1-1 What is Web Dynpro Security

1-1-1 Prerequisites
This unit is not intended to teach the basics of application security. Participants should have a basic understanding of user authorization and authentication as applied to applications running within a middle tier application server.

1-1-2 Introduction
Web Dynpro (WDP) is a tool for developing the presentation layer of applications. As such, only the authorization side of security can be addressed. However, it is seldom that a WDP application is built that does not access a backend system, where authentication becomes an issue. For example, a WDP application that uses an Adaptive RFC model will have to deal with R/3 authentication and authorization. Although WDP can be used as a standalone application, it is intended to be used within the context of the Enterprise Portal. Therefore, in its current implementation, WDP does not provide any configuration tools for declaratively setting authorization and authentication. For WDP applications running within the Portal framework, Portal security features can be leveraged for this purpose, as shown in the diagram below.

For standalone WDP applications, security must be “programmed into” the application. However, the Enterprise Portal only covers a small part of the security landscape for applications running within its framework. The Portal’s framework can be configured to authenticate and authorize access to applications. The URL to access WDP applications running within an iView can be secured based on a user’s credentials. However, finer granularity of security must still be “programmed into” the WDP application. For example, if the developer of a WDP application needs to restrict access to updating
information in a view, the programmer must write the code to check the user’s credentials and take the appropriate steps in the event of unauthorized access.

This unit shall cover WDP security within the context of WDP applications running within the Enterprise Portal. With regard to authentication, only user login will be discussed in this unit. More complex issues such as single-sign-on (SSO), where the user’s authentication credentials are used to secure the WDP application’s access to backend systems, shall not be discussed in this unit. This is a complex issue that is beyond the scope of this unit and will be addressed in subsequent classes dealing with more advanced WDP features.

This unit shall cover WDP user authorization. This includes both declarative and programmatic user authorization. In regard to declarative authorization, only the mechanics shall be discussed here because, assuming the WDP application is running within the Portal, configuration is entirely a Portal issue and will be covered in the unit on Portal security. The exercises in this unit shall address programming for WDP developers who must secure application data at a finer level of granularity than provided by the features of declarative Portal authorization.

1-2 Basics of Web Dynpro Platform

It is important to understand something about the Web Dynpro (WDP) platform and implementation to gain a good foundation for understanding the WDP security model. The WDP platform consists of several services based on a set of platform independent Java interfaces called the Server Abstraction Layer (SAL). WDP has its own runtime and is essentially a proprietary J2EE Container within the SAP J2EE Engine. All SAL services are contained in a Java package named: com.sap.tc.webdynpro.services.sal. The SAL exposes APIs for use by both the WDP runtime and its applications. The API is divided into two parts under the “sal” Java package: The “api” package is intended for use by applications and the “core” package is intended for exclusive use by the WDP runtime. The WDP services under the SAL package are listed in the table below. Only the “um” package will be the subject of this unit.

<table>
<thead>
<tr>
<th>Service</th>
<th>Sub-package under com.sap.tc.webdynpro.services.sal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>deployment</td>
<td>Provides deployment information of Web Dynpro applications.</td>
</tr>
<tr>
<td>Configuration</td>
<td>config</td>
<td>Read access to configuration data.</td>
</tr>
<tr>
<td>User management</td>
<td>um</td>
<td>Provides user authentication, authorization, roles, user profile.</td>
</tr>
<tr>
<td>Internationalization</td>
<td>localization</td>
<td>Supports handling of locale dependent resources.</td>
</tr>
<tr>
<td>URL generation</td>
<td>url</td>
<td>Generation of URLs to Web Dynpro applications and Web resources.</td>
</tr>
<tr>
<td>Runtime repository</td>
<td>repository</td>
<td>Read/write access to runtime metadata.</td>
</tr>
<tr>
<td>System landscape</td>
<td>sal</td>
<td>Access to the system landscape directory.</td>
</tr>
<tr>
<td>directory</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Context adapter</td>
<td>adapter</td>
<td>Provides methods to access request parameters.</td>
</tr>
<tr>
<td>Look and Feel</td>
<td>leaf</td>
<td>Provides methods to get access to the defined look and feel</td>
</tr>
<tr>
<td>Administration</td>
<td>admin</td>
<td>Provides system information helpful for system administration.</td>
</tr>
</tbody>
</table>
Basics of SAP User Management

User management within the SAP® Web AS 6.40 is provided by the User Management Engine (UME). The UME runs as a service within the SAP J2EE Engine and includes an API for programming and GUIs for administration of user management functions.

UME is the “glue” that ties together user management throughout the entire SAP NetWeaver™ Platform. It is a fully configurable and programmable interface for both the ABAP and J2EE runtime stacks. The basic categories of UME security are:

- **Authorization:** What is this user allowed to do?
- **Authentication:** Which user is logged in?
- **Profile:** What is the user’s name, address, etc.

In addition, UME can be mapped to standards such as the J2EE Security model and provides options for persistence including implementation over LDAP. The diagram below shows the basic relationships between the UME and various NetWeaver components.
Both Web Dynpro and Enterprise Portal have their own thin layer of Java packages (com.sap.tc.webdynpro.services.sal.um and com.sapportals.portal.prt) for access to UME objects and functionality. Once UME objects are obtained, full access to the UME programming model is available.

For security of J2EE components such as servlets, JSPs, and EJBs, the J2EE security model must be used. The Java security model is a generalized security model designed to provide interoperability with other programming languages and enterprise systems. To facilitate integration with legacy security models, J2EE provides the Java Authentication and Authorization Service (JAAS); a mechanism for other security models to plug seamlessly into J2EE security. For example, once JASS is configured to work with UME, standard J2EE mechanisms can be used to administer the security of servlets, JSPs and EJBs. “Under the blankets,” J2EE administration functions are being mapped to UME and are being managed by the UME persistence mechanism. The UME security model is similar to J2EE. This is often difficult to see because the terminology differs between these models and often conceals the actual principles that make these similar. No attempt will be made to compare UME with the J2EE security model because WDP application developers will always utilize the UME model when programming security. However, when accessing backend systems where SSO is enabled, developers must explicitly catch authorization and authentication exceptions that may occur and propagate these to the GUI using appropriate popup windows and recovery logic. The diagram below illustrates this principle. However, as noted in Section 1-1-2, authentication issues in WDP applications, such as SSO, shall be addressed in a subsequent class dealing with more advanced WDP features.
In the above diagram, the WDP application uses a Web Services model to call an EJB. The EJB in turn makes a call to an R/3 BAPI. In the event that the user’s credentials do not allow access to the BAPI, an exception is thrown in the EJB and is propagated back to the View Controller of the WDP application where the Web Service call was initiated. The WDP developer catches the Web Services interface exception and displays an appropriate warning message in a popup window. If no exception occurs, the WDP plug is fired to display the view to update the employee record.

1-4 Authorization of WDP Applications

WDP uses the SAP UME for user authorization. The primary objects of UME authorization are users, groups, and roles. Secondary objects used in authorization are permissions and actions.

1-4-1 Permission and Actions

Internally, UME uses Java Permissions for access control. A permission object consists of a target ("On whom will the action controlled by this permission be performed?") and an action ("What will be performed on the target, if so allowed by this permission?").

Java Permissions are programmatic but are also referenced in the declarative part of UME. All Java Permissions inherit from the java.security.Permission base class. There are many built-in permissions in the J2EE platform such as FilePermission, SocketPermission, etc. Generally, WDP application developers will create their own permission class that inherits from a base UME permission class (that eventually inherits from the java.security.Permission base class). However, there is nothing within the framework that requires application developers to create unique permission classes for each application. These can be developed and reused for various applications that need a similar permissions structure. For example, a general purpose DocumentPermission class may be useful for many applications.

The UME permission classes provide various levels of sophistication for authorization management. Because the subject of this unit is WDP security, only the most basic type of permission object will be explored here. More sophisticated uses of UME shall be covered in a class specifically dedicated to the features of UME.

A Permission is basically a software class that is programmed to provide various checks on declarative Actions. As long as the Actions are generic in nature, a given Permission class may be used with multiple applications. For example, a “CREATE” Action may be applicable to many types of applications. As long as the Permission object is only concerned with a user’s “right” to create “something” within an application, the same Permission class can be reused. On the other hand, if the user has the right to “CREATE” something in the given application, but that “something” restricts the user to be at least 18 years old, another permissions class must be developed that contains code to access and check the user’s age. The relationship between Permissions and Actions is illustrated below.
Actions are declarative in nature. Actions are declared and assigned to a permissions class using an XML file that can be generated manually or by some GUI configuration tool. The diagram above shows four Action objects declared in an XML file of an application. These reference a Permission class that can be instantiated at runtime for a given Action to generate a Permission for the application.

In the simplest case, where access to an application is limited to a certain class of individuals such as managers, a single Action can be declared that effectively represents a class of users instead of a specific action by a user. Using this approach,
the security features of the Enterprise Portal\(^1\) can control application access declaratively (see the diagram below).

In the diagram above, a simple `DocumentPermission` class and an `Action` named `Manager` are created. The Action is created by adding it to a security XML file that is included in the EAR file of the `EmployeeAdmin` application. The Action is linked to the DocumentPermission class by a reference in the XML file. When the application is deployed, the UME configuration database is updated with the `EmployeeAdmin.Manager` Action. Note that an Action is always associated with an application. Subsequently, the Action will be available to a system administrator as part of configuring user access to the application. Although the diagram above shows a one-to-one correspondence between an Action, a Permission, and an application, in more complex scenarios, multiple Actions can be assigned to a Permission and multiple Permissions can be assigned to an application. For example, `DocumentPermission` can allow or disallow Actions for viewing, updating, and creating employee documents such as performance reviews. The EmployeeAdmin application may have another permission named `SalaryAdminPermission` with its own set of Actions.

---

\(^1\) Note: As explained in section 1-1-2, standalone WDP applications must program all authorization checks in the application. For those running within the Enterprise Portal, basic authorization can be configured declaratively.
Declarative Control of Application Access

Deriving Permissions and Actions for an application is half the effort of authorization. First, the application must be configured either declaratively or programmatically to restrict access to users based on their allowed Permissions (e.g., it would not be wise to allow all employees to access each other’s salary information). Second, users that are allowed access to the application must be configured so they are assigned the required permissions to access the application (e.g., managers of the employee must be allowed to administer the employee’s performance review, etc.).

To accomplish these tasks, the system administrator may declaratively assign Permissions for the application and/or, the application developer may write code to check the Permissions of users accessing the application. In the latter case, the developer usually is programming finer access control than is available via the configuration tools used by the system administrator. For example, perhaps the application is available for viewing by HR Managers but only Engineering Managers have “create” and “update” permissions. If the configuration tools used by the system administrator can only limit access to the application as a whole (e.g., either the user has permission to access the application or not), then the application programmer will have to write code to provide finer granularity of access control. In the example shown in the previous diagram, we are assuming a very simple level of authorization control. In this case the Action, EmployeeAdmin.Manager, is intended to be configured declaratively by a system administrator. The only level of access control provided by the application is for users that are given the EmployAdmin.Manager Action—only these users will have permission to access the application. We can analogize this scenario as a person having a badge to gain access to a building—either the person is allowed access the building or is not. The diagram below illustrates this simple scenario.
In the above diagram, the system administrator uses a configuration tool to add access control to the EmployeeAdmin application. This type of access control is entirely declarative and is limited in scope to simply allow access or not to allow access. For WDP applications running within the Enterprise Portal, access control is managed dynamically by the Portal runtime using the configuration database of the UME. No programming is required by the application developer.

1-4-3 Roles, Users, and Groups

With UME, as with most other authorization systems, user access to applications is controlled using Role objects. Roles are used to control access even in the simple case demonstrated in the diagram of the previous section.

Role objects can be created declaratively or programmatically but are usually created declaratively by a system administrator as part of system configuration. Roles have a name and description that should portray an obvious association between users and the applications they will be using in the system landscape. For example, a Role of Manager should have obvious meaning to the users of the software system. Roles often correspond to the actual classification of individuals within an organization. To complete the definition of a Role, as defined for system administration, the Role needs to be associated with one or more Actions and one or more Users and/or Groups as shown below:

A User object simply represents an individual who has some access to the applications in the system while a Group is simply a collection of one or more users.

Based on the objects and associations presented above, authorization becomes the process of matching user permissions with application permissions based on the Actions allowed by the application. In the simple case presented above, the EmployeeAdmin application has been assigned to allow access to any user that has DocumentPermission with Manage access (Manage is the Action in this case). To determine if the user is allowed access, the security manager of the Container or runtime (e.g., Portal Container) will extract the list of Actions from the user’s Role for the given application (in this case, all Actions for the EmployeeAdmin application) and create Permissions for these. The list of Permissions is checked against the allowed Permissions for the application. The first match will give the user...
permission to access the application. If the user has no Actions in its Role(s) for the given application, or if none of its Permissions match those of the application, then the user is denied access to the application. This process is illustrated in the diagram below.

In the diagram above, the Portal Security Manager calls UME to obtain the user’s allowed Actions on the given application. Then the security manager generates the Permissions from the user’s Actions. Finally, the security manager calls UME to obtain the Permissions assigned to the application and checks these against the user’s Permissions. If a match occurs, the user is allowed access. If the security manager checks all of the user’s Permissions without finding a match, then access is denied.

For WDP applications running in the Enterprise Portal, the administrator can restrict application access based on:

1. The user’s company object
2. The Permissions that are derived from the user shown in the above diagram

1-4-4 Programming Control of Application Access

This section shall deal with programming control of application access. As noted in section 1-3-2:
“For example, perhaps a WDP EmployeeAdmin application is available for viewing by HR managers but only Engineering Managers have “create” and “update” permissions. If the configuration tools used by the system administrator can only limit access to the application as a whole (e.g., either the user has permission to access the application or not), then the application programmer will have to write code to provide finer granularity of access control.”

This section will explain the mechanics of programming access control. Referring to the diagram in section 1-3-1 and the scenario described in the previous paragraph, the system administrator will configure the EmployeeAdmin application as before to only be accessible by users with EmployeeAdmin:Manage Actions. However, the application programmer will want to add the finer level of control indicated above. The diagram below illustrates the steps taken by the WDP programmer to accomplish this task.

In the diagram above, the WDP programmer adds code to the wdDoInit() method of the “CreatePerformanceReview” View Controller. The code calls UME to obtain the user’s Role and checks this against the Role of “Engineering Manager.” If the Role is a match, access is allowed, otherwise, the application will respond to the user with a popup window indicating that access is denied.

**UME Access from WDP**

As indicated in Section 1-3, WDP provides a central interface for UME access. The com.sap.tc.webdynpro.services.sal.um.api.IWDClientUser class provides access to both anonymous and authenticated client users. This class is more or less a wrapper around the UME’s com.sap.security.api.IUser interface.

The IWDClientUser class provides many useful methods:
<table>
<thead>
<tr>
<th>Method</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IsAnonymousUser</code></td>
<td>Check if the client user is anonymous</td>
</tr>
<tr>
<td><code>forceLoggedInClientUser</code></td>
<td>Force user authentication (user must be logged in)</td>
</tr>
<tr>
<td><code>forceLogoffClientUser</code></td>
<td>Force user logoff</td>
</tr>
<tr>
<td><code>CheckAuthentication</code></td>
<td>Reads the configuration of the current WDP application and checks if user authentication is required. If so, it forces a user login</td>
</tr>
<tr>
<td><code>GetCurrentUser</code></td>
<td>Access the current client user (the client user attached to the current session)</td>
</tr>
</tbody>
</table>

**Application Coding Example**

```java
// All used classes of the user management service are contained in package
// com.sap.tc.webdynpro.services.sai.um.api

// Let applicationPart be a WDDeployableObjectPart that represents a
// deployable object part of type WDDeployableObjectPartType.APPLICATION.

// Example 1:
// Check application configuration if user login is required and
// force login if this is the case; use method
// checkAuthentication(WDDeployableObjectPart) for that. The class
// returns true if authentication was successful.

if ( WDClientUser.checkAuthentication(applicationPart) ) {
    // do something
} else {
    // authentication was not successful, so don't go on with
    // the regular application
}

// Example 2:
// get the currently logged in user
IWDClientUser user = WDClientUser.getCurrentUser();
// check whether the current user is authenticated or anonymous
if ( user.isAnonymous() ) {
    // anonymous user
} else {
    // authenticated user
}

// read some user profile information
String name = user.getLastName();
String title = user.getTitle();
// get the com.sap.security.api.IUser; it is null in case user
// represents an anonymous user
IUser iUser = user.getSAPUser();
if (iUser != null) {
    // do something
}
```