Oracle® Database

Development Security Guide

Release 14.7.5.0.0
G24944-01
September 2024

Oracle Database Development Security Guide, Release 14.7.5.0.0

G24944-01

Primary Authors: (primary author), (primary author)

Contributing Authors: (contributing author), (contributing author)

Contributors: (contributor), (contributor)

Contents

1
Preface

1.1 Purpose

This guide is designed to help the user to quickly get acquainted with the Customer Standard Instructions maintenance process.

1.2 Audience

This guide is intended for the central administrator of the Bank who controls the system and application parameters and ensures smooth functionality and flexibility of the banking application.

1.3 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

1.4 Conventions

The following text conventions are used in this document:

Table 1-1 Conventions

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.

Table 1-1 (Cont.) Conventions

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1.5 Critical Patches

Oracle advises customers to get all their security vulnerability information from the Oracle Critical Patch Update Advisory, which is available at Critical Patches, Security Alerts and Bulletins. All critical patches should be applied in a timely manner to ensure effective security, as strongly recommended by Oracle Software Security Assurance.

1.6 Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1.7 Basic Actions

Table 1-2 Basic Actions

Action	Description
Approve	Used to approve the initiated report. This button is displayed, once the user click **Authorize**.
Audit	Used to view the maker details, checker details, and report status.
Authorize	Used to authorize the report created. A maker of the screen is not allowed to authorize the report. Only a checker can authorize a report, created by a maker.
Close	Used to close a record. This action is available only when a record is created.
Confirm	Used to confirm the performed action.
Cancel	Used to cancel the performed action.
Compare	Used to view the comparison through the field values of old record and the current record. This button is displayed in the widget, once the user click **Authorize**.
Collapse All	Used to hide the details in the sections. This button is displayed, once the user click **Compare**.
Expand All	Used to expand and view all the details in the sections. This button is displayed, once the user click **Compare**.

Table 1-2 (Cont.) Basic Actions

Action	Description
New	Used to add a new record. When the user click **New**, the system displays a new record enabling to specify the required data.
OK	Used to confirm the details in the screen.
Save	Used to save the details entered or selected in the screen.
View	Used to view the report details in a particular modification stage. This button is displayed in the widget, once the user click **Authorize**.
View Difference only	Used to view a comparison through the field element values of old record and the current record, which has undergone changes. This button is displayed, once the user click **Compare**.
Unlock	Used to update the details of an existing record. System displays an existing record in editable mode.

1.8 Screenshot Disclaimer

Personal information used in the interface or documents is dummy and does not exist in the real world. It is only for reference purposes.

2 Scope

This topic explains the scope of the manual.

Read Sections Completely

Each section should be read and understood completely. Instructions should never be blindly applied. Relevant discussion may occur immediately after instructions for action, so be sure to read whole sections before beginning implementation.

Understand the Purpose of this Guidance

The purpose of the guidance is to provide security-relevant code and configuration recommendations.

Limitations

This guide is limited in its scope to the security-related guideline for developers.

3

How to address the OWASP Top10 in FLEXCUBE UBS

This topic explains to address the OWASP Top10 in FLEXCUBE UBS.

Injection

Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or SQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code.

FLEXCUBE uses Oracle database and, it has adequate inbuilt techniques to prevent SQL injections as, underlined below:

1. **Use of prepared statements (parametrized queries)**: FLEXCUBE uses PreparedStatement with bind variables to construct and execute SQL statements in JAVA.
2. **Use of Stored procedures**: Stored procedures have the same effect as the use of prepared statements when implemented safely. **Implemented safely** means the stored procedure does not include any unsafe dynamic SQL generation. FLEXCUBE uses safe Java stored procedures calls.
3. In addition to the above, wherever dynamic queries exist, FLEXCUBE uses adequate defense to sanitize the untrusted input. The use of DBMS_ASSERT.SIMPLE_SQL_NAME and the use of bind variables justify the fact.
4. **Escaping all user supplied input**: This third technique is to escape user input before putting it in a query. If it's a concern that rewriting the dynamic queries as prepared statements or stored procedures might break the application or adversely affect performance, then this might be the best approach for the purpose. However, this methodology is frail compared to using parametrized queries and there's no guarantee that it will prevent all SQL Injection in all situations.

FLEXCUBE uses context specific escaping. It has a StringEscapeUtils.java file, where context specific escaping is handled.

Broken Authentication and Session Management

In FLEXCUBE Universal Banking Solutions application session interval will be validated against the session interval stored in the configurable file FCUBS.properties file. Validations are added to check the maximum time limit for the inactive session from being expired. Java API method javax.servlet.http.HTTPSession will set the max time out period for the session.

A maximum limit is imposed on the value passed to set the maximum limit of session interval. The maximum limit is a positive practical value. This validation is required to prevent long running sessions that can be actively targeted.

The default value for session time out is 30 minutes and it is configurable in the FCUBS properties file.

The session used for login authentication will be invalidated (destroyed) and a new session will be created once the user logged in successfully to the application and the new session will be used to store the required variables.

A session attribute IsAuthenticated set to “Y” on successful login to the application. A new random token (Cross-site request forgery) is also generated and the same is available in the session attribute.

The entire subsequent request within the session will be having the Authenticated and Cross-site request forgery tokens. Every request sent to the application from the browser is validated against the IsAuthenticated attribute and Cross-site request forgery token.

A hidden form is used to submit the logout request to the server, with the response resulting in a 302 redirect instead of a client-initiated redirect to the login page.

The session gets expire once the user logs off from the application or if idle for its maximum limit.

Cryptography used

PCI council defines **Strong Cryptography** as:

Cryptography based on industry-tested and accepted algorithms, along with strong key lengths and proper key-management practices. Cryptography is a method to protect data and includes both encryption (which is reversible) and hashing (which is not reversible, or “one way”). SHA-1 is an example of an industry-tested and accepted hashing algorithm. Examples of industry-tested and accepted standards and algorithms for encryption include AES (128 bits and higher), TDES (minimum double-length keys), RSA (1024 bits and higher), ECC (160 bits and higher), and ElGamal (1024 bits and higher)

Encryption algorithm: The application leverages the AES encryption algorithm to store sensitive information into the properties file. This algorithm uses a 256 bit secret key for encryption and decryption which would be stored at property file.

Hashing algorithm: FLEXCUBE Universal Banking Solutions leverages the SHA-512 hashing algorithm for user password authentication. This algorithm generates a password digest for the user password by using the SALT (Random number generated using SHA1PRNG algorithm) and the iteration number available in the property file.

Session storage

FLEXCUBE Universal Banking Solutions application does not store Http Session objects.

A unique sequence number generates and stored in the current user table for the purpose of mapping server-side sessions with the entries in the current user table.

During session expiry (triggered by the container), the session listener provides the application with the sequence number of the session. The application makes checks as to whether the entry in the current user table contains the same sequence number. Only in such a case should the entry be deleted.

When authentication of credentials (involving an incorrect user ID) is unsuccessful, the user id should not be logged in the audit logs (database table). The following possible scenarios will be accounted for:

- **Session logging**
  An unsuccessful login attempt is stored in the database with the terminal’s IP address and timestamp. Invalid and expired session IDs submitted to the application are categorized as authentication failures and, the same is logged in the database table.
- Cross-Site Scripting (XSS)
  This topic explains the Cross-Site Scripting (XSS) prevention rules.
- Insecure Direct Object References
  This topic explains insecure direct object references.

3.1 Cross-Site Scripting (XSS)

This topic explains the Cross-Site Scripting (XSS) prevention rules.

XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user-supplied data in a page sent to the browser without properly validating or escaping that content. FLEXCUBE is coded keeping in view the XSS prevention rules as below

1. **Technique#1: HTML Escape before inserting untrusted data into HTML element content**
   Across the FLEXCUBE application, context-specific escaping has been used to sanitize the untrusted data. For HTML content, the below function takes care of escaping the probable tainted data:

   public static String escapeHTML(String input);
   Escaping the following characters with HTML entity encoding to prevent switching into any execution context, such as script, style, or event handlers, has been done. Use of recommended hex entities is in place. In addition to the five characters significant in XML (&, <, >, ", '), the forward slash is included as it helps to end an HTML entity.

   & --> &

   < --> <

   > --> >

   " --> "

   ' --> '

   / --> /

2. **Technique #2: JavaScript Escape Before Inserting Untrusted Data into JavaScript Data Values**
   Including untrusted data inside any other JavaScript, context is quite dangerous as it is extremely easy to switch into an execution context with characters including (but not limited to) semi-colon, equals, space, plus, and many more. For JavaScript context, the below function takes care of escaping the probable tainted data:

   public static String escapeJavaScript(String input);

3. **Technique #3: Escape JavaScript Characters**
   This works in conjunction with rule#2. Except for alphanumeric characters in FLEXCUBE, all characters less than 256 are escaped with the \xHH format to prevent switching out of the data value into the script context or another attribute. No use of any escaping shortcuts

like \", because the quote character may be matched by the HTML attribute parser which runs first. These escaping shortcuts are also susceptible to "escape-the-escape" attacks where the attacker sends \", and the vulnerable code turns that into \\" which enables the quote.

4. **Technique #4: URL Escape And Strictly Validate Before Inserting Untrusted Data into HTML URL Parameters.**
   FLEXCUBE encodes URL with the URLEncoder java class. It doesn't check for a valid URL but directly does URL encoding, and that encoding is based on the context of display.
5. **Technique #5: Use of HttpOnly and secure cookie flag**
   FLEXCUBE uses the HTTPOnly flag on the session cookie and any custom cookies that are not accessed by any JavaScript.

3.2 Insecure Direct Object References

This topic explains insecure direct object references.

1. **Use of prepared statements (parameterized queries)**
   FLEXCUBE uses PreparedStatement with bind variables to construct and execute SQL statements in JAVA.
2. **Input Validation**
   FLEXCUBE is a web-based application. The request data from browser to server will be passed using request headers and request parameters. All the request fields coming from the client are validated using white list validation to prevent cross-site scripting.

   User-defined method validateParameter() is used for input validation which checks each character of the request field with a range of allowed characters.User defined methods escapeJavaScript(), escapeHTML() and escapeURL() will sanitize the output data before flushing it into client browser.

   escapeJavaScript() will escape all characters except immune JavaScript characters and alphanumeric characters in the ASCII character set. All other characters are encoded using the \\xHH or \\uHHHH notation for representing ASCII or Unicode sequences.escapeHTML() will escape the characters with equivalent HTML entities obtained from the lookup map. The lookup map will have entities such as amp, quot, lt, gt, etc.escapeURL() will encode the URL using URLEncoder class.White list validation is also used to restrict Image/signature/excel upload and to check rights for every operation performed by the user.
3. **.Image Content validation**
   The signature upload will check for image type and image content using the inbuilt classes (ImageIO and JarFile) available in java.
4. **Field validation**
   Field level validations exist for all mandatory fields. Database too had limits on the type and the length of data. Blacklisted characters are not allowed in the mandatory fields. Nevertheless, FLEXCUBE has free-text fields, which takes all data, entered by the user, as a String.
5. **Restriction on Blacklist characters**
   Similar to white list validation black list validation is also used for validating the request fields. FLEXCUBE uses blacklist validation to check whether the request XML contains unwanted tags like scripting tag, HTML tag, anchor tag, etc inside the XML content. It is also used for the advance summary field's validation to check whether proper request fields are coming from the browser.

   The below table shows the list of bad characters which should not be allowed in the URL path but, the FLEXCUBE operations require many of the below characters to be passed in

the request. So FLEXCUBE will encode the below bad characters before sending them through the URL and, the same will be decoded at the server to prevent the hacker from modifying the request.

Table 3-1 Bad URL Characters

Bad URL Characters(Unsafe Characters)	
&	//
<	./
>	/.
;	/*
\"	*.
\'	~
%	\
)	25%
(	%25u
+	%25U
,	%00-%1f, %7f-%ff
" " (space)	%00-%1f and %7f-%ff
-	%25u and %25U

6. **Restriction on Script/HTML Tags**
   FLEXCUBE has blacklist validation for the unwanted tag in XML like scripting tag or HTML tag inside XML content, particularly in the header.

3.3 Security Misconfiguration

This topic explains the security misconfiguration.

1. **Configuration Files**
   Configuration files are securely placed inside the Classes folder of the WEB-INF folder which, is not publicly accessible.
2. **Exception handling in java**
   Different types of exceptions can rise in the application. Java exceptions are handled using try-catch blocks available in java. Sometimes we use the Throw statement to throw an exception that is caught by the catch block. Caught exceptions will be written into the log files for debug purposes whenever required. Whenever an exception occurs in the application, proper information is used to send to the front end user by showing an alert.
3. **Exception handling in Oracle Database**
   Database exceptions handled using **EXCEPTION** statement available in PL/SQL. Caught exceptions will be written into the log files for debug purposes, and a proper error message is created to send the same in response to the user.
4. **Package Lockout Situation Handled in Backend**
   The application will be hanged in an oracle system package lockout situation. Locked objects will be released manually using SQL scripts or through database restart. We have handled the cursor lock-out problem in the required packages.
5. **Auto Generated Password**
   The password is generated by the system in accordance with the password policy. The salt is also be generated every time the password is changed by using a predefined algorithm.

The salt concatenated with an auto-generated password and SHA-512 hash applies on the resultant which results in the password digest.

Once the successful generation of password digests both salt and password digest are stored in the DB.

6. **Custom Password**
   The password is keyed in by the administrator/user in accordance with the password policy. The salt is generated every time the password is changed by using a predefined algorithm.

   The salt concatenated with an auto-generated password and SHA-512 hash applies on the resultant which results in the password digest.

   Once the successful generation of password digests both salt and password digest are stored in the DB.

   Oracle FLEXCUBE Universal Banking does not provide any default user/password. User and password need to be created at the time of installation.

7. **Sand Box for File Upload**
   The application uses a sandbox for placing files that are uploaded via the signature/image upload screen. The sandbox is placed in a specified location (the location will be specified in the properties file) on the server.

8. **BI Publisher Reports – Generation and Access**
   The application uses a sandbox for placing the generated reports file into a sandbox area. The sandbox is placed in a specified location (the location will be specified in the properties file) on the server. The application validates if the user has explicit Rights to generate Reports.

3.4 Sensitive Data Exposure

This topic explains the sensitive data exposure.

1. **Secure Transformation of Data (SSL)**
   The FLEXCUBE UBS Installer allows a deployer to configure FLEXCUBE UBS such that all HTTP connections to the FLEXCUBE UBS application are over SSL/TLS. In other words, all HTTP traffic in the clear will be prohibited; only HTTPS traffic will be allowed. It is mandatory to enable this option in a production environment, especially when WebLogic Server acts as the SSL terminator.

   A two-way SSL is used when the server needs to authenticate the client. In a two-way SSL connection, the client verifies the identity of the server and then passes its identity certificate to the server. The server then validates the identity certificate of the client before completing the SSL handshake.

   To establish a two-way SSL connection, need to have two certificates, one for the server and the other for the client. This is required for the decentralized setup of the application.

   For Oracle FLEXCUBE Universal Banking Solutions, need to configure a single connector. This connector is related to SSL/TLS communication between the host or browser and the branch which, uses two-way authentication.

   If the secure flag is set on a cookie, then browsers will not submit the cookie in any requests that use an unencrypted HTTP connection. Thereby preventing the cookie from being trivially intercepted by an attacker monitoring network traffic.

   The below configuration has to be ensured in weblogic.xml within the deployed application ear.

   - Cookies are set with Http only as true

- Cookie secure flag set to true
- Cookie path to refer to deployed application

```
<wls: session-descriptor>
 <wls: cookie-http-only>true</wls: cookie-http-only>
</wls: session-descriptor>
<wls: session-descriptor>
<wls: cookie-secure>true</wls: cookie-secure>
<wls: url-rewriting-enabled>false</wls: url-rewriting-enabled>
</wls: session-descriptor>
<session-descriptor>
 <cookie-name>JSESSIONID</cookie-name>
 <cookie-path>/<DeployedApplicationPath></cookie-path>
 <cookie-http-only>true</cookie-http-only>
 <cookie-secure>true</cookie-secure>
 <url-rewriting-enabled>false</url-rewriting-enabled>
</session-descriptor>
```

Always make sure Cookies are set with always Auth Flag enabled by default for WebLogic server.

2. **Sign-On Messages**
The below table shows the general Sign-On messages which would be displayed to the user during invalid authentication.

Table 3-2 Sign-On Messages

Message	Explanation
User Already Logged In	The user has already logged into the system and is attempting a login through a different terminal.
User Authentication Failed	Entered an incorrect User ID and Password.
User Status is Disabled. Please contact your System Administrator	The user profile has been disabled due to the number of dormancy days allowed for the user has exceeded the dormancy days configured in the system.
User Status is Locked. Please contact your System Administrator	The user profile has been locked due to an excessive number of login attempts using an incorrect user ID or password. The number of attempts could have matched either the success or a cumulative number of login failures (configured for the system).

3. **CACHE Control in Servlet and jsp**
Three basic HTTP response headers prevent a page from being cached to a disk. Different browsers handle them in slightly different ways, so they need to be used in combination to ensure all browsers do not cache the specific page. These headers are:

- Expires
- Pragma
- Cache-control

In addition, these headers can either be sent directly by the server or placed in the HTML code as HTTP-EQUIV META tags within the HEAD section. The "Expire" header gives a date at which point the page should expire and no longer be cached. Internet Explorer

supports a date of "0" for immediately and any negative number for already expired. The "Pragma: no-cache" header indicates that the page should not be cached.

4. **Clickjacking/Frame-bursting**
FLEXCUBE uses the X-Frame-Options HTTP response header to indicate whether or not a browser should be allowed to render a page in a <frame> or <iframe>. This is used to avoid Clickjacking attacks by ensuring that the content is not embedded into other sites.

3.5 Missing Function Level Access Control

This topic explains in missing the function level access control.

Likely, users working in the same department at the same level of hierarchy need to have similar user profiles. In such cases, you can define a Role Profile that includes access rights to the functions that are common to a group of users. A user can be linked to a Role Profile by which you give the user access rights to all the functions in the Role Profile.

Application level access has been implemented via the **Security Management System** (SMS) module. SMS supports **ROLE BASED** access of Screens and different types of operations.

FLEXCUBE Universal Banking Solutions supports dual control methodology, wherein every operation performed has to be authorized by another user with the requisite rights. Please refer the SMS user manual for more details.

Apart from the role based access control of particular functions, products can be restricted for the user as described below.

- **Disallowed functions**: Function IDs or UI level restrictions can be provided for the user by including the function Ids in the disallowed list. This will restrict the user from accessing the UI. When accessed, an error message dialogue box displays `User not authorized to access the screen`.
- **Disallowed account class**: The user could be restricted to perform any operation using a particular account class. When disallowed, no accounts could be created by the user using the account class.
- **Disallowed products**: The user could be restricted to use the product(s) of any module(s) if disallowed. This is really required when restricting users department wise. For example, staff of the accounts department need not be given access to view the loans of customers.
- **Disallowed branches**: The user could be restricted to access branches other than his branch (reporting branch). He can be given access to login from other branches of the bank at approval from the authenticated person, an action which again requires manual authorization.

3.6 Cross-Site Request Forgery (CSRF)

This topic helps to understand the Cross-Site Request Forgery (CSRF).

In the case of XMLHttpRequest objects, the XMLHttpRequest object sets a custom HTTP header in the request, with the header value being the Cross-site request forgery token; The server then verifies for the presence of such a header and the Cross-site request forgery token. This serves as protection at endpoints used for XMLHttpRequest requests since only XMLHttpRequest objects can set HTTP headers (apart from Flash; and both cannot make cross-domain requests).

3.7 Use Components with Known Vulnerabilities

This topic explains to use components with known vulnerabilities.

Source code scanning done using the latest fortify to identify the sources code issue and will provide the proper fix for the reported issues.

The third party libraries scanning for every release has been done to validate if any security issues rise for any of the components or not. Update the 3PL with the latest security patch or upgraded to the latest version.

3.8 Unvalidated Redirects and Forwards Network Security

This topic explains the unvalidated Redirects and Forwards network security.

The application uses 302 redirect wherever required. FLEXCUBE UBS uses response.sendRedirect(newURL);

4

Secure Gateway Services

This topic explains to secure gateway services.

Different applications deployed on disparate platforms and using different infrastructure need to be able to communicate and integrate seamlessly with Oracle FLEXCUBE Universal Banking to exchange data. The Oracle FLEXCUBE Integration Gateway will cater to these integration needs.

The integration needs to be supported by the Gateway can be broadly categorized from the perspective of the Gateway as follows:

- **Inbound application integration**: Used when any external system needs to add, modify or query information within Oracle FLEXCUBE Universal Banking.
- **Outbound application integration**: Used when any external system needs to be notified of the various events that occur within Oracle FLEXCUBE Universal Banking.

Inbound Application Integration

Oracle FLEXCUBE Inbound Application Gateway provides XML-based interfaces enhancing the need to communicate and integrate with the external systems. The data exchanged between Oracle FLEXCUBE Universal Banking and the external systems will be in the form of XML messages. These XML messages are defined in Oracle FLEXCUBE Universal Banking in the form of XML Schema Documents (XSD) and are referred to as **FCUBS formats**.

FCUBS Inbound Application Integration Gateway uses the Synchronous and Asynchronous Deployment Pattern for addressing the integration needs.

The Synchronous Deployment Pattern is classified into the following:

- Oracle FLEXCUBE Universal Banking EJB Based Synchronous Inbound Application Integration Deployment Pattern.
- Oracle FLEXCUBE Web Services Based Synchronous Inbound Application Integration Deployment Pattern.
- Oracle FLEXCUBE Universal Banking MDB Based Asynchronous Inbound Application Integration Deployment Pattern.

EJB Based Synchronous Deployment Pattern

The Enterprise Java Beans (EJB) deployment pattern will be used in integration scenarios where the external system connecting to Oracle FLEXCUBE Universal Banking is **EJB literate**, i.e., the external system is capable of interacting with Oracle FLEXCUBE Universal Banking based upon the EJB interface. In this deployment pattern, the external system will use the RMI/IIOP protocol to communicate with the Oracle FLEXCUBE EJB.

In this deployment pattern, the EJB displayed by Oracle FLEXCUBE will be a stateless session bean. The actual request will be in the form of an XML message. After the necessary processing is done in Oracle FLEXCUBE based on the request, the response is returned to the external system as an XML message. The transaction control for the processing will stay with the Oracle FLEXCUBE EJB.

Web Services Based Synchronous Deployment Pattern

The web services deployment pattern will be used in integration scenarios where the external system connecting to Oracle FLEXCUBE Universal Banking wants to connect using standards-based, inter-operable web services.

This deployment pattern is especially applicable to systems that meet the following broad guidelines:

- Systems that are not EJB literate, i.e., such systems are not capable of establishing connections with Oracle FLEXCUBE based upon the EJB interface; and/or
- Systems that prefer to use a standards-based approach.

In this deployment pattern, the external system will use the SOAP (Simple Object Access Protocol) messages to communicate to the Oracle FLEXCUBE Universal Banking web services.

The services displayed by Oracle FLEXCUBE Universal Banking are of a message-based style, i.e., the actual request will be in the form of an XML message, but the request will be a **payload** within the SOAP message. After the necessary processing is done in Oracle FLEXCUBE based on the request, the response is returned to the external system as an XML message which will be a **payload** within the response SOAP message. The transaction control for the processing will stay with the Oracle FLEXCUBE Universal Banking.

HTTP Servlet Based Synchronous Deployment Pattern

The HTTP servlet deployment pattern will be used in integration scenarios where the external system connecting to Oracle FLEXCUBE Universal Banking wants to connect to Oracle FLEXCUBE Universal Banking using simple HTTP messages.

This is especially applicable to systems such as the following:

- Systems that are not '**EJB literate**', i.e., are not capable of establishing a connections with Oracle FLEXCUBE Universal Banking based upon the EJB interface; and/or
- Systems that prefer to use a simple HTTP message-based approach without wanting to use SOAP as the standard.

In this deployment pattern, the external system will make an HTTP request to the Oracle FLEXCUBE servlet.

For this deployment pattern, Oracle FLEXCUBE Universal Banking will display a single servlet. The actual request will be in the form of an XML message. This XML message is embedded into the body of the HTTP request sent to the Oracle FLEXCUBE servlet. After the necessary processing is done in Oracle FLEXCUBE Universal Banking based on the request, the response is returned to the external system as an XML message which is once again embedded within the body of the response HTTP message. The transaction control for the processing will stay with the Oracle FLEXCUBE Universal Banking.

MDB Based Asynchronous Deployment Pattern

The MDB deployment pattern is used in integration scenarios where the external system connecting to Oracle FLEXCUBE wants to connect to Oracle FLEXCUBE using JMS queues. This is especially applicable to systems such as the following:

- Systems that prefer to use JMS queues based approach without wanting to wait for the reply.

Here external system sends messages in XML format to request a queue on which an MDB is listening. When a message arrives in the queue, it is picked up for processing. After the

necessary processing is done in Oracle FLEXCUBE Universal Banking, based on the request, the response is sent to the response queue as an XML message.

Outbound Application Integration

The Outbound Application Integration is also called the Oracle FLEXCUBE Universal Banking Notify Application Integration layer. This application layer sends out notification messages to the external system whenever events occur in Oracle FLEXCUBE Universal Banking.

The notification messages generated by FCUBS on the occurrence of these events will be XML messages. These XML messages are defined in FCUBS in the form of XML Schema Documents (XSD) and are referred to as **FCUBS formats**.

Secure Web Services

Web services can be secured by applying security policies available in the weblogic server. We can attach two types of policies to Web Logic Web services and clients at design and deployment time.

- **Oracle WSM policy**: We can attach Oracle Web Services Manager(WSM) policies to Web Logic JAX-WS Web services and clients.
- **WebLogic Web service policy**: These policies are provided by Oracle Web Logic Server and can be attached to any web service deployed in Web Logic.

We can use Oracle Enterprise Manager Fusion Middleware Control to attach Oracle WSM security policies to Web Logic Java EE Web services and clients.

We can attach policies to WebLogic Web services at both design time and after the Web service has been deployed.

At design time, use the **weblogic.jws.Policy** and **weblogic.jws.Policies** JWS annotations in JWS file to associate policy files with Web service. We can associate any number of policy files with a Web service, although it is up to us to ensure that the assertions do not contradict each other. We can specify a policy file at the class level of our JWS file.

After the Web service has been deployed, use the Oracle Web Logic Server Administration Console to attach Web Logic Web service policies to Web Logic Web services.

Access Service and Operation

In a message, it is mandatory to maintain a list of Service Names and Operation Codes. This information is called Gateway Operations.

A combination of every such Service Name and Operation Code is mapped to a combination of Function ID and Action. Every screen in Oracle FLEXCUBE Universal Banking is linked with a function ID. This information is called Gateway Functions.

Users can gain access to an external system using the Gateway Functions. The Function IDs mapped in Gateway Functions should be valid Function IDs maintained in Oracle FLEXCUBE Universal Banking. Hence, for every new Service or Operation being introduced, it is important that you provide data in Gateway Operations and Gateway Functions.

Gateway Password Generation Logic for External System Authentication

As a secure configuration password authentication should be enabled for the external system maintained. The same can be verified in the External system detail screen level.

Once these features are enabled, the system will validate for Encrypted password as part of every request sent by the External System.

The Message ID which is present as part of the header in Request XML is considered as the hash. External System generates a unique Message ID, which is a functional mandatory field in the header. Create a Message Digest with the SHA-512 algorithm.

The hash created from the previous step and the password in the clear text together is encrypted in the AES encryption method. Apply Base64 encoding to encrypted value and send to the Oracle FLEXCUBE Universal Banking gateway.

XSD Validation and Input Validation

Oracle FLEXCUBE Universal Banking supports the XSD validation for all types of Gateway. Each node in request XML is getting validated with the corresponding webservice XSD's.

Restriction on Script/HTML tags

Oracle FLEXCUBE Universal Banking Gateway has blacklist validation for the unwanted tag in XML like scripting tag or HTML tag inside XML content, particularly in the header.

List of Services

- FCUBSACService
- FCUBSAMService
- FCUBSAccFinService
- FCUBSAccService
- FCUBSBstService
- FCUBSCAService
- FCUBSCFService
- FCUBSCIService
- FCUBSCLService
- FCUBSCNService
- FCUBSCPGServices
- FCUBSCcyService
- FCUBSCoreService
- FCUBSCoreentitiesService
- FCUBSCustomerService
- FCUBSDEService
- FCUBSDLService
- FCUBSDQService
- FCUBSEPSService
- FCUBSFAService
- FCUBSFIService
- FCUBSGIService
- FCUBSGLService
- FCUBSIAService
- FCUBSICService

- FCUBSIFService
- FCUBSINService
- FCUBSISService
- FCUBSIVService
- FCUBSInteractionService
- FCUBSLDService
- FCUBSLEService
- FCUBSLMService
- FCUBSLeadService
- FCUBSMFService
- FCUBSMOService
- FCUBSMSService
- FCUBSMessagingService
- FCUBSORService
- FCUBSRBService
- FCUBSRMService
- FCUBSSCVService
- FCUBSSFService
- FCUBSSIService
- FCUBSSMService
- FCUBSSTService
- FCUBSSigService
- FCUBSSwitchService
- FCUBSTDFinService
- FCUBSTDService
- FCUBSUPService
- FCUBSVPService
- FCUBSXPService
- SMSUserService

List of Interfaces

Integration/Interface with Oracle Products

1. Oracle FLEXCUBE UBS - ODA Integration User Guide

Integration/Interface with Product Processors

1. Oracle FLEXCUBE UBS - Common Core Integration User Guide
2. Oracle FLEXCUBE UBS - OBVAM Integration User Guide
3. Oracle FLEXCUBE UBS - Payments Integration User Guide
4. Oracle FLEXCUBE UBS - ELCM Integration

5. Oracle FLEXCUBE UBS - OBTR Integration User Guide
6. Oracle FLEXCUBE UBS - OBTF Integration User Guide
7. Oracle FLEXCUBE UBS - Oracle Banking Liquidity Management Integration
8. Oracle FLEXCUBE UBS - Oracle Banking Origination Integration User Guide
9. Oracle FLEXCUBE UBS - OFSAA Integration User Guide
10. Oracle FLEXCUBE Investor Servicing Integration User Guide
11. Oracle FLEXCUBE UBS - Corporate Lending Integration User Guide
12. Oracle FLEXCUBE UBS - Common Core - OBMA Core Integration

Generic Interfaces

1. Relationship Pricing Interface User Guide
2. Oracle FLEXCUBE UBS - External Accounting Interface
3. Oracle FLEXCUBE UBS - Biometric Integration User Guide
4. Debit Card Interface User Guide
5. Document Management System Interface User Guide
6. Hajj Registration Interface User Guide
7. Single Customer View Hand-off User Guide