Adobe® LiveCycle® Mosaic ES2 Implementations

Creating intuitive, contextual workspaces for better decision-making

Introduction

Adobe LiveCycle Mosaic ES2 provides a framework to create composite rich internet applications (RIAs). Composite RIAs are web-centric applications combining multiple UI components that are assembled and displayed on the client during runtime, leveraging contextual information. LiveCycle Mosaic includes server components, browser and desktop clients, a Flex/ActionScript SDK and a JavaScript SDK.

LiveCycle Mosaic ES2 is targeted at enterprise RIAs and specifically suited to build user-centric applications, including workspaces for support and sales. Mosaic allows IT departments to distribute the development of various components of a composite application (e.g. customer records, issue reports, enrollment application) and then combine those components dynamically into a rich composite application. The Mosaic framework is highly customizable and allows IT to deliver applications that can be personalized and mashed-up by end-users as well as applications that are completely pre-configured by IT.

Tiles are a core concept of Mosaic applications. Tiles are application user interfaces developed in Flex or HTML, enhanced with the Mosaic client SDK and registered in the Mosaic catalog. As opposed to traditional enterprise portals, Mosaic fully leverages modern RIA technologies (Flex and Ajax), resulting in a lightweight server (based on Java and Spring) and a rich client SDK available in ActionScript and JavaScript. Tiles can connect to any backend server, as well as hosted applications, in the same way that applications run in a standalone browser. Just by adding simple script to the client code and without any changes to the server code, tiles can leverage the Mosaic framework to communicate with other tiles in a view, persist contextual data and trigger actions (e.g. launching of another tile). This client-centric concept makes it very easy to convert existing web application UI's into reusable Mosaic tiles.

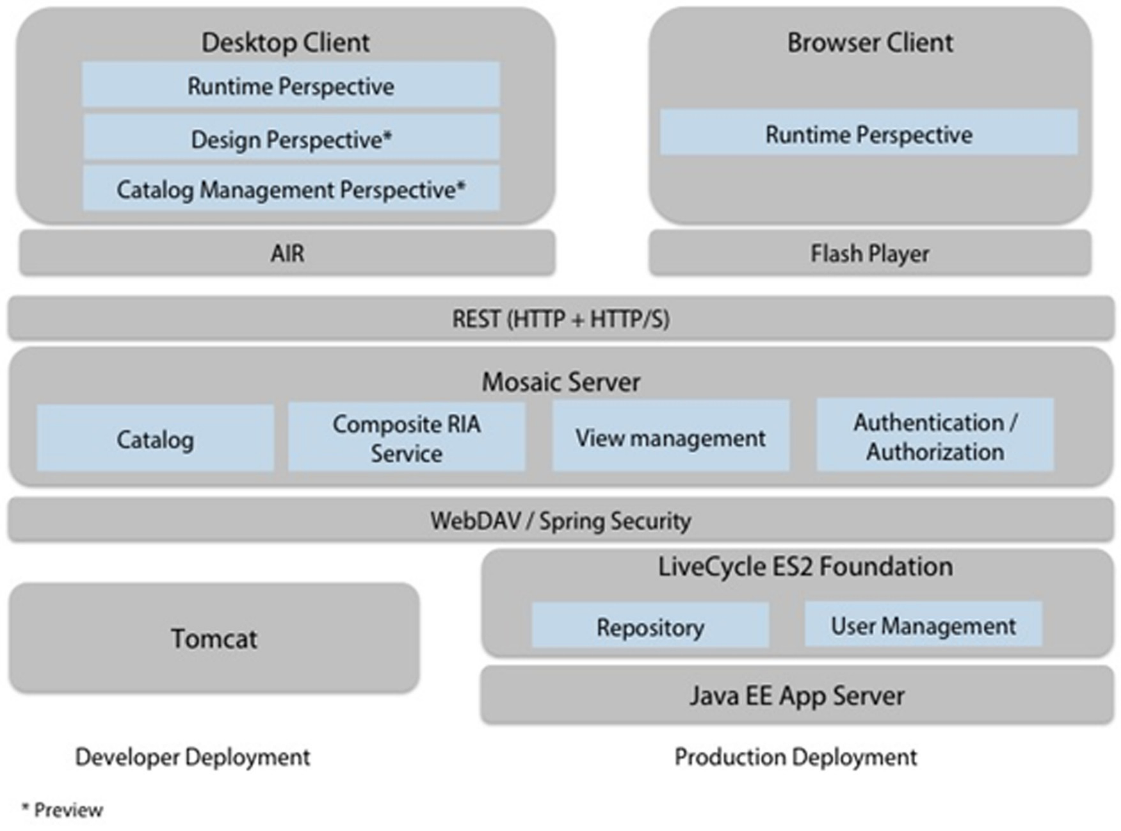
To minimize total cost of ownership (TCO), Mosaic integrates with existing enterprise software infrastructure. Most importantly it allows integration with the existing security infrastructure by integrating with existing authentication and authorization services. Mosaic assets can be stored in the out-of-the box repository or integrated via WebDAV with external repositories. Mosaic integrates seamlessly with other LiveCycle ES2 modules via the LiveCycle Foundation.

Table of contents

Architecture and deployment

The Mosaic server is implemented in Java using the Spring framework, and runs on top of Tomcat as well as other Java EE application servers. For a local developer deployment, there is a fully functional Mosaic standalone server that includes Tomcat. It can be unzipped onto a development computer (desktop, laptop or server) and be immediately ready to use. Alternatively it can also be deployed onto an existing Tomcat instance. For production environments we recommend deploying Mosaic on top of the LiveCycle ES2 foundation. LiveCycle ES2 is supported on IBM WebSphere, Oracle WebLogic and JBOSS application servers.

The LiveCycle foundation provides advanced user management capabilities, integration with LDAP, authentication and SSO infrastructure, as well as a robust repository. Detailed system requirements for server, development infrastructure and client can be found within the LiveCycle Mosaic ES2 Documentation.



The Mosaic server provides the following core services to run composite RIAs:

Catalog

The catalog service manages the composite RIA assets, specifically all the available tiles. The catalog contains both metadata about the tiles (e.g. Name, description, min/max size) as well as the SWF file. In the case of

HTML tiles it includes the URL to the actual web application.

Composite RIA service

This is the core runtime service of Mosaic. The composite RIA service gets the requests from the runtime and delivers the application XML. The composite RIA XML description is registered with this service.

View management

The view management service handles the persistence of views on the server. Once a user requests to save a view, it is stored in the repository. This service also retrieves stored views from the repository and serves it to the client runtime on request.

Authentication and authorization

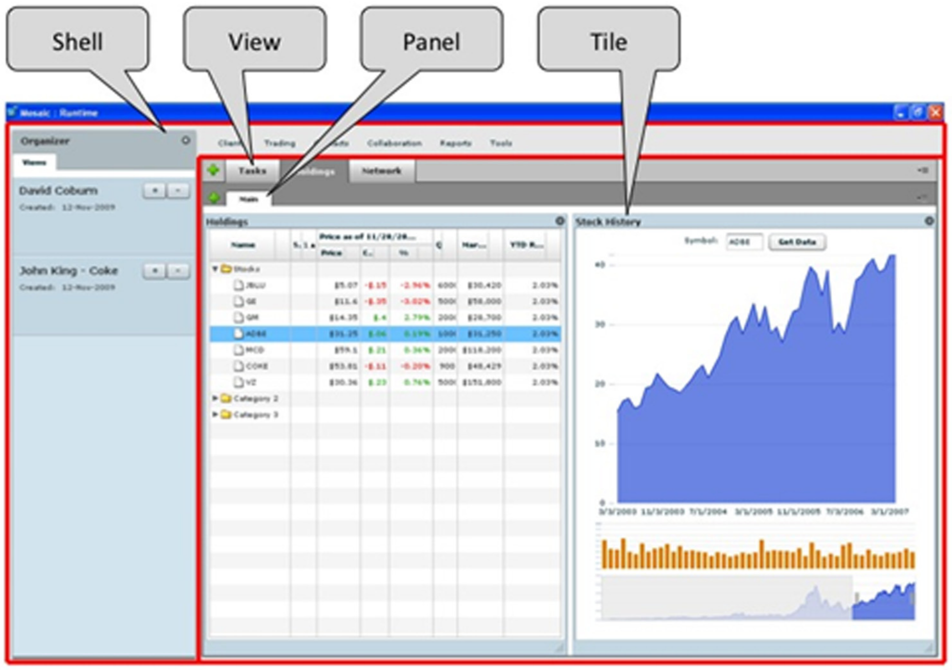
These service components handle Mosaic security. Based on the Spring security framework, they manage the user authentication. An XACML based service handles authorization policies.

Users can access Mosaic applications both via a web browser and as a dedicated desktop client. The clients communicate with the server using REST over HTTP or HTTP/S (configured on the server). The browser client requires the Flash Player 10 plug-in. Mosaic applications run within the Flash Player on the client. The desktop client is based on Adobe AIR. It can be deployed on Windows, Mac OS and Linux.

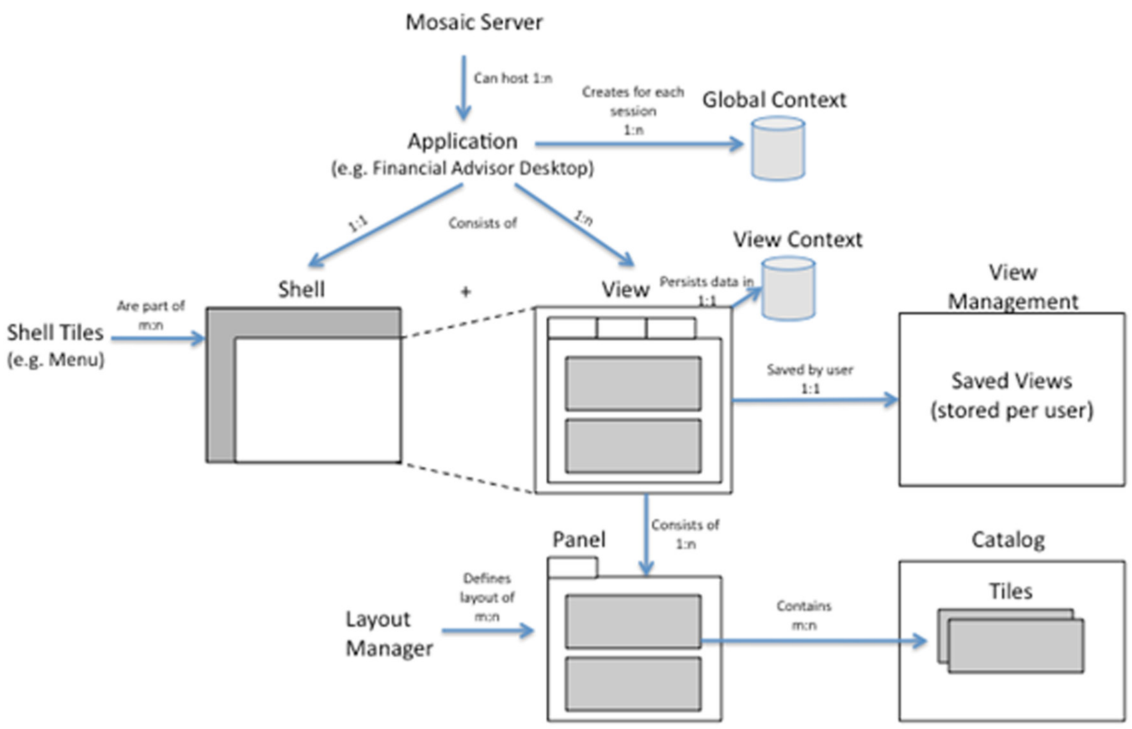
Developing Mosaic applications

Before creating Mosaic composite RIAs, it is important to understand the different framework objects and how they relate to each other.

In a nutshell, Mosaic assembles multiple UI components (called tiles) together into a dynamic user-centric application. Tiles are the "puzzle pieces" of a Mosaic application. They are reusable application UI modules that are stored and organized in the Mosaic catalog. Tiles can be created in Adobe Flex or HTML/JavaScript.



A Mosaic server can host one or multiple Mosaic applications. A Mosaic application is described in XML (for more info on the Mosaic application XML, see the [XML Application Reference](#)). The Mosaic application XML is deployed on the Mosaic server and used during runtime to dynamically assemble and present the application to the end-user. The two core elements of a Mosaic application are the shell and one or multiple views. The shell includes components that are always available when a user is working with the application.



The shell could include a menu tile, company logo, search tile or similar common functionality. Shell positioning and layout is dynamic and can be defined in the application XML. Within the shell the user can access one or multiple views. A view packages one or multiple tiles that are required to perform a specific task (e.g. a customer overview view would include tiles showing information of a specific customers from different back-end systems and the web). Furthermore each instance of a view is associated with a context (for example, the customer name and ID). The context is a Mosaic framework-managed memory object that is accessible to all tiles in a view (in case of the global context it is accessible to all tiles in the application). The end-user can customize the view (e.g. by adding or deleting tiles) or change the context and then save the modified view for later usage. A view can have multiple panels to help organize the layout of the tiles. A panel is typically represented with a tab in a customized Mosaic view. A panel arranges multiple tiles together in a predefined layout (see the Layout element for more information on Mosaic layout types).

Developing tiles

Since tiles are the fundamental building blocks of a Mosaic application, the first step in a Mosaic project is to develop the tiles and register them in the catalog. Mosaic comes with a number of sample tiles with source code to accelerate custom tile development. Mosaic recognizes three types of tiles: Flex application tiles (SWF), Flex module tiles (SWC) and HTML tiles.

Flex tiles (sub-applications or modules)

Flex tiles can be developed in Adobe Flash Builder and existing Flex applications can be easily converted into tiles in three steps:

- add the Mosaic TileSDK.swc into the Flex library folder,
- change the namespace (mx:Application to MX:Tile or mx:Module to MX:ModuleTile), and
- add the namespace "com.adobe.mosaic.core" to the root MXML element.

At this point, the tile can be enhanced with specific Mosaic runtime capabilities (see Tile APIs) and uploaded into the catalog.

Application tiles versus module tiles

Application tiles provide the most flexibility. They are compiled as SWF and can run both within Mosaic and as standalone applications. Application tiles also can be compiled with different versions of the Flex SDK (3.3 and higher), providing the largest degree of independence between different application development teams. The flexibility comes with the price of a larger memory footprint since each tile requires the Flex framework to be loaded into memory.

Module tiles need to be compiled with the same version of the SDK the Mosaic framework is compiled with (in Mosaic 9.0 that is Flex SDK 3.4.0).

It is generally recommended to use application tiles whenever possible.

HTML tiles

An HTML-based application can be registered in the Mosaic catalog without any modifications. In the browser client, this tile will be loaded as an iFrame within the overall Mosaic application. In this case, the HTML is rendered by the browser and will support any features (e.g. plug-ins, including Flash Player and Adobe Reader) that the browser supports. The Mosaic framework handles all the underlying complexities with running HTML in a Flash container, including layering (z-order determination) and communication between tiles.

In the desktop client, the AIR web browser engine, based on WebKit, renders the HTML tile. Therefore the web site needs to be compatible with WebKit (also used in Safari and Google Chrome). AIR does not support browser plug-ins (except the embedded Flash Player and Adobe Reader).

To provide tighter integration between an HTML tile and the overall Mosaic application, a range of tile APIs are available in JavaScript (see Tile APIs). In order to leverage these APIs, a link to the mosaicBridge.js library needs to be embedded into the HTML. All APIs are client-side API's and don't require modification of the business logic on the server.

An HTML tile can also participate in the overall context of the Mosaic application without code changes. If the web application supports passing of parameters via attributes in the URL, those can be passed dynamically with data from the Mosaic context during runtime (e.g. the symbol of stock to be displayed in Google finance).

Tile APIs

Many of the tile APIs are available in both ActionScript and JavaScript and the scope of API functionality may extend with each release of LiveCycle Mosaic. LiveCycle Mosaic ES2 (9.0) includes the following API functionality:

Inter-tile communication

The context is a runtime object that is accessible to tiles to set or get context attributes. There are two context instances: A view context that is available to all tiles in a specific view instance and a global context that is available to all tiles in an application. When a user saves a view, the view context is saved with it, so it can not only be used to pass data between tiles but also to re-instantiate tiles when saved views are being loaded by the user. The API allows tiles to set attributes in the view context or global context, register context attribute watchers (monitoring if an attribute in the context changes), and get attributes from the context.

Sending message between tiles. Messages are not persisted. The API allows tiles to send a message and add a message listener, subscribing to messages by namespace and name.

Navigation and user interface actions

- Get the name of the currently active view and panel.
- Add a new view in the application.
- Add a new panel to a view.
- Get a tile from the catalog and add it to the current panel.

Saving and loading custom views

- Request a save view action (activates user interface to save a view).
- Save view programmatically (without user interaction).
- Get a list of all saved views of current user.
- Load a saved view back into the application.
- Delete a saved view.

User information

- Get full name of current user.
- Get login handle of current user.
- Get SAML assertion of the current user (enables Single-Sign-On).

Styling

- Get stylesheet(s) referenced by application. This allows developers to retrieve the global stylesheet(s) of an application and apply it to the style of a tile.

Catalog

- Return name of catalogs used by application.
- Get catalog of a specific tile.
- Get a specific catalog.

Other APIs

- Navigate and manipulate the application XML programmatically.
- Get label of tile.
- Get list of all views in an application.
- Get list of panels in a view.
- Get parent view of panel.
- Get list of all tiles in panel.
- Get parent panel of tile.
- Get parent view of tile.

Best practices for inter-tile communication and using persisting context

Passing information between tiles can be accomplished in multiple ways, including using the view context, the global context and messaging. In many cases, the best option is to use the view context. When publishing data to the view context, all other tiles in the view can pick up the information. Even when the user adds a tile to the view later that tile can pick up the existing context and adjust itself to show relevant information to the user. The view context is constrained to sharing of data between tiles in the same view. Since users could have multiple views open with the same tiles but different contexts (e.g. support views showing support information for different customers) that is the desired behavior in many cases. Sometimes, however, it is necessary to pass data to all tiles in all views. An example of this is sharing user-specific configuration data. A user configuration tile running in the shell might allow a user to set application-wide properties. The global context can be used to pass these properties to all tiles. In general, tiles in the shell can only use the global context, since they are not associated with a specific view.

In certain cases you might not want to have data that is exchanged between tiles persisted in memory or on disk. The view context is stored in memory and persisted alongside with the tile and layout information when a user saves a view on the server. The global context is only stored in memory throughout a user session. The third option to share data between tiles is to use the messaging API's. This option is specifically recommended for use cases that require a transient exchange of data between tiles. Using the messaging APIs, tiles can send information to any other tiles running in the application (not restricted to a view) that are listening to a specific namespace and subject name.

Leveraging the view context can add significant usability and productivity boosts to your application. To fully leverage the context, a couple of guidelines should be considered when developing tiles:

Always check the current context on initialization of a new tile. When a new tile is being added to a view the context might be already set with information that could be useful to the new tile. Make sure that you consume the content properly and take action in your tile UI (e.g. if the context includes a customer ID and you load a tile with a list of customers, make sure that the customer currently set in the context is activated and highlighted in your list). Also keep in mind that users might be able to save the view with context (assuming that it is enabled in the composite RIA). In that case you also want to reload the context into a tile when the user opens the view. Also ensure your code is listening for changes in the context so your tile can react to those change (e.g. retrieve and display detailed information about the current customer in context).

Properly document the inter-tile communication you are using in your code, including the names of the attributes, the API you are using, and when the API is triggered. This information needs to be coordinated between all the teams that develop Mosaic tiles. Currently this cannot be automatically discovered in Mosaic. There are plans to possibly introduce a feature to handle the automatic inspection of inter-tile communication features of a tile in a future Mosaic release.

Some use cases require loading initial configuration information into a tile. In Mosaic this information can be stored within the Mosaic catalog in the "/Resources" directory. The information (e.g. an XML file) is then available to all tiles via a URL. Since the catalog can be easily moved from development to test and production server, this ensures that the configuration information is always available in the same relative path.

Communication between tiles and backend

Mosaic does not facilitate the communication between the tiles and backend application logic. Tiles can leverage the same methods to communicate with backends that are used today for Flex or HTML applications. For Flex tiles we recommend leveraging LiveCycle Data Services to establish a robust and highly performing connection between the tile and the server application logic. Tiles can also leverage BlazeDS or direct Web Services or REST communication.

Additional information on tile API's:

- The LiveCycle Mosaic ES2 ActionScript Language Reference
- The LiveCycle Mosaic ES JavaScript Language Reference

Managing tiles in the Mosaic catalog

All tiles are managed on the Mosaic server in the catalog. The catalog keeps track of all available tiles and their metadata. Tiles can be uploaded and managed in the catalog using an XML description and Ant. As part of the metadata, the user can configure the initial, maximum and minimum width and height of the tile when loaded into a view. Additional metadata includes the tile name, label, category, tags, description and icon, improving manageability and searchability of the catalog. A preview version of a visual catalog management tool is available in the Mosaic desktop client. A full version of the visual catalog manager, as well as a catalog browser tile for end-users, will be available in a future version of Mosaic.

Designing composite RIAs

Mosaic applications are defined in XML and dynamically assembled during runtime on the client. The application XML can be created in any text or XML editor, or even programmatically by another application. The Mosaic 9.0 desktop client includes an early preview version of a Mosaic application design tool that lets you edit and save the application XML in the desktop client. A visual design tool may be available in the future.

The first section in the application XML references one or multiple catalogs used by the application. In this section the application developer specifies the URIs and names of the catalogs that store the tiles used in the application. Tiles in an application could be retrieved from multiple catalogs.

The shell is the outer frame of the application that is consistent, regardless of the specific view the user is working with. The shell typically includes a menu and the view organizer, but it is completely configurable for custom applications. The shell supports different layout managers, and can be configured in different ways (e.g. just top bar, side bar, bottom bar, L-shape, etc.). The tiles that should show up in the shell need to be configured and positioned. Generally it is most efficient to start with a shell from one of the application samples provided and then customize it.

An Adobe-provided out-of-the-box component for the shell is the view organizer. The organizer can be activated in the shell by adding <view:Organizer height="x" width="y"/> to the application XML. The organizer allows users to access and load their saved custom views. The APIs used by the view organizer are also part of the runtime SDK allowing custom implementation of tiles that can manage custom views.

The other major aspect of Mosaic application design is to configure the default application views. Fundamentally the view is the container for the tiles. Tiles can be located on one or multiple panels, and each panel can have a different layout manager. There are both layout options for fixed positioning (user cannot modify tile position) and dynamic positioning (user can re-arrange tiles) including absolute layout, column layout, dynamic column layout, dynamic row layout, flow layout, horizontal layout, row layout and vertical layout. Using containers, multiple layout models can be combined on one panel. Additionally the panel XML can define the name of the panel, if tiles in the panel have chrome or are displayed without chrome and if the user can delete them or not.

Deploying composite RIAs

Mosaic applications are deployed using Apache Ant, a software tool for automating software build processes. A sample Ant script to deploy Mosaic catalogs and applications is provided with Mosaic. Once the catalog XML description, SWF files for tiles and application XML are loaded in the proper folders, Ant will deploy them on the server. This procedure works exactly the same regardless if the development server is running on Tomcat or the production server is running on the LiveCycle foundation in a Java EE server. Once the application is deployed, it is accessible in the desktop client and browser client.

This standard deployment model is easy to automate via scripting, and can be used to move Mosaic application and catalog deployments from development to test and then to production servers.

Browser client

Mosaic applications can be directly accessed via an application specific URL such as http://server:8080/mosaic/#/applications/<application_name> in the browser. If not already authenticated, the user will be required to log in. The Mosaic application itself runs within the Flash Player (Flash Player 10 or higher).

Desktop client

The Mosaic desktop client runs on Adobe AIR. Users can install the runtime client on their desktop and—based on their permissions—access different perspectives:

Runtime perspective

The runtime perspective allows users to run Mosaic composite applications straight from their desktop. A separate launch window displays a list of all applications available to the specific user.

Design perspective

The design perspective is available as a preview in Mosaic ES2. It allows developers to create and modify Mosaic applications. The preview version enables developers to change the composite application XML within the client and re-deploy it to the server. A visual design may be available in a future version.

Catalog management perspective

The catalog management perspective is the administration tool for all Mosaic catalogs. It allows users to visually browse and update the catalog.

End-user functionality

Mosaic provides a range of out-of-the box end-user functionality that can be enabled in custom Mosaic applications. All of the functionality can be configured (or completely disabled) by the IT department.

Users can add new views and new panels to an existing application. Tiles can be added to existing panels or to new panels. Furthermore, users can rearrange or delete tiles from panels. A user can save the customized view with a new name. The saved view immediately shows up in the view manager, a shell component that allows users to browse saved views. Saved views can later be opened again in the applications or can be deleted if no longer needed.

Authentication, authorization and single-sign-on

Authentication is the process of confirming the identity of the user. Out-of-the-box, Mosaic supports authentication based on username and password using HTTP Basic Authentication and Web Form Login. Users can be managed directly in the LiveCycle user management system, which also allows synchronization with external user management systems such as LDAP or custom databases. Since the Mosaic authentication is based on the Spring Security framework, administrators can customize the security.xml file and use any Spring Security mechanism. The Spring Security framework provides integration with NTLM/Active Directory, LDAP Directories, major SSO vendors, OpenID, as well as simpler arrangements such as JDBC database with username and passwords and built-in Application Server mechanisms.

Authorization is the function of specifying access rights to individual resources. Authorization utilizes the established identity to make decisions about what the user is allowed to access and what types of access are permitted. Authorizations are configured in the security.xml and mosaic_http_access_policy.xml files. Beyond standard role-based authentication, Mosaic also provides a policy decision engine based on XACML. XACML allows the configuration of complex policy rules including roles, user attributes, date ranges, and time of the day.

The Mosaic client also respects server session timeouts. The server administrator can set session timeouts to any time range or inactivity period. The client handles this by performing a status ping every minute. If the ping results in a redirection to the login page, the client will require the user to re-authenticate before continuing.

The objective of single-sign-on (SSO) is to make users more productive and eliminate separate login displays to different applications. Mosaic supports SSO, leveraging existing enterprise SSO solutions as well as SSO with other LiveCycle services. Mosaic offers tile developers the ability to utilize the current user's Mosaic user context to log in to other enterprise systems. This is achieved by providing an API in the Tile SDK that returns a Base64-encoded SAML assertion available from the configured authentication provider.

The enterprise world is full of proprietary solutions for solving the single sign-on problem, and it follows that there are many different ways Mosaic can be configured to work into those various SSO solutions. Using the Spring Security framework as the basis for securing Mosaic opens many options.

The Spring Security framework provides the ability to integrate with Kerberos/SPNEGO. This configuration requires that the client runs only on browsers that support and are configured for Kerberos/SPNEGO (i.e. Internet Explorer, Firefox, Safari).

The Spring Security framework also supports the use of HTTP-based SSO solutions, such as CA SiteMinder. This configuration is known as the "PreAuthentication" filter (i.e. the user was authenticated by some other web entity prior to reaching the servlet). This filter can be configured to check for the standard HTTP headers and/or cookies provided for this type of SSO, such as SM_USER and SM_SERVERSESSIONID in the case of SiteMinder.

Finally, the Spring Security framework provides JAAS (Java Authentication and Authorization Service) integration. JAAS is a long-time standard in the Java world that is well supported across application servers. This integration is very powerful because many systems have a JAAS interface. The Oracle Application Server SSO provides out-of-the-box JAAS interfaces for integrating web applications to Oracle's SSO solution. Many enterprises may find that there is an existing JAAS interface that they have already written for integrating into a legacy authentication system. This configuration is known as the "JaasAuthenticationProvider". There is a commented section in the Mosaic security.xml file that provides a general idea for how one would integrate a JAAS provider.

Glossary

Catalog

A catalog stores tiles, view templates, shells, panels, style sheets and other components of a Mosaic application.

Context

A context is a data store that allows sharing of data between multiple tiles. It can include both simple and complex objects. Tiles can both get data from and store data in the context. There are two different types of context. The view context is the context to share data between tiles in one view (this context should be used by default for most cases). A global context allows sharing of data between all tiles in all views of a Mosaic application (to be used for global preferences and similar use cases).

Layout

A layout controls the arrangement of tiles in a panel. You can configure layouts when you design a Mosaic application. Some layouts allow users to change the arrangement of tiles in a panel.

Mosaic application

A shell that can hold one or multiple views defines a Mosaic application. A Mosaic application can be accessed via a specific URL in the browser or via the Mosaic desktop client. Application examples include: Financial Advisor Desktop, Support Desktop, Sales Workplace, and Case Worker Desktop.

Mosaic browser client

Users can access Mosaic applications using a web browser with Adobe Flash Player installed.

Mosaic desktop client

The Mosaic desktop client is an Adobe AIR application that allows users to access Mosaic applications (runtime perspective). It also includes the design perspective and catalog manager perspective.

Mosaic server

The Mosaic server includes the catalog that stores application components, handles persistence of users' custom views and identity/entitlement management. The server is implemented in Java using the Spring framework. The server can run on Apache Tomcat (for development environments) and on the Adobe LiveCycle ES2 Foundation (J2EE-based app servers), which is recommend for production environments.

Panel

A panel can consist of one or multiple tiles. Panels visually group together tiles in a view. You can group tiles in panels by task or some other category and configure a unique layout for each panel. You can allow users to add and remove panels from a view.

Shell

A shell and one or multiple views running within the shell define the user interface of a Mosaic application. The shell layout is configurable for each Mosaic application. A shell typically contains key functions that need to be available to end users throughout using a specific Mosaic application including a shell menu, view organizer and other custom tiles.

Shell menu

The shell menu is a specific tile located in the shell of a Mosaic application. It organizes and lists UI components (e.g. pre-configured views or tiles) that are available to a user. For example, the user can add a new tile to an active panel by selecting it from the shell menu. Adobe provides a sample Flex shell menu that can be used as a starting point to create custom shell menus.

Tile

Tiles are the building blocks of Mosaic applications. Typically a tile encapsulates a rich internet application or the user interface for a backend application. A tile can be implemented as a Flex application (SWF), a Flex module (SWC) or HTML application. Using the tile SDK developers can leverage Mosaic framework capabilities (like the shared context) in their tiles. Tiles can be used both in views (grouped in panels) and in the shell.

Tile SDK

The tile SDK comprises JavaScript and Actionscript libraries that developers can use to create Mosaic tiles and access the Mosaic runtime APIs. The tile SDK also includes the API documentation.

View

A Mosaic view contains one or more panels in which tiles are laid out to suit the user's needs. Mosaic applications contain one or more views. Users can save views on the Mosaic server and retrieve them using the view organizer, which can be enabled or disabled for each Mosaic application.

View organizer

The view organizer is an optional Adobe-delivered UI component in the shell. It is used to access views that were previously saved by the user and load them back into the Mosaic application. Users can only see the views in the view organizer that they saved (they cannot see views of other users).

For more information and additional product details:
http://www.adobe.com/devnet/livecycle/

We welcome your comments. Please send any feedback on this technical guide to LCES-Feedback@adobe.com.

Adobe®

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704
USA
www.adobe.com