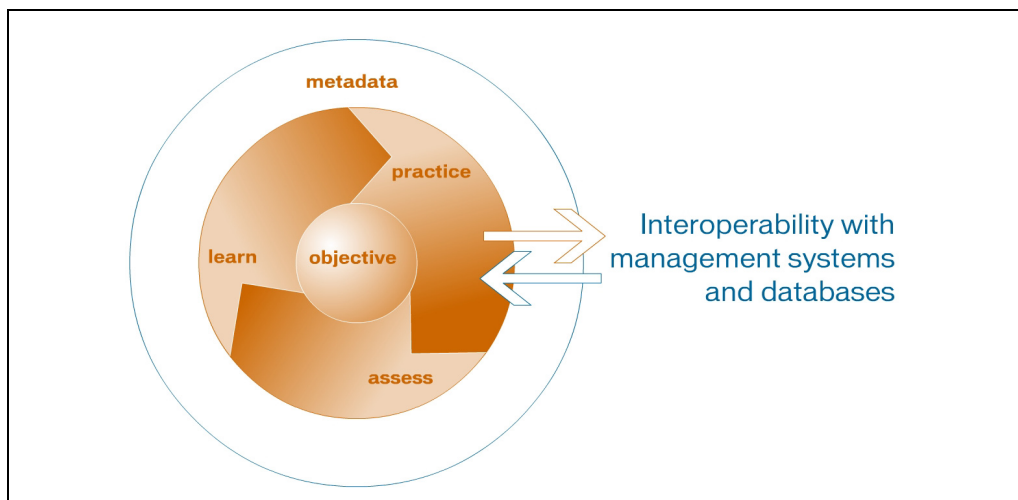


Figure 1: Structure of a Learning Object

A LO is constructed from Media Assets, such as paragraphs of text or html, screen titles, captions, video, animation, diagrams, and sound narration. It is using this model that the Learning Object Demo application has been built.

Audience

When examining the Learning Object Demo application, it is important to consider the audience, which can be separated into two groups. The **first** group is the technical team involved in the implementation of the application. The **second** includes the users of the application itself, the content developers, instructional designers and subject matter experts.

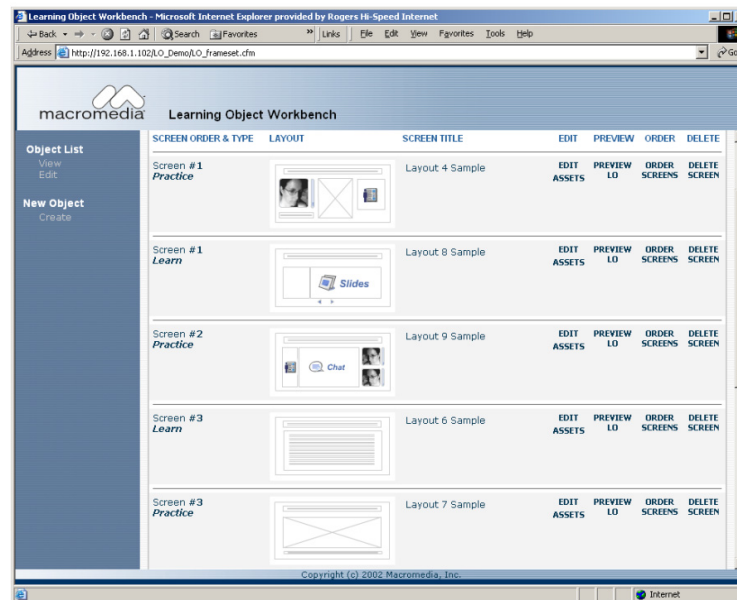
The technical, or IT team will be responsible for installation, management, customization and administration of the application on the organization's server. The skills required for this individual or group includes skills of a: general web developer, server administrator, database administrator, Flash developer and ColdFusion developer.

The second team is comprised of those that will be using the application. These include SMEs, instructional designers, graphic designers and usability and QA testers who will design, build and test the use of the Learning Objects.

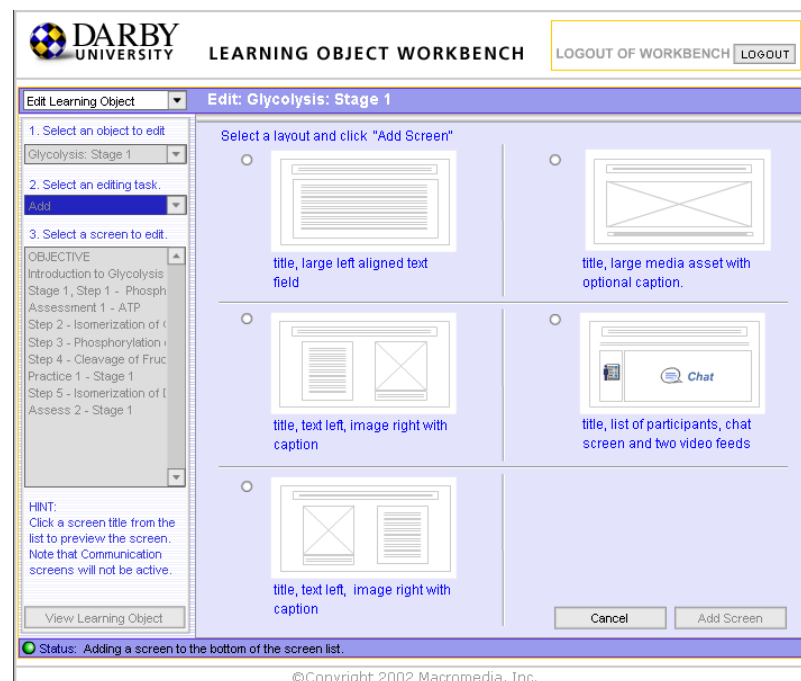
Enhancing User Experience

Version 1 of the LO Demo project began by constructing a ColdFusion back end (development environment) and a Flash MX front end (delivery environment). The communication model between the two was primarily a one-way street with ColdFusion packaging up the data in XML and sending it to the Learning Object to parse. The only two-way communication would come through a chat screen layout using the Flash Communication Server.

On review of the first version of the demo it became clear that the Workbench could provide an enhanced user experience that is more responsive and intuitive if a Rich Internet Application interface was used to communicate with the server via Flash Remoting. This coupling harnessed data manipulation web service power of ColdFusion and the interactive presentation ability of Flash to produce a web-able application without the standard browser-based HTML limitations.



ColdFusion / HTML Interface



Flash MX Interface

Figure 2: Before and after views of the LO Workbench environment

Architecture

The architecture behind the second version of the LO Demo application is comprised of the Database, Flash Communication Server MX, the ColdFusion MX server application and the use of the Flash Player 6 in the users web browser. The Flash user interface communicates with ColdFusion component web services through Flash Remoting MX and the browser to server communication for the Flash Communication Server MX components are handled through shared objects.

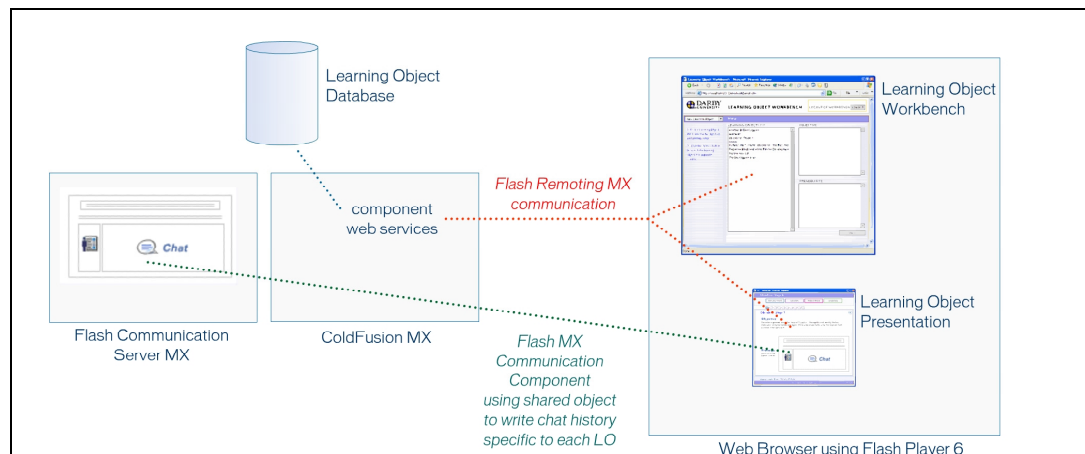


Figure 3: *Architecture of the Learning Object Demo Application*

Tools

Table 2 summarizes the Macromedia tools used to develop the Learning Object architecture. For further details and documentation please visit the appropriate product area of the Macromedia website.







Logo	Product	Description
	Flash MX	A developers' tool to create rich Internet content and applications.
	Flash Player	Allow end users to view rich content and applications as stand-alone applications or through a web browser.
	Flash Remoting MX	Connects Macromedia Flash applications to server side application logic and web services.
	Flash Communication Server MX	A server solution for uniting communications and applications
	ColdFusion MX	Server Environment to rapidly build and deploy Internet applications.
	Dreamweaver MX	A development tool for building websites and web applications.

Table 1: Tools used to develop the Learning Object Demp app architecture.

Database

The design of the database (see **Figure 4**) was created directly from the Learning Object instructional design model discussed in the whitepaper: **Macromedia MX: Strategies and Architectures for E-Learning Content**. The database (filename: learningobjects2.mdb) tables reflect a particular learning object model and the fields have a specific relationship to each other. For resources on designing databases and using SQL please visit the Macromedia website at: http://www.macromedia.com/support/training/instructor_led_curriculum/ft_to_sql.html.

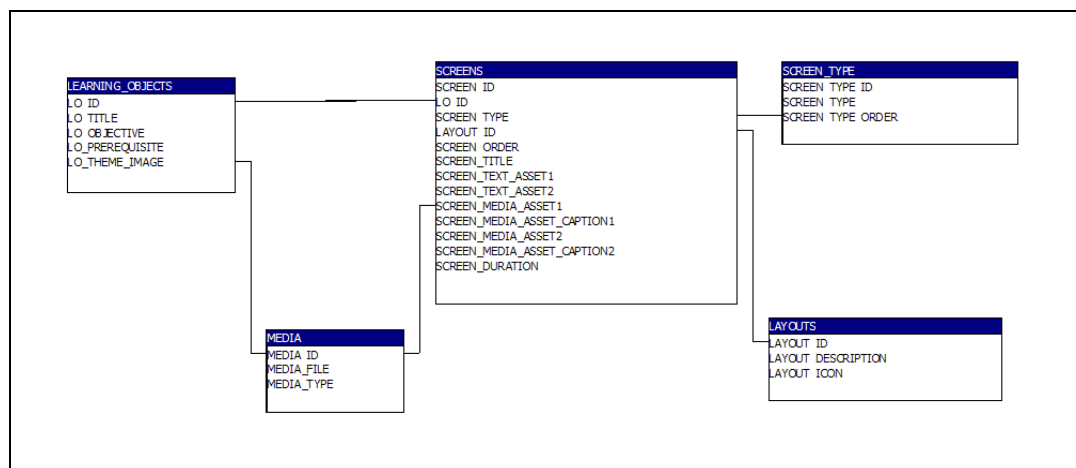


Figure 4: Relationship of database tables in the LO Demo Application

The descriptions below provide a reference for each of the fields in the database tables and their relationships as certain fields key off information in other tables.

Field Name	Description
LO_ID	Primary Key. Unique identifier for each Learning Object
LO_TITLE	Title of the Learning Object
LO_OBJECTIVE	The objective for the Learning Object
LO_PREREQUISITE	The prerequisite knowledge needed to successfully complete the Learning Object
LO_THEMEIMAGE	Linked to the MEDIA table, MEDIA_ID field. The graphic that appears as a faded background of the Objective Screen

Table 2: *LEARNING_OBJECTS Table*

Field Name	Description
SCREEN_ID	Primary Key. Unique identifier for each Screen
LO_ID	Direct relationship to the LEARNING_OBJECTS table, LO_ID field
SCREEN_TYPE	Linked to the LAYOUTS table, LAYOUT_ID field. Each screen may be classified as either a Learn, Practice or Assess screen type
LAYOUT_ID	Linked to the LAYOUTS table, LAYOUT_ID field. Each screen uses a layout from a library of pre-determined layout templates.
SCREEN_ORDER	The order in which the screen should appear
SCREEN_TITLE	The Title of the Screen
SCREEN_TEXT_ASSET1	Text content. This field is used as required by the screen layout template.
SCREEN_MEDIA_ASSET1	Linked to the MEDIA table, MEDIA_ID field. JPG Image or Flash SWF file. This field is used as required by the screen layout template.
SCREEN_MEDIA_ASSET_CAPTION1	Text Caption. This field is used as required by the screen layout template.
SCREEN_TEXT_ASSET2	Text content. This field is used as required by the screen layout template.
SCREEN_MEDIA_ASSET2	Linked to the MEDIA table, MEDIA_ID field. JPG Image or Flash SWF file. This field is used as required by the screen layout template.
SCREEN_MEDIA_ASSET_CAPTION2	Text Caption. This field is used as required by the screen layout template.
SCREEN_NARRATION	Audio File. This field is not currently implemented.
SCREEN_DURATION	Estimated time to complete the Screen of content. These durations are added together to provide an estimate duration for the entire Learning Object

Table 3: *SCREENS Table*

Field Name	Description
SCREEN_TYPE_ID	Primary Key. Unique identifier for each Screen Type
SCREEN_TYPE	Each screen may be classified as either a Learn, Practice or Assess screen type
SCREEN_TYPE_ORDER	The order in which the 'Learn', 'Practice' or 'Assess' buttons are presented in the Learning Object

Table 4: *SCREEN_TYPE Table*

Field Name	Description
LAYOUT_ID	Primary Key. Unique identifier for each screen layout template
LAYOUT_DESCRIPTION	Text describing the screen layout template.
LAYOUT_ICON	Filename of the icon representing the screen layout template

Table 5: *LAYOUTS Table*

Field Name	Description
MEDIA_ID	Primary Key. Unique identifier for each media asset
MEDIA_FILE	The name of the file on the server
MEDIA_TYPE	Type of file. 1=jpeg, 2= Flash swf

Table 6: *MEDIA Table*

Developers implementing the LO Demo may use their choice of ODBC compliant databases by importing the learningobjects.mdb file into the appropriate database. Once the database is in place a Data Source Name needs to be setup. This is accomplished by registering the database in the ODBC Data Source Administrator on the server (see **Figure 5**). The ODBC DSN also needs to be registered with the ColdFusion Server. (see **Figure 6**)

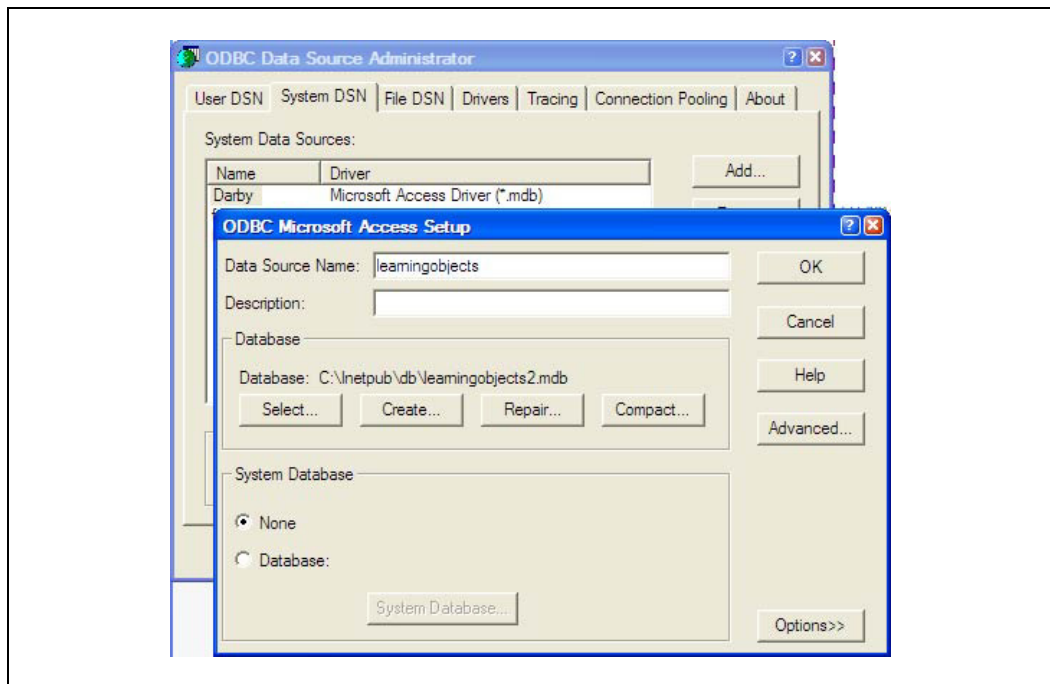


Figure 5: Control Panel ODBC settings.

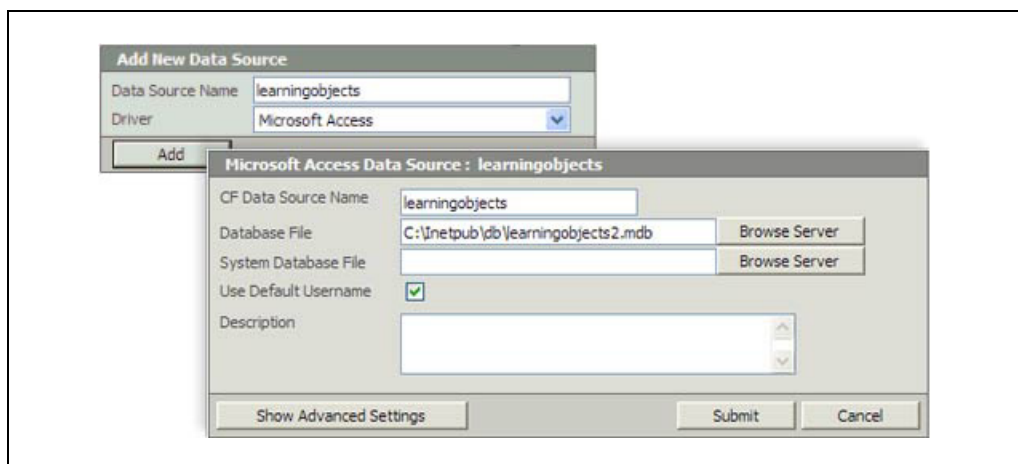


Figure 6: ColdFusion DSN settings.

Data “Glue”: Client/Server/Database Communication

Whether it is by parameters, arguments, XML, Flash Communication components or Coldfusion components; lines of communication must be open between the Client, Server, and Database. ColdFusion components (CFCs & Flash Remoting) were selected as communication methods in the LO Demo application because they are an easily a customizable solution that delivers clean, fast data.

ColdFusion is an efficient tool for working with databases. With a new partner, Flash, ColdFusion comes into its own and they make a spectacular combination. In ColdFusion you can write your own communication components. Don't panic, it is truly straightforward and not too complicated. Developers will need a good SQL guide. As mentioned earlier there is a good instructor-led course available on the Macromedia website at: http://www.macromedia.com/support/training/instructor_led_curriculum/ft_to_sql.html.

In this demo, this Learning Object presentation UI uses one CFC with two functions (loData.cfc) and the Workbench uses one CFC with eleven functions (loWorkbenchData.cfc). All the LO Demo application methods in one place!

Flash Remoting MX provides the connection between Flash MX and the web application server. It has a powerful yet simple programming model, you can easily integrate Flash user interfaces with applications built using Macromedia ColdFusion MX, and also Microsoft .NET, Java, and SOAP-based web services.

A download of pre-built Flash Remoting MX components along with the Macromedia Rich Application Starter Kit can be found for free download at; <http://www.macromedia.com/software/flashremoting/downloads/>.

The Learning Object Interface

The Learning Object Interface is the view that the learner or content expert testing the Learning Object would use. It consists of three basic parts:

- 1 The communication / instructional delivery area and the forward and back arrows. This is the Learning Object content.
- 2 Adaptive navigation areas that include titles and navigation based on screen type and duration.
- 3 The invisible communications between the Flash interface and the server/database (and potentially a Learning Management system).

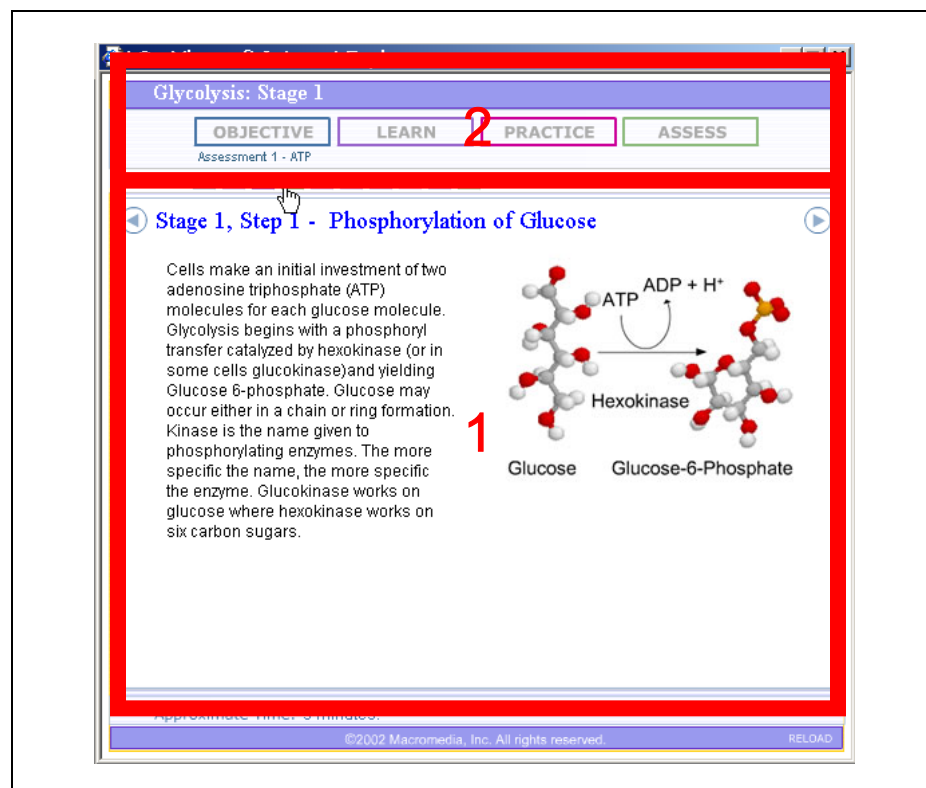


Figure 7: The user's view of the Learning Object. Learning space headed with adaptive navigation.

The Learning Area

Let us compare two views of the learning area from user's perspective (**Figure 8**) as well as from the developer's position (**Figure 9**):

Stage 1, Step 1 - Phosphorylation of Glucose

Cells make an initial investment of two adenosine triphosphate (ATP) molecules for each glucose molecule. Glycolysis begins with a phosphoryl transfer catalyzed by hexokinase (or in some cells glucokinase) and yielding Glucose 6-phosphate. Glucose may occur either in a chain or ring formation. Kinase is the name given to phosphorylating enzymes. The more specific the name, the more specific the enzyme. Glucokinase works on glucose where hexokinase works on six carbon sugars.

Figure 8: The user's view of the learning area.

Properties Variables Locals Watch	
_global	
Name	Value
objectToStringTree	
onClick	
RecordSet	
relatedLOs	
RsDataFetcher	
RsDataProviderClass	
screenList	
0	
1	
2	
_ID	1
layout_id	1
lo_id	12
screen_duration	5
screen_media_asset_caption1	"Production of glucose 6-phosphate."
screen_media_asset1	"glyc1.jpg"
screen_order	2
screen_text_asset1	"Cells make an initial investment of two adenosine triphosphate (ATP) molecules ..."
screen_title	"Stage 1, Step 1- Phosphorylation of Glucose"
screen_type	2
3	
4	

Figure 9: The developer's view of the learning area.

The learning space is essentially a multimedia slide show focused around a learning objective. The categories listed in **Figure 9** (layout, screen parameters, media assets, caption, etc.) are common to all screens. Some of the values are default settings and the specified layout may not use them (for example: image information is not required in a text only layout).

A technical definition of a learning object screen is that each screen is identified by its screen identification number (ID) and the learning object number (lo_id) to which it belongs. The content is divided into screen duration, screen media asset caption1, screen media asset file name, test and title. The form this content takes is determined by the layout_id and its context is screen_type (for example: screentype 2 is a learning screen, as opposed to objective, practice or assess) and its order in the series of slides. That is the essence of a screen object. The code for the learning screen delivery is simple and linear – users are able to navigate both forward and backward using linear directional arrows.

Adaptive Navigation

Now we put the screen in the context of the learning process and we give the user/learner the ability to peruse the entire object in different ways.

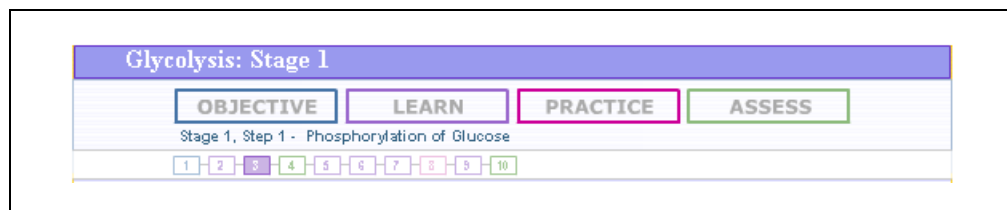


Figure 10: Navigation that identifies screens based on their type: Objective; Learn; Practice and Assess.

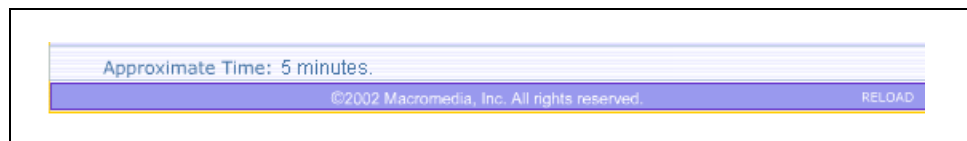
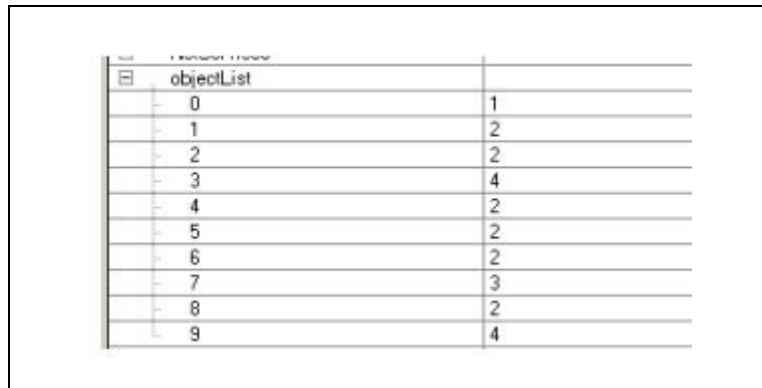


Figure 11: Estimated duration of the Learning Object

The Learning Object title orients the learner to the LO content. Rolling over the screen buttons reveals the title of each screen. Random navigation is possible by clicking on any of the screen number buttons. The user can also isolate screens by clicking the screentype buttons (these include: objective, learn, practice or assess). If 'Assess' is clicked, the assess screens are highlighted (4 and 10 in this example) and the screen buttons may then be clicked on for display.

This navigational feature accommodates differences in learning styles and allows for quick review of relevant material and prior learning assessment. A learner may, for example, jump to an assessment, which results in a screen that indicates the specific screen numbers that still require review.

The developer's view of this is that each screen's "type" is stored in an array when the data is loaded. The other screentypes include 1=Learn, 2=Practice, 3=Assess (**Figure 10**). Developers extending this demo application may choose to add their own screentypes.

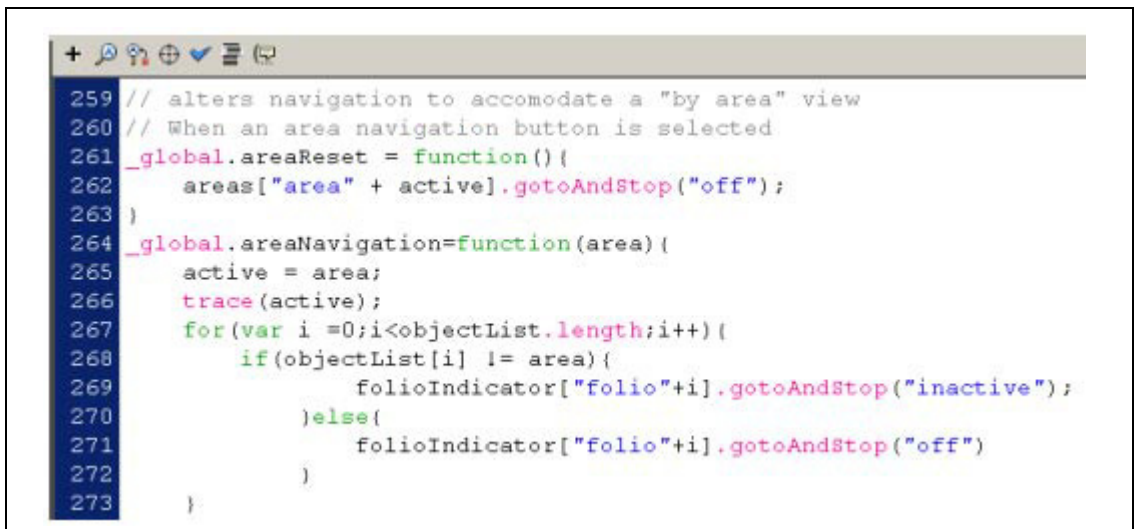


The screenshot shows a table titled 'objectList' with two columns. The first column contains indices from 0 to 9, and the second column contains corresponding values: 1, 2, 2, 4, 2, 2, 3, 2, 4.

0	1
1	2
2	2
3	4
4	2
5	2
6	2
7	3
8	2
9	4

Figure 12: Developers view of screentype ID and screen order.

The screentype ID and screen order determines the individual coloring of the screen icons, as well as which buttons are active or inactive when a particular screentype button is pressed. When a screentype button is clicked an “active area” variable is declared (the value is an integer between 1 and 4) and the array (objectList) is scanned to determine which numbered buttons are active.



```
259 // alters navigation to accomodate a "by area" view
260 // When an area navigation button is selected
261 _global.areaReset = function(){
262     areas["area" + active].gotoAndStop("off");
263 }
264 _global.areaNavigation=function(area){
265     active = area;
266     trace(active);
267     for(var i =0;i<objectList.length;i++){
268         if(objectList[i] != area){
269             folioIndicator["folio"+i].gotoAndStop("inactive");
270         }else{
271             folioIndicator["folio"+i].gotoAndStop("off")
272         }
273     }
```

Figure 13: Flash ActionScript function to determine whether a screentype button is for ‘Learn’, ‘Practice’ or ‘Assess’.

Data Transfer

There are two sets of data that are passed to the learning object. The first occurs in the lo.cfm document that houses the lo.swf (the learning object) file.

```

7 <meta http-equiv=Content-Type content="text/html; charset=iso-8859-1">
8 <TITLE>LO</TITLE>
9 </HEAD>
10 <BODY bgcolor="#FFFFFF" leftmargin="0" topmargin="2" marginwidth="0" marginheight="0">
11 <div align="center">
12 <OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" codebase=
    "http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=6,0,0,0" WIDTH='
    HEIGHT="504" ALIGN="middle">
13 <PARAM NAME=movie VALUE="LO.swf?varLO_ID=<cfoutput>#rsLOinformation.LO_ID#</cfoutput>&var
    <cfoutput>#rsLOinformation.LO_TITLE#</cfoutput>&varStudentName=Tanya">
14 <PARAM NAME=quality VALUE=high>
15 <PARAM NAME=bgcolor VALUE=FFFFFF>
16 <EMBED src="LO.swf?varLO_ID=<cfoutput>#rsLOinformation.LO_ID#</cfoutput>&varLOtitle=<cfou
    #rsLOinformation.LO_TITLE#</cfoutput>&varStudentName=Tanya" WIDTH="554" HEIGHT="504" ALIGN='

```

Figure 14: CFM page showing the passing of the LO_ID to the Flash file.

The LO_ID variable (the learning object identification number from the database) is passed in the URL when the window is first opened. A CF recordset retrieves specific parameters from the database, based on the LO_ID. These are then passed to the movie in the Flash Object/Embed tags.

Once the preliminary data is passed the movie then establishes a link with the database through a web service (this is Flash Remoting) and calls two queries that download the data. **Note:** the netservices gateway URL for Flash Remoting is set to 'localhost' in the LO Demo asset files.

```

40 //Load data
41 // include Flash Remoting scripts.
42
43 _global.screenList = new Array();
44 #include "NetServices.as"
45 #include "DataGlue.as"
46 //comment this out on final publish
47 #include "NetDebug.as"
48 if(isGatewayOpen == null){
49     /* perform this code once - conditional statement that checks if a connection
50     has already been made. If not, the condition is set to true and the connection
51     is made.*/
52     isGatewayOpen = true; //sets variable.
53     NetServices.setDefaultGatewayURL("http://localhost/flashservices/gateway");
54     /* sets the connection does not "make" the connection.
55     The url may need to change depending on the server location.
56     This should be set dynamically -- shared as include? Shared Obj?*/
57     gatewayConnection = NetServices.createGatewayConnection();// connects and sets object
58     loDataService = gatewayConnection.getService("LO_Demo.components.loData",this);
59     trace ("Connected.");
60     loDataService.LO_Objective(loID);
61     loDataService.LO_Screens(loID);
62     trace ("sent request");
63     Layouts.activeArea.text="LOADING...";
64 }
65 function LO_Objective_Result(objective){
66
67     trace ("server responded : records: " + objective.getLength());

```

Figure 15: Flash Action Script establishing a Flash Remoting Service call

When a Flash Remoting service is established and a data call is made, only the minimum amount of required data is delivered in the form of an object. This object can now be stored as a variable or used directly as a data source for a component.

The Learning Object Workbench

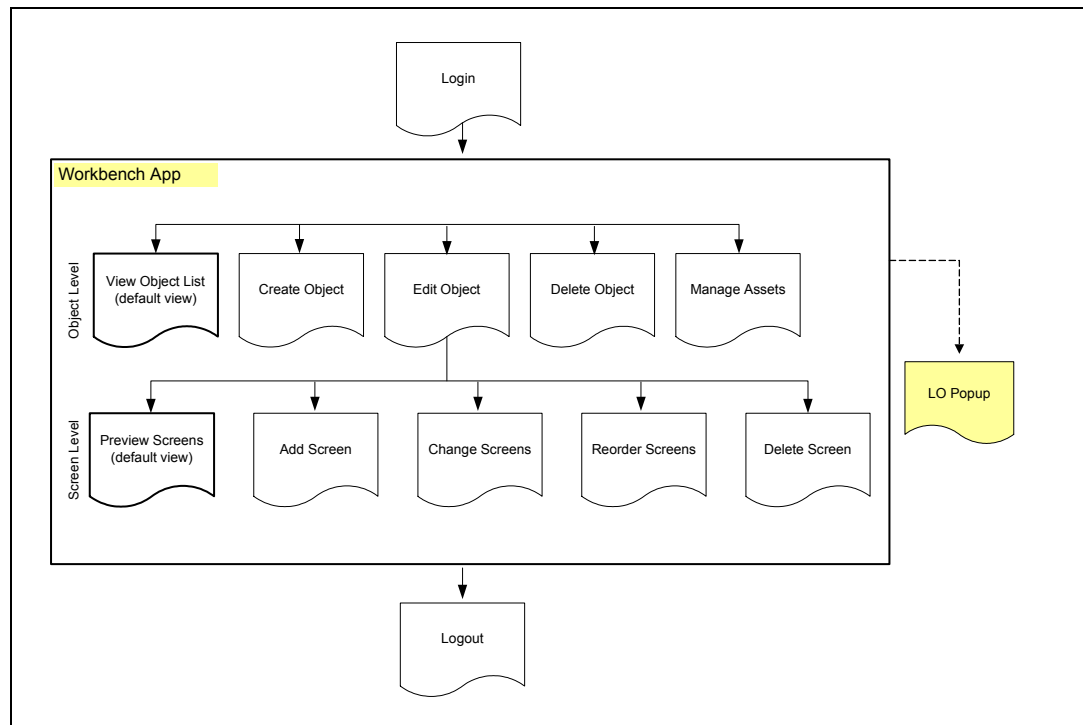


Figure 16: Workflow for users in the Learning Object Workbench

When first viewing the Learning Object Workbench, users are presented with a list of available Learning Objects. The Status light at the bottom of the screen indicates that a user is on line and connected. A red status light indicates a lack of positive connection. Users might try to refresh the screen in order to gain a connection. The task combo box at the left allows navigation through all of the Object tasks.

Clicking through the list of learning objects allows a user to view each Learning Object's unique Objectives and the Prerequisites. As well, the Learning Object may be launched in a separate pop-up window for review. A banner on the Workbench indicates the active task and which Learning Object is in progress. The area by the "Status" light will provide messages regarding Workbench functions and secondary task status. Task instructions and components appear on the left, the main workspace is on the right.

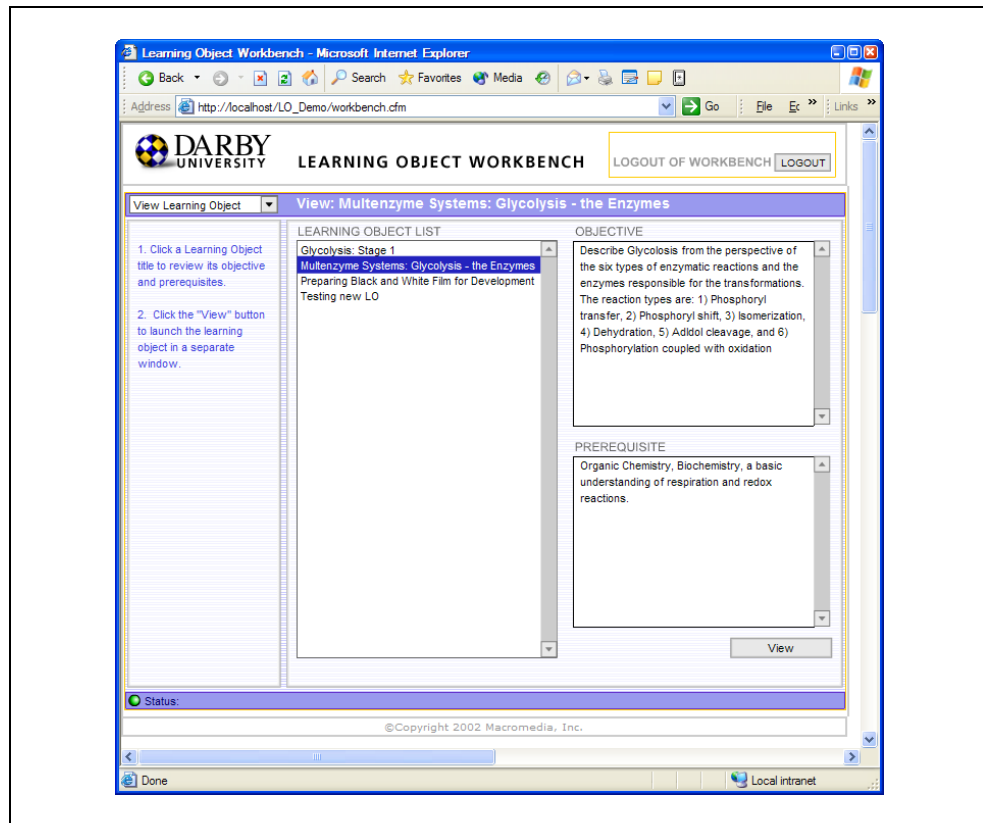


Figure 17: The interface user are presented with after logging into the Workbench

There is a preliminary Service call that checks the connection and controls the status indicator. Let's look at the communication loop, starting with the Flash call:

```
70 // was the connection successfull?  
71 _global.online=false  
72 _global.checkConnection=function() {  
73   trace ("Connecting");  
74   loWorkbenchService.onlineTest();  
75   trace ("sent request");  
76 }  
77 function onlineTest_Result() {  
78   status.indicator.gotoAndStop ("active");  
79   status.traceout.text="Connected"  
80   _global.online=true;  
81 }
```

Figure 18: Flash MX Actionscript funtion testing that a user is connected

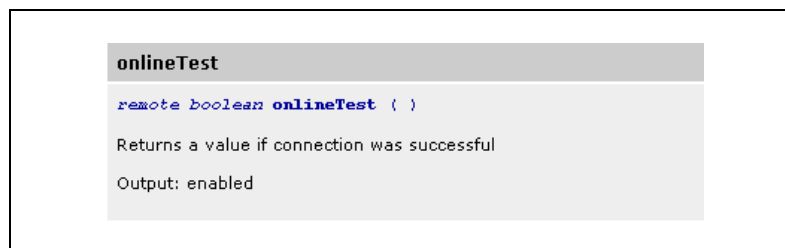
During the establishment of the connection the service is named, in this case it is named “loWorkbenchService”, which will persist throughout the use of that service. The call – “loWorkbenchService.onlineTest()” – can be abstracted for meaning as - <serviceName.method>. The service will respond with a <method>_Result(); in this case, onlineTest_Result(). For further information, please refer to the Flash Remoting documentation at; <http://www.macromedia.com/software/flashremoting/>.

By examining the ColdFusion code for the method we find that this function (the first in the series that makes up the cfcomponent) simply returns true if the connection is made. This is a way to test the connection while the connecting the indicator provides a real-time reading of the server status.

```
1 <!-- Start ColdFusion Component -->
2 <!-- The component is called from the Learning Object Workbench Interface -->
3 <cfcomponent>
4
5     <!-- ::Function - This is an on-line test that returns a value if connection was successful-->
6     <cffunction access="remote" name="onlineTest" returntype="boolean" hint="Returns a value if
7 connection was successful">
8         <cfreturn true>
9     </cffunction>
```

Figure 19: ColdFusion component summary.

The method appears in the component summary as:



The screenshot shows a component summary for the 'onlineTest' method. It is titled 'onlineTest' in a bold header. Below the title, the code 'remote boolean onlineTest ()' is displayed. Underneath the code, the text 'Returns a value if connection was successful' is shown. At the bottom, it says 'Output: enabled'.

Figure 20: Component summary.

Looking around the interface, the main UI focus is on the task at hand and all of the “shell” functions are understated and on the periphery. Instructions for a specific component or area is adjacent to the component or work area. The main work area is divided into an instruction/subtask column and a primary working space.

The “Log out” panel is in the banner and can send a message to end the server session. The task combo box is populated by an internal array. The basic rule here is that if the information (variable, array or object) is relatively stable (tasks won’t change very often, if ever) then the information resides in the application code, otherwise it resides in the database.

There are two Message areas. The first is the task message that is adjacent to the combo box and displays the active task and the object being worked on. The second is the status area. This is where a function method (creation, deletion) result is reported, screen editing and other secondary messages (connection, screen editing task activation) are displayed.

Here's one more bit of initiating code before looking at tasks. Global variables and initial conditions are set in an initiating method. In this code there is one convention worth showing.

```
function initWorkBench() {  
    // set root designator for movie clips and object.  
    _global.wb_root = this;  
}
```

Figure 21: Flash MX Actionscript setting the *wb_root* for all movie clips as a global variable

The root is given a global variable, in this case *wb_root* (where 'wb' is short for Workbench). When you want to refer to the root, use *wb_root*. This allows you to load this movie clip into another movie or incorporate it and not lose functionality – no more root confusion.

Let's now examine a specific task – the View Object List. The list itself is a listbox that has been populated by a service call (*loWorkbenchService.getLOScreens*) for the entire contents of the Learning Object table of the database. The listbox label is the title of the object and the data is the object ID. This organization assumes a linear learning object structure. When a Learning Object title is clicked on, the adjacent text boxes display the Objective and Prerequisite text.

The full Learning Object may be viewed by selecting a title from the list and clicking the 'View' button on the bottom of the screen. The 'onClick' code on the button gets the learning objects ID and sends a javascript command through 'Get URL'. The javascript then passes the learning object id as a variable to the child window that is launched. The Learning Objects ID could be obtained getting the data from the selected listbox item:

```
lb_objects.getSelectedItem().data
```

Where *lb_objects* is the instance name of the listbox component that contains the list of objects.

```
// Learning object list is clicked on View and Delete.  
if (component._name=="lb_objects") {  
    with(main) {  
        _global.loID= lb_objects.getSelectedItem().data;  
        trace(loID);  
        viewObjective.htmlText=loList.items[lb_objects.getSelectedIndex()].LO_OBJECTIVE;  
        viewPrereq.htmlText=loList.items[lb_objects.getSelectedIndex()].LO_PREREQUISITE;  
        var whichPane =cb_task.getSelectedIndex();  
        taskDescription.text=panes[whichPane]+" "+loList.items[lb_objects.getSelectedIndex()].LO_TITLE;  
    } //end with main  
}
```

Figure 22: *Flash MX Actionscript using the global variable loID*

The method used to declare a global variable “loID” when an item in the object listbox is selected declares it’s unique ID as the global loID. The loID is a unique identifier for structuring the Learning Object. This LO_ID field in the database and is used almost all of the SQL statements to create a Learning Object. The same method is applied to the ScreenID (declare a global variable).

Variations on a Theme: Deleting and Creating Objects

The Delete Pane is almost identical to the View Pane. A list of objects is presented, where a user clicks on a LO title to see the Objective and the Prerequisite for that LO. The full Learning Object can be launched from the ‘View’ button.

Clicking ‘Delete’ causes a prompting - a warning dialog box that allows you to cancel the action or continue a Learning Object deletion.

The ‘Cancel’ button closes the window without changing the list. By clicking on the dialog box ‘Delete’ button and the Learning Object that is selected in the list box is deleted, thence the window closes and the object list is refreshed.

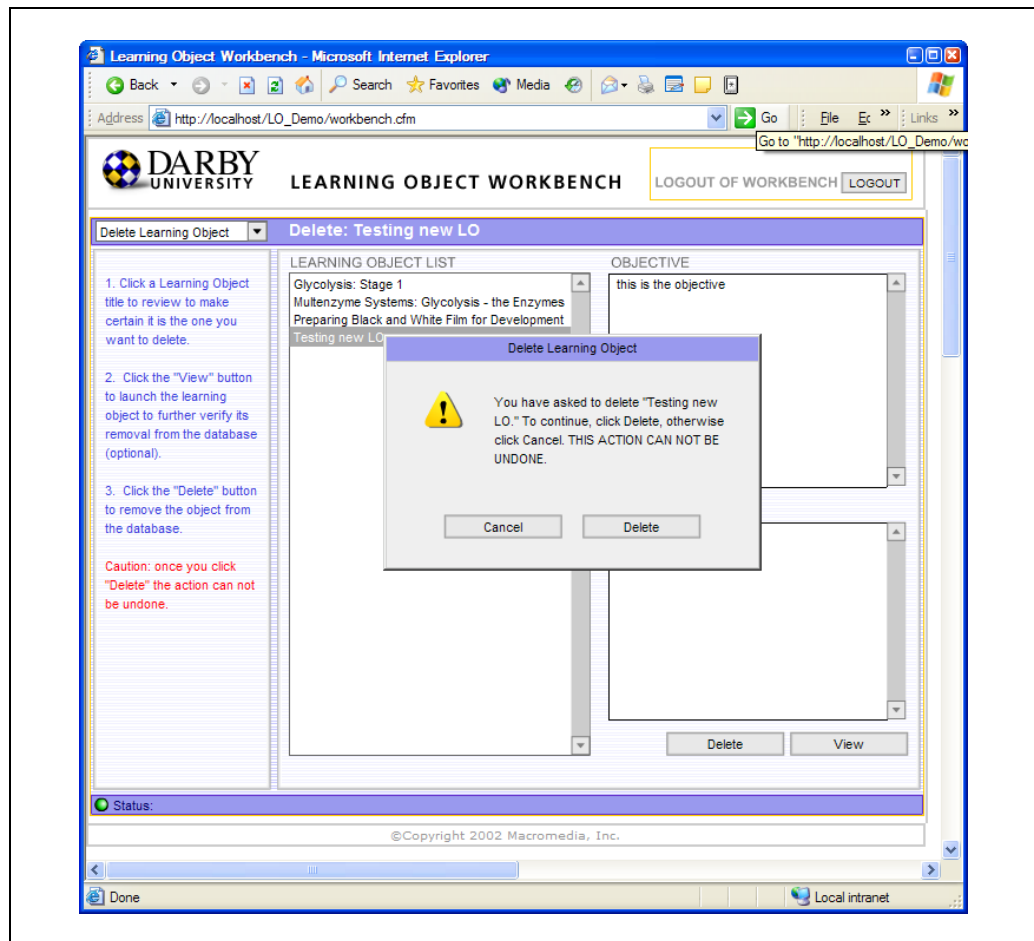


Figure 23: Confirmation of the 'Delete LO' task.

The Create Learning Object is very simple and presents a screen that compliments the View Object screen. From this screen the Title, Objective and Prerequisite for the new Learning Object are filled in to the text fields. 'Clear', removes all of the text from the form and 'Create' validates the form and enters the new Learning Object into the database. When the database transaction is complete the View screen appears with your new learning Object in place.

The Delete Screen's ActionScript function is identical to that of the View screen except that when the 'Delete' button clicked a Message Box component is placed on the screen. The user is given two opportunities to cancel before the delete service call is made. Clicking the 'Delete' button in the message box makes a service call to the server, which activates a SQL delete item statement (the loID is passed to the server). Here is the ColdFusion component's function:

```
<!-- ::Function - Delete Learning Object....-->
<cffunction name="deleteLO" hint="Deletes Learning Objects" access="remote" returntype="string">
<cfargument name="deleteSelected" type="numeric" required="true" default="">
<cfquery name="q_deleteLO" datasource="learningobjects">
DELETE FROM LEARNING_OBJECTS WHERE LO_ID =#arguments.deleteSelected#
</cfquery>
<cfset objectDeleted = "deleted">
<cfreturn objectDeleted>
</cffunction>
```

Figure 24: ColdFusion Component Function to delete a Learning Object

When the deletion is complete the server returns a string and the Flash Movie updates the list (by calling getLearningObject method used to populate the list in the first place). The Workbench reuses a few simple functions and variables to keep the database information up to date and to make the application responsive to the users actions. In addition to refreshing the list, the status area shows that the object (by title) was deleted.

The responsiveness (refreshing lists –without refreshing the whole screen, displaying status messages, sliding up windows when needed and removing them when the task is complete) of the interface puts the user in an “application” environment. This allows the user to focus on their primary task without being deterred by clunky browser page redraws typical of an HTML-style environment.

The Create pane is a simple compliment to the Learning Object views we have seen thus far. It is three input text boxes, a ‘Clear’ (to remove text) and a ‘Create’ button. Notice that the action buttons (‘View’, ‘Delete’, and ‘Create’) are disabled upon first entering the pane. An action (clicking a title or entering text) enables them.

When the user clicks the ‘Create’ button the form is validated and a function call is made to the server. As it is not possible to pass a blank variable, default values must be inserted into the empty fields. When the new object is created the server returns a string which prompts the Workbench to return the user to the View Object List pane and refresh the list.

Uploading Media

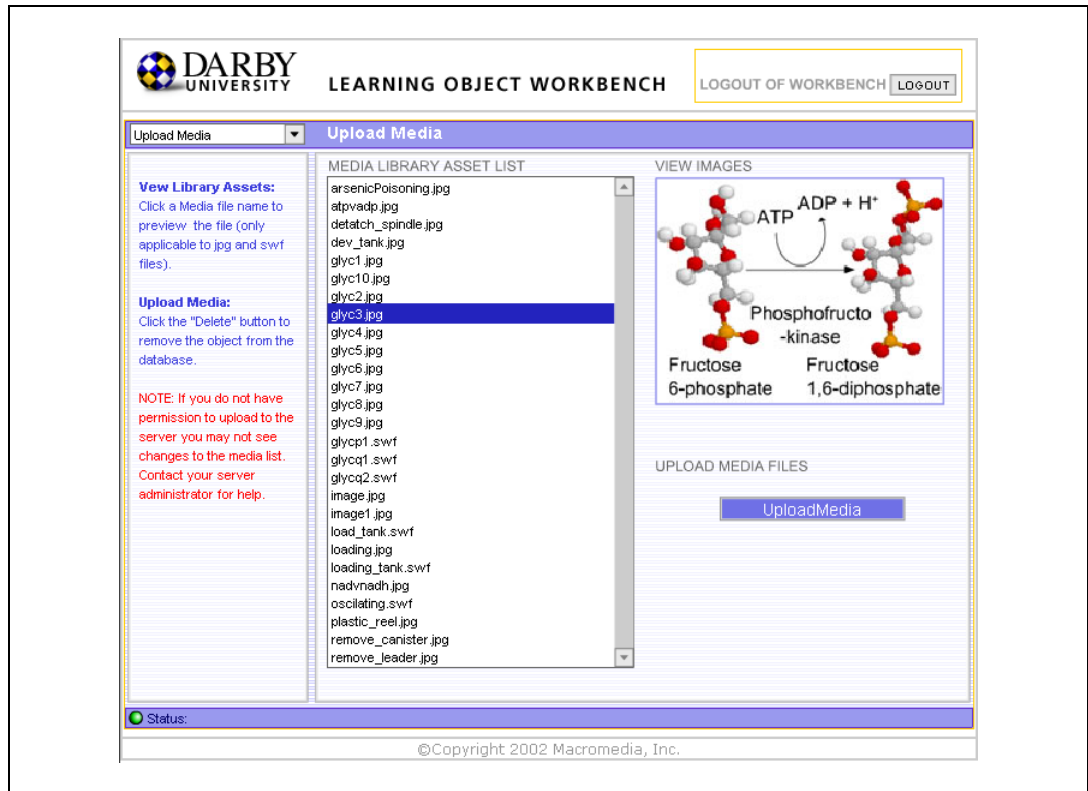


Figure 25: Media Asset Library List

Media Assets can be added from the Object or Screen Level of the Learning Object Workbench. This pane in the Workbench lets the user peruse the media library for assets and then up if user doesn't find what they want they click a button and are given a popup form to upload files.

Clicking on the 'Upload Media' button opens a popup window containing a file named Upload_Media.cfm. This cfm page uses ColdFusion MX server functionality to transfer a suitable media file to the server through the browser. The figure below shows the UI of the Media upload window that allows the user to browse for the image or Flash file they wish to upload from a location on their computer.

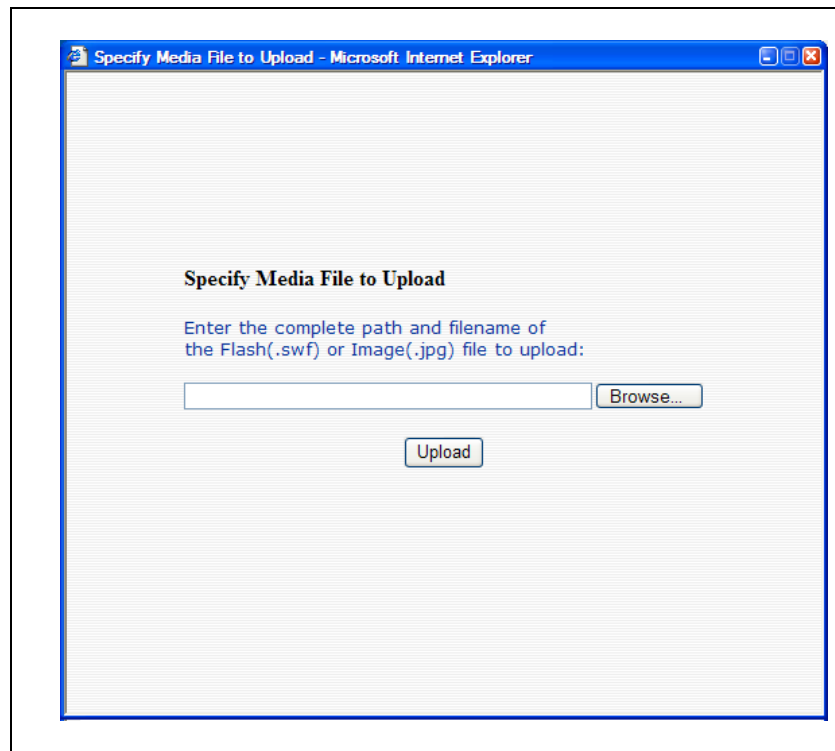


Figure 26: *Uploading Media Assets*

A second page, Upload_File_Action.cfm then processes the file and saves it to a specific location on the server. The LO demo saves the files to C:\Inetpub\wwwroot\LO_Demo\media_assets. From within the ColdFusion <cfile> tag we can control the type of file that is uploaded and the location in which it is placed on the server. The cfile upload also checks that it is the correct file type and control the read/write access to the file once it is saved. The screen snap below illustrates the cfml code used to place a media file on the server.

```
1 <html>
2 <head> <title>Upload File</title></head>
3 <body>
4 <h2>Upload File</h2>
5
6 <cfile action="upload"
7     destination="C:\Inetpub\wwwroot\LO_Demo\media_assets"
8     nameConflict="MakeUnique"
9     fileField="Form.FiletoUpload"
10    accept="image/jpg, application/x-shockwave-flash">
11
12 Your media file uploaded successfully.
13
14 <p><a href="javascript:window.close();">Close Window</a></p>
15 </body>
16 </html>
```

Figure 27: ColdFusion cffile tag for uploading media assets

For further information on using the cfml upload tag, see the ColdFusion MX documentation on *Uploading files with the cffile tag* and the section on *Securing Applications*.

Inside the Workbench: Screen Level - Edit

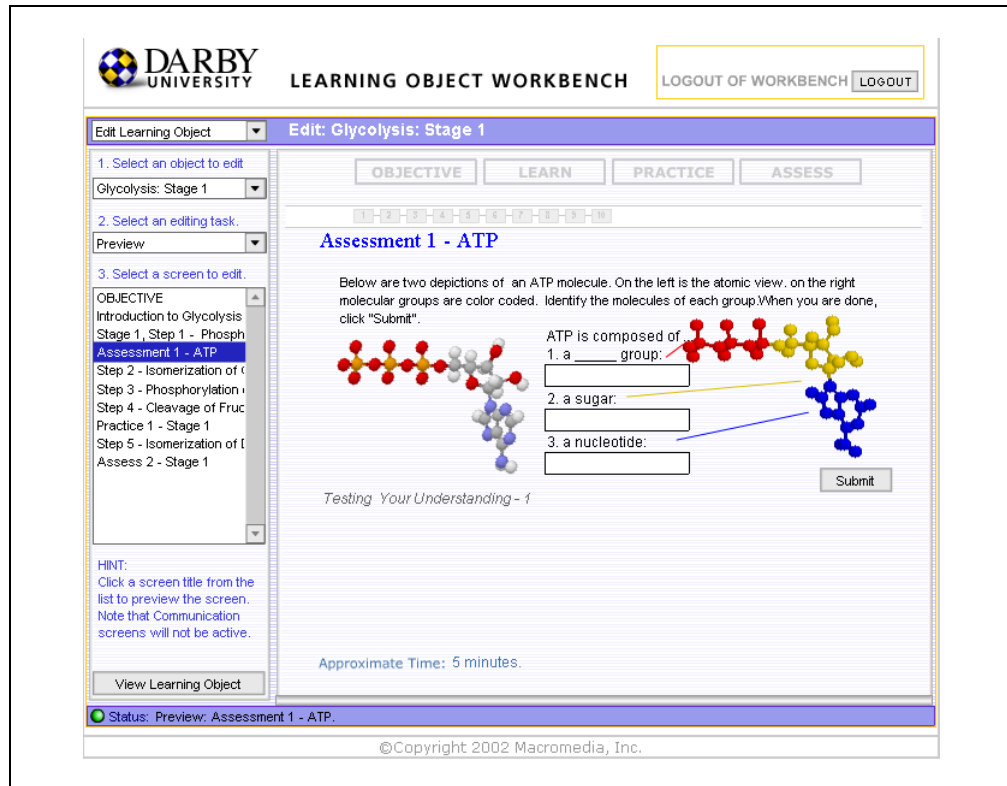


Figure 28: Editing a Learning Object in the Workbench

The Edit Pane locates users at the Screen Level. The drop-down box on the left of the user interface allows the user to select a specific Learning Object for editing. Once the object is selected the balance of functionality is revealed. There are basically two views in Edit: Preview and Forms. Preview allows users to see how the LO content will be presented while Forms view allows users to add, modify or delete LO content.

Editing Tasks - Preview and Delete Screens

The first task in the dropdown list is the 'View Learning Object'. Here users may preview Learning Objects by selecting from the list and clicking the 'View' button.

When the 'Delete' editing task is selected, the user is prompted to select a screen to edit. Once selected, the 'Delete' button is made available and when clicked, a warning dialog box prompts the user to confirm the action. If the user continues by clicking the 'Delete' button the screen is deleted, the popup closes, the list is reordered and refreshed.

If the user tries to 'Delete' the first screen – the Objective – the popup warns the user that by deleting that screen the entire object will be deleted from the database. If the user clicks 'Delete' the object is deleted and interface changes to the View Object list.

Form View - Add and Change Screens

Screen layouts are derived from templates. When you select "Add" from the editing task drop down window and a window slides open to reveal the screen templates.

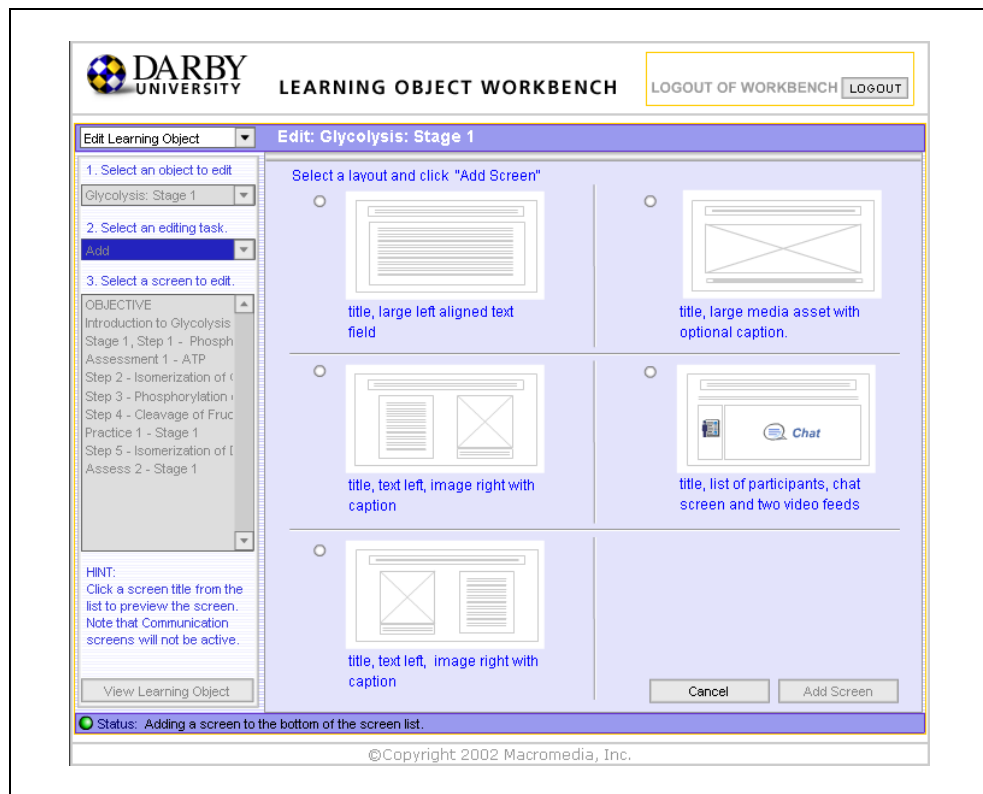


Figure 29: Screen layout templates.

The user selects an appropriate template and determines where to place it into the screen list. By default, the new screen will be added to the bottom of the list. Please notice that some of the main controls have been intentionally disabled in this view; 'Cancel' closes the forms pane and returns you to preview. When 'Add' is clicked the screen is added and the Change form is revealed.

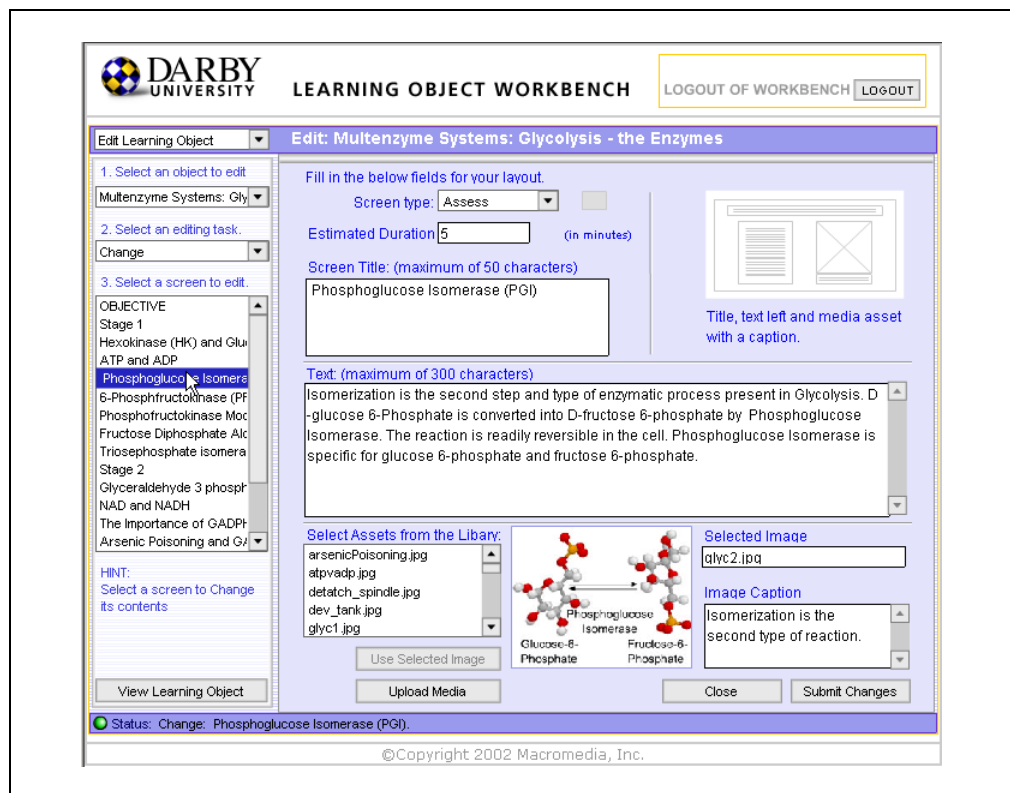


Figure 30: *Editing the content in the screen layout*

As a user clicks through the screen list the layout templates change depending on the layout of the page. 'Edit' allows change to all of the properties of a screen and view each screen in 'Change' mode.

Click 'Submit Changes' and when the submission is complete the Status bar indicates success and the screen information for the entire object is refreshed. Click "Close" to leave the forms mode.

It is possible to return to 'Add' by selecting it from the task dropdown. Selecting a task other than 'Add' will cause the form window to close and you will return to preview mode.

Users can preview Learning Object from most screens in edit mode to get a better feel for the end user experience.

Server/Database Communications

Edit is the entry to the Screen level. Let's summarize the types of server/database communication being used in the application. There are two types of methods being utilized: the *get* and the *set* functions.

Get functions query the database for particular data. This is defined by the functions in the ColdFusion Component. On receipt of the “recordset” (what the Server returns to the Client from the Database) the Flash MX Workbench UI extracts bits to populate components in the user interface. The Workbench may also use the recordset directly as a “Data Provider” for a component. Get methods are used to populate the dynamic lists used in the Workbench, i.e. Learning Object List and Screen List. These methods are called whenever the user performs a task or action that changes the list. Tasks are composed of actions.

Set functions make changes to the database by inserting, changing or deleting data. The Workbench passes an argument to the server, which performs the method via an SQL statement and passes the response to the Workbench. This is often a string or a Boolean value. The receipt of this response elicits a secondary call to get the new data and refresh the lists. These methods are called when the user performs a task that changes the data content or order, e.g. Reorder, Change.

The challenge is to determine which mode (environment) is best to perform these tasks. These fall into two basic categories: *preview* and *form*. **Preview** mode is where it is of use to see a screen in context with the full object. The Preview task is the most obvious example of this case, the other two are Reorder and Delete.

Form mode is dictated by a need to isolate the task. Isolation is determined by complexity. The Workbench screen tasks, Add and Change are perfect examples of two types of complexity. Add is a simple action with lots of possible choices and Change is a simple choice with lots of actions.

Enough theory – lets review the screen level. When users enter ‘Edit’ visible is a rather simple interface. The Learning Object is embedded into the viewing area and muted, and the instruction area has morphed into a control panel with one instruction – Select a Learning Object to Edit – and a combo box populated with a list of Learning objects (populated by a getter method) and the re-occurring ‘View’ (Learning Object) button. When the Learning Object is selected from the combo box more of the controls are revealed. An Objective screen for your selection is available in the view area and are presented with a Task List (combo box populated with an internal array) a Screen list and Hints for each task.

Preview is just that, move through the list box to see different screens. When you click on the items in the list box two global variables are redefined – Screen_ID and Screen_Layout.

Selecting ‘Delete’ prompts the user to then click a screen title to delete. When clicked the expected Message Box appears. If the first screen is select the user will delete the entire object and be taken out of Edit and directed back into View on the Object level. The message boxes are different depending on which is selected; a stern warning on screen one a more gentle alert on other screens. The severity of warning is shown in the icon. There is another consideration: if a screen is selected that is in the middle of the list then the list will need to be reordered.

Selecting 'Reorder' changes the control column. 'View Object' becomes 'Submit Object Order' and a 'Cancel' button appears. Two arrow buttons under the screen list appear and users are instructed to select an item and reposition it using the arrows. The user can preview the Learning Object and when satisfied may submit to the database. An update function then loops through the list box and resets the order. When the list is complete and the last response is sent the user is back in Preview and all controls have been reset.

Selecting either 'Add' or 'Change' obscures the embedded learning object with a sliding screen (its "roll bar" has been visible at the bottom of the view area in the Edit this entire time). The learning object behind the screen is made invisible. Users can move between 'Add' and 'Change' via the task list.

Adding a screen requires the user to select one of a series of layout templates. Once selected the user can determine where the screen will insert be inserted- the default is at the bottom of the list. The 'Add Screen' button does one of two things. If insert is not specified it invokes a simple add method. Otherwise it invokes the add method and when that is accomplished it reorders everything after the new screen. No screen can be added before the first because that is the Objective screen. When the server responds the 'Add' form is replaced by the 'Change' form for further editing.

Conclusion

The authors of the Learning Object Demo application see this release as marking the next step in the ongoing evolution of Learning Objects concepts and practice. By making the working source-code and documentation freely available to developers in the eLearning and higher education industries, Macromedia hopes to encourage wider debate and best-practices around the reusable Learning Object content model.

Next Steps

While the Learning Object Demo application has evolved, there yet remains features and considerations the developers would like to add that would further enhance the LO Demo app. These include:

- conformance with common eLearning standards like SCORM, AICC and others;
- integration into Learning Management Systems (LMSs);
- accessible Learning Object content for those with disabilities.

Macromedia would like to encourage developers and organizations that have implemented real-world examples of the Learning Object Demo application to contact the authors. We look forward to seeing quality examples of the Learning Object Demo in action.

Macromedia Learning Objects Development Center

Macromedia is committed to help businesses, educational institutions, and government agencies create personalized, relevant learning experiences. The Macromedia Learning Objects Development Center hosts a series of whitepapers, development assets, and implementation models that will help organizations leverage their technology investments to improve organizational and employee productivity and performance. Visit www.macromedia.com/go/objects to access the full set of resources available.

Download version 2 of the Learning Object Demo application from the Macromedia web site at:
http://download.macromedia.com/pub/solutions/downloads/elearning/org_objects_demo.zip.

Ann Gallenson is the founder of A Charles Design, Inc.; an interactive multimedia development and design consultancy. Ann is a user interface and product designer and developer whose career spans eighteen years and includes designs for educational courseware, commercial software, medical, video, and broadcast products. She is formally trained in science and mathematics and has a background in teaching, fine arts, and graphic design.

You can contact Ann at ann@acharlesdesign.com

Jay Heins is an eLearning consultant with a background in fine arts, video and electronic media, Jay provides direction in the appropriate use of emerging web technologies for Numen Communications. He manages all aspects of the studio's development process, including client relations, site architecture, programming, production, and visual design. His creative leadership provides Numen's work with a clear sense of structure and visual aesthetics. Jay has a B.F.A. in Fine Arts from the University of Ottawa in Ontario, Canada.

You can contact Jay at jay@numen.ca

Tanya Heins is the Sr. Product Manager eLearning and Higher Education at Macromedia, where she works on the company's eLearning and Higher Education product strategy. Over the course of Tanya's career she has focused on content design and development, eLearning architectures and the development of eLearning environments that adapt to a student's preferred learning style. Tanya Heins has a background in teaching, fine arts, and applied research in eLearning. She holds an M.F.A from the University of Victoria in British Columbia, and has nine years of professional interactive design and development experience, ranging from designing CD-ROMs and database-driven eLearning to custom development of award winning learner and content management systems.

You can contact Tanya at theins@macromedia.com