Oracle® Application Testing Suite

Security Guide

Release 13.3.0.1

E74489-05

May 2018

This document provides an overview of the security features in the Oracle Application Testing Suite.

If you have any questions or problems, please contact our support group at http://www.oracle.com/support/index.html.

Contents

This document has the following sections:

OATS Installer Security Features

This section has the following topics:

Overview

The Oracle Application Testing Suite application installation includes the following security features:

- Installation procedure password. The installation procedure includes setting a master Administrator password that is required for accessing the server-based applications (Administrator, Oracle Test Manager, and Oracle Load Testing).
- Administrator access rights required for installation. The user/administrator installing the Oracle Application Testing Suite must have local administrator access rights and have the ability to run the 'elevated' installer.

- User access right required for OpenScript (i.e. Oracle Functional Test, or OFT). Users working with OpenScript must have local administrative user access rights.
- Oracle WebLogic server is installed as part of the Oracle Application Testing Suite to use with OATS server-based applications (Administrator, Oracle Test Manager, and Oracle Load Testing).

User -Defined Installation Password

The Oracle Application Testing Suite installation includes setting a master Administrator password that is required for accessing the server-based applications (Administrator, Oracle Test Manager, and Oracle Load Testing).

During the installation for Oracle Application Testing Suite you will be asked to provide a master password. *Remember this password*. The master password specified during installation is used to set all of the following user passwords:

- Oracle Test Manager users "default" and "administrator".
- Oracle Load Testing users "default" and "administrator".
- User "JMSAdmin" used for Oracle Load Testing agent to controller authentication.
- User "oats" for Weblogic Server - this is a Weblogic superuser.
- User "oats-agent" in Weblogic Server for JMS communication between agent and controller - this is *not* a Weblogic superuser.
- Database users "oats", "olt", "otm".

Administrator Access Rights Required for Installation

Installation user/administrator must have local administrator access rights and have the ability to run the 'elevated' installer using right-click and select **Run as administrator** from the popup menu.

User Access Right Required for OpenScipt

Users working with OpenScript must have local administrative user access rights. User working with OpenScript must have full access to change browser settings and to install add-ons to the browser (BHO/plug-ins).

When testing Oracle FORMS, users must have access right to start OpenScript as 'elevated' using right-click and select **Run as administrator** from the popup menu.

Oracle WebLogic Server Installed

Oracle WebLogic server 12.1.3. is installed as part of the Oracle Application Testing Suite for use with OATS server-based applications (Administrator, Oracle Test Manager, and Oracle Load Testing). Refer to the Oracle WebLogic Server documentation for security information.

OATS Administrator Security Features

This section has the following topics:

Overview

The Oracle Application Testing Suite Administrator application includes the following security features:

- Password-based Administrator log in.
- Separate user databases for Oracle Test Manager and Oracle Load Testing users.
- Ability to optionally enable authentication for Oracle Load Testing. When Oracle Load Testing login is enabled, users must login to access Oracle Load Testing.
- Ability to specify users, passwords, and privileges for Oracle Load Testing. The Administrator allows you to create user accounts, assign them user names and passwords, and assign the type of access that they have in Oracle Load Testing, none, full control, or view only.
- Ability to specify users, passwords, and role-based privileges for Oracle Test Manager. The Administrator allows you to create user accounts and assign users roles in specific projects. Roles determine the permissions the user has in each project.
- Ability to configure LDAP-based login for Oracle Test Manager users. You can implement Lightweight Directory Access Protocol (LDAP) server authentication to allow Oracle Test Manager users authentication against the LDAP server providing users a single sign-on solution.
- Ability to configure role-based field security settings to specify which roles have read/write permissions and which roles have read-only permissions for specific fields.

Administrator Login

Specific users can be granted Administrator access to Oracle Application Testing Suite applications. This restricts access to the Administrator features to specific Administrator users who can set up user accounts and grant user privileges.

The initial Administrator password is specified during the Oracle Application Testing Suite installation.

Separate User Databases for Oracle Load Testing and Oracle Test Manager

Separate user databases allow OATS Administrators to limit access to either Oracle Load Testing or Oracle Test Manager to specific users.

When you log into the Oracle Application Testing suite Administrator, you select the database to access using the login screen.

Oracle Load Testing Authentication

The Oracle Application Testing Suite Administrator provides the ability to optionally enable authentication for Oracle Load Testing. When Oracle Load Testing login is enabled, users must login to access Oracle Load Testing. The Oracle Load Testing authentication is enabled by default.

Log into the Oracle Application Testing Suite Administrator and select the OLT Database. When you add or edit a user profile, you can specify the type of access that is granted to the user.

Oracle Load Testing users can be granted the following type of access:

- **None** - the user cannot access Oracle Load Testing.
- **Full Control** - the user has full control in Oracle Load Testing.
- **View Only** - the user cannot run tests but can view reports.

Oracle Test Manager Authentication

The Oracle Application Testing Suite Administrator specifies the user authentication for Oracle Test Manager.

User authentication for Oracle Test Manager can be Oracle Test Manager database-based or configured to use Lightweight Directory Access Protocol (LDAP) server authentication. Select **Setup LDAP Config** from the **Tools** menu to specify the LDAP server configuration. LDAP authentication can be specified as follows:

- User+Password
- SSL+User+Password

Oracle Test Manager Role-Based Privileges

The Oracle Application Testing Suite Administrator specifies the role-based privileges for Oracle Test Manager. The Roles tab lists the defined roles and the permissions provided to each role.

The default roles are as follows:

- Developer
- Full Access
- Planner
- QA Engineer
- Tester
- Viewer

Additional roles can be defined and have the read, write, delete, execute, and set/override results permissions specified for that role. The Roles tab shows the permissions specified for the selected role in the **Permissions for <role>** section.

Permissions for <role> - displays the read, write, delete, execute, and set/override results permissions for this role for projects, test plans, requirements, tests, issues and reports.

- **Project** - displays the read, write, and delete permissions for projects for users assigned to this role.

- **Test Plan** - displays the read, write, and delete permissions for test plans for users assigned to this role.
- **Requirement**- displays the read, write, and delete permissions for requirements for users assigned to this role.
- **Test** - displays the read, write, delete, and execute permissions for tests for users assigned to this role.
- **Test Set** - displays the read, write, delete, and execute permissions for test sets for users assigned to this role.
- **Issue** - displays the read, write, and delete permissions for issues for users assigned to this role.
- **Public Reports** - displays the read, write, and delete permissions for public reports assigned to this role.
- **My Reports** - displays the read, write, and delete permissions for reports assigned to this role.

The default roles and permissions are as follows:

Permissions for Developer

- Read: Project, Requirement, Test, Issue, Public Reports, My Reports
- Write: Issue, Public Reports, My Reports
- Delete: Public Reports, My Reports
- Execute: <none>
- Set/Override results: <none>

Permissions for Full Access

- Read: Project, Test Plan, Requirement, Test, Issue, Public Reports, My Reports
- Write: Project, Test Plan, Requirement, Test, Issue, Public Reports, My Reports
- Delete: Project, Test Plan, Requirement, Test, Issue, Public Reports, My Reports
- Execute: Test
- Set/Override results: Test

Permissions for Planner

- Read: Project, Test Plan, Requirement, Test, Issue, Public Reports, My Reports
- Write: Project, Test Plan, Requirement, Test, Issue, Public Reports, My Reports
- Delete: Test Plan
- Execute: <none>
- Set/Override results: <none>

Permissions for QA Engineer

- Read: Project, Requirement, Test, Issue, Public Reports, My Reports
- Write: Project, Requirement, Test, Issue, Public Reports, My Reports
- Delete: Requirement, Test, Issue, Public Reports, My Reports
- Execute: Test
- Set/Override results: <none>

Permissions for Tester

- Read: Project, Requirement, Test, Issue, Public Reports, My Reports
- Write: Issue
- Delete: <none>
- Execute: Test
- Set/Override results: <none>

Permissions for Viewer

- Read: Project, Requirement, Test, Issue, Public Reports, My Reports
- Write: <none>
- Delete: <none>
- Execute: <none>
- Set/Override results: <none>

The default roles and permissions are customizable.

Oracle Test Manager Project-Based Access

The Oracle Application Testing Suite Administrator defines the testing projects available to users. Oracle Test Manager users can be assigned to specific projects to restrict project access to a defined set o f users and roles.

To specify the user access for specific projects:

1. Start the Oracle Application Testing Suite Administrator
2. Select the Oracle Test Manger database and login with administrator credentials.
3. Select the Projects tab.
4. Select the project in the projects section.
5. Select one or more users in the Assignments for <project name> section. Hold the Ctrl key to select more than one user.
6. Click Assign.
7. Select the roles to assign to each user in the project.
8. Click **OK**.

Oracle Test Manager Role-Based Field Security

The Oracle Application Testing Suite Administrator defines the fields that are used in Oracle Test Manager to describe issues, requirements, and tests. There are two types of fields that apply separately to requirements, tests, issues, and test runs:

Default fields - these are fields that come with the product such as Type, Priority, and Status for requirements. You cannot add or delete default fields. You can change whether they are required or enabled, and you can add, delete, or rename their associated options.

Custom fields - these are fields that you add that are specific to your product. For each custom field, you define its properties, then add options where appropriate.

Role-based field security settings specify which roles have read/write permissions and which roles have read-only permissions for specific fields. When you add or edit fields or field options, a **Permissions** link in the dialog box provide access to the Permissions settings for the selected field or field option.

Oracle OpenScript Security Features

This section has the following topics:

Overview

The Oracle OpenScript application includes the following security features:

- SSL Version preferences. The Oracle OpenScript HTTP Preferences provides an option for selecting the Secure Socket Layer version to use for the proxy server.
- HTTP Script Authentication during recording. Scripts can support Basic, NTLM v1, and Digest authentication during script recording.
- User Configurable Data Encryption (Obfuscate/Encrypt script data) in the General Preferences.
- Data Encryption (Obfuscate/Encrypt script data) for specific scripts from the Tools menu.
- Encryption Service error recovery. The Oracle OpenScript Error Recovery options specify the error recovery action when the password encryption service was not initialized or if Encrypting/Decrypting failed.
- Error recovery constants for the encryption service available to be used in Script programming code.
- Built-in encoding and encryption script functions. The Oracle OpenScript built-in script functions include encode/decode, encrypt/decrypt, and obfuscate/deobfuscate functions.
- The Oracle OpenScript built-in script functions for encode/decode, encrypt/decrypt, and obfuscate/deobfuscate functions can be used in the Function/Text Substitution Rules used for script correlation.

- OpenScript API security features. The OpenScript API supports data encryption and decryption methods in the OpenScript Application Programming Interface.
- Shared Data Service access credentials. The Shared Data Service module include preferences for setting the global shared data access credentials.
- Security extensions and attachments to Web Services scripts. Security extensions for username and password (either Password Text or Password Digest) can be added the Web Services scripts created using the Web Service module.
- Encryption properties supported with Agent command line settings.

SSL Version Preferences

The Oracle OpenScript HTTP Preferences provides an option for selecting the Secure Socket Layer version to use for the proxy server. When recording a secure site in the browser, the user only sees the Proxy Recorder's certificate not the secure web site's certificate. The Browser, Proxy Recorder, and Secure Server each have their own private and public keys which are used to encrypt/decrypt data.

To set the SSL version recording preferences for the HTTP module:

1. Start OpenScript.
2. Open or create a new HTTP script.
3. Select **OpenScript Preferences** from the **View** menu.
4. If necessary, expand the OpenScript node in the left pane.
5. If necessary, expand the Record node in the left pane.
6. Select HTTP.
7. If necessary, select the General tab.
8. Select the Secure Protocol:

   SSL: Use Secure Socket Layer protocol with the proxy server. OpenScript supports SSLv1.0, SSLv2.0, SSLv3.0 and TLSv1.0.

   TLSv1.2: Use Secure Socket Layer Transport Layer Security version 1.2.
9. Click **Apply** or **OK** when finished.

To set the SSL version playback preferences for the HTTP module:

1. Start OpenScript.
2. Open or create a new HTTP script.
3. Select **OpenScript Preferences** from the **View** menu.
4. If necessary, expand the OpenScript node in the left pane.
5. If necessary, expand the Playback node in the left pane.
6. Select HTTP.
7. If necessary, expand the Security section.
8. Select the Secure Protocol:

   SSL: Use Secure Socket Layer protocol with the proxy server. OpenScript supports SSLv1.0, SSLv2.0, SSLv3.0 and TLSv1.0.

   TLSv1.2: Use Secure Socket Layer Transport Layer Security version 1.2.

9. Click **Apply** or **OK** when finished.

HTTP Script Authentication During Recording

For web sites that require NTLM v1 Authentication, the browser opens a login dialog and waits for user input of the username and password. Once the username and password are entered, the browser sends the last request again with the Authentication information in http request header. However, OpenScript uses a proxy to record the http request and response and cannot get the NTLM username and password information that the user inputs into the browser because the password is not included in the content transfer on the network. For NTLM v1 Authentication, OpenScript opens a second login dialog for entering the username and password again during recording. OpenScript records an Authentication step automatically with username and encrypted password as the step before the navigate step.

To add authentication to an HTTP script manually:

1. Record an HTTP script.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the HTTP folder.
5. Select the **Authentication** node and click **OK**.
6. Enter the URL to access for authentication.
7. Enter a username.
8. Enter the password for the user. Passwords are encrypted using the Base-64 Crypt algorithm.
9. Click **OK** to add the Authentication node to the script tree.
10. In the Java Code view, the Authentication consists of the code executed in the `http.addAuthentication` procedure:

```
http.addAuthentication("http://testserver2", "username", decrypt("encrypted
password"));
```

User Configurable Data Encryption

The General preferences set the default encryption setting for all scripts.

To set the default encryption setting:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. If necessary, expand the OpenScript node in the left pane.
4. If necessary, expand the General node in the left pane.
5. Select Encryption and set the default encryption type.
6. Click **Apply** or **OK** when finished.

These preferences set the default encryption preferences. The resulting dialog presents the following fields:

Do not encrypt script data: When selected, passwords are stored and displayed as plain text in the script.

Obfuscate script data: When selected, passwords are obfuscated before storing and displaying in the script. Obfuscated passwords are hidden but *not* securely encrypted.

Encrypt script data: When selected, passwords are encrypted in the script. You will be asked to specify a password for the script if you create new scripts containing sensitive data or when opening encrypted scripts for playback.

Caution: HTTP scripts do not automatically obfuscate/encrypt sensitive script passwords.

Use the **Encryption** options on the **Tools** menu to set the encryption password for specific scripts.

Encryption Service Error Recovery

The Playback preferences set the default encryption setting for all scripts

To set the default encryption error recovery settings:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. If necessary, expand the OpenScript node in the left pane.
4. If necessary, expand the Playback node in the left pane.
5. Select Error Recovery.
6. If necessary, expand the General node in the right pane.
7. Set the encryption error recovery actions, as follows:

   Encryption Service Not Initialized - specifies the error recovery action when the password encryption service was not initialized.

   Encrypting/Decrypting Failed - specifies the error recovery action if an error occurs encrypting or decrypting a script.

   The default action is "Fail" script playback.
8. Click **Apply** or **OK** when finished.

Error Recovery Constants

Error recovery constants for the encryption service available to be used in Script programming code.

To add error recovery to a script:

1. Open or create a script project.
2. Select the script node where you want to add the log message.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the General node and select **Error Recovery Action**.

   Exception: Select the type of exception error. The list will vary depending upon the script type. The exception for the encryption services is "General - Encryption Service Not Initialized".

Action: Select the error recovery action: Fail, Warn, Ignore, Report, or Pause as follows:

- Fail: Report the error as failure and stop script execution.
- Warn: Report the error as a warning and continue script execution.
- Ignore: Ignore the error and continue script execution.
- ReportErrorAndContinue: Report the error to the results log and continue script execution.
- Pause: Pause playback and wait for user's decision to continue or abort script execution.

5. Click **OK**. The SetErrorRecovery node is added to the script tree.
6. In the Java Code view, the `setErrorRecovery(`*`scriptType.constant`*`, ErrorRecoveryAction.`*`action`*`);` method will be added to the script code:

```
setErrorRecovery(BasicErrorRecovery.ERR_ENCRYPTION_SERVICE_NOT_INITIALIZED,
ErrorRecoveryAction.Fail);
```

Encoding and Encryption Script Functions

Built-in script functions are used as variables to access or manipulate various type of data or parameters within OpenScript scripts. The built-in functions are indicated by the `@` sign and include the following types of functions:

- Encryption and encoding functions
- File functions
- Information functions
- Random functions
- Time functions

Built-in function are added to scripts using the Variable Substitution option available for the various script actions and parameters. Dialog boxes for script actions with parameter values include a Substitute Variable icon that opens the list of script variable and built-in `@` functions available to use a script parameters. Built-in `@` functions can also be added directly in the code view.

To access the built-in `@` function using the Tree view:

1. Start OpenScript.
2. Open or create a new script.
3. Select the script node where you want to add an action or substitute a parameter with a variable or built-in function.
4. Double-click the node if editing an existing node to open the parameters.
5. Click the Substitute Variable icon.
6. Expand the **@Functions** node.
7. Select the function to use and click **Finish**.
8. Click **OK**.

To add the built-in @ function using the Code view, add the function name and parameters to the code using the @ sign and double-curly brace transform syntax {{}}, similar to the following example:

```
{{@today(MM/dd/yyyy)}}
```

Use the eval() function to transform strings of data, substituting the values of variables where applicable. For example:

```
info(eval("{{@today(MM/dd/yyyy)}}"));
```

This section describes the built-in encoding and encryption @ functions and the required parameters. See the *Oracle Functional Testing OpenScript User's Guide* and the *Oracle Functional Testing OpenScript Programmer's Reference* for additional information about script functions.

@encode({{myVariable}}) - encode the data in myVariable converting binary content to hexadecimal string. It converts \ to \\ and any non-printable chars to \##. Returns null when input bytes are null. The allowable printable characters are:

- 9 (tab)
- 10 (line feed)
- 13 (carriage return)
- 32 through 126

@decode({{myVariable}}) - decode the encoded data in myVariable converting hexadecimal string to binary content. It returns the decoded byte array for the given hexadecimal string. Returns null when input bytes are null.

The following code examples show the @encode and @decode functions both with and without the eval() function:

```
getVariables().set("varEncodeDecode", "abc\\%2Fxyz\\t\\r\\n \\0D\\0A");
info("no eval encode = {{@encode({{varEncodeDecode}})}}");
//-or-
info(eval("eval encode = {{@encode({{varEncodeDecode}})}}"));

getVariables().set("varEncode", "{{@encode({{varEncodeDecode}})}}");
info("no eval decode = {{@decode({{varEncode}})}}");
//-or-
info(eval("eval decode = {{@decode({{varEncode}})}}"));
```

@encrypt({{myVariable}}) - encrypts the data in myVariable to protect sensitive data within scripts and makes the data non-human readable.

@decrypt({{myVariable}}) - decrypts the encrypted data in myVariable and makes the data human readable.

The following code examples show the @encrypt and @decrypt functions:

```
getVariables().set("myVariable", "test123");
info("variable text is {{myVariable}}");
getVariables().set("encrypt", "{{@encrypt({{myVariable}})}}");
info("encrypted = {{encrypt}}");
//-or-
info("encrypted = {{@encrypt({{myVariable}})}}");

getVariables().set("decrypt", "{{@decrypt({{encrypt}})}}");
info("decrypted = {{decrypt}}");
//-or-
```

```
info("decrypted = {{@decrypt({{encrypt}})}}");
```

`@obfuscate({{myVariable}})` - obfuscates the data in `myVariable` to protect sensitive data within scripts and makes the data non-human readable.

`@deobfuscate({{myVariable}})` - deobfuscates the obfuscated data in `myVariable` and makes the data human readable.

The following code examples show the `@obfuscate` and `@deobfuscate` functions:

```
getVariables().set("myVariable", "test123");
getVariables().set("obfuscate", "{{@obfuscate({{myVariable}})}}");
info("obfuscated = {{obfuscate}}");
//-or-
info("obfuscated = {{@obfuscate({{myVariable}})}}");

getVariables().set("deobfuscate", "{{@deobfuscate({{obfuscate}})}}");
info("deobfuscated = {{deobfuscate}}");
//-or-
info("deobfuscated = {{@deobfuscate({{obfuscate}})}}");
```

`@urlEncode({{myVariable}})` URL encode the value specified in `myVariable`. For example, the string, 'the file "abc" is in \root\etc', would be encoded as: the+file+%22abc%22+is+in+%5Croot%5Cetc. This function is only available for Load Testing scripts.

The following code example shows the `@urlEncode` function:

```
getVariables().set("myVariable", "the file \"abc\" is in \\root\\etc");
getVariables().set("urlEncode", "{{@urlEncode({{myVariable}})}}");
info("urlEncode = {{urlEncode}}");
//-or-
info("urlEncode = {{@urlEncode({{myVariable}})}}");
```

The above example returns the String `the+file+%22abc%22+is+in+%5Croot%5Cetc`.

`@xmlEncode({{myVariable}})` XML encode the value specified in `myVariable`. For example, the string, 'the file "abc" is in \root\etc', would be encoded as: the file "abc" is in \root\etc. This function is only available for Load Testing scripts.

The following code example shows the `@xmlEncode` function:

```
getVariables().set("myVariable", "the file \"abc\" is in \\root\\etc");
getVariables().set("xmlEncode", "{{@xmlEncode({{myVariable}})}}");
info("xmlDecode = {{xmlEncode}}");
//-or-
info("xmlDecode = {{@xmlEncode({{myVariable}})}}");
```

The above example returns the String `the file &quot;abc&quot; is in \root\etc`.

`@@xmlDecode({{myVariable}})` decodes the XML encoded value specified in `myVariable`. This function is only available for Load Testing scripts.

The following code example shows the `@xmlDecode` function:

```
getVariables().set("xmlDecode", "{{@xmlDecode({{xmlEncode}})}}");
info("xmlDecode = {{xmlDecode}}");
//-or-
info("xmlDecode = {{@xmlDecode({{xmlEncode}})}}");
```

The above example returns the String `the file "abc" is in \root\etc`.

Function/Text Substitution Rules

The Oracle OpenScript built-in script functions for encode/decode, encrypt/decrypt, and obfuscate/deobfuscate functions can be used in the Function/Text Substitution Rules used for script correlation.

To add a new Function/Text Substitution correlation rule:

1. Start OpenScript.
2. Open or create an HTTP script.
3. Select **OpenScript Preferences** from the **View** menu.
4. If necessary, expand the OpenScript node in the left pane.
5. If necessary, expand the Correlation node in the left pane.
6. Select HTTP in the left pane.
7. If necessary, expand the Web Default correlation library.
8. Click **Add Rule**.
9. Select Function/Text Substitution as the type.
10. Enter a name for the new rule.
11. Select the Function/Text in the **Source** tab.
12. Click the **Target** tab.
13. Specify a Regular Expression to indicate the location in the HTML source to replace with the Function/Text selected in the source tab. Use (( )) to indicate the part to replace. For example: `\?username=((.+?))&password=.+?` . See the *Oracle Functional Testing OpenScript User's Guide* for additional information about correlation rules.
14. Click **OK** when finished.

Encode/Decode API Features

The OpenScript API supports data encryption and decryption methods in the OpenScript Application Programming Interface.

To add encrypt and decrypt data in a script:

1. Open or create a script project.
2. Select the Java Code view and add the `encrypt()` and `decrypt()`methods with a String specifying the text to encrypt or decrypt, as follows:

```
String encrypted = encrypt("encrypt this string");
  info(encrypted);
String decrypted = decrypt(encrypted);
  info(decrypted);
```

Shared Data Service Access Credentials

To set Shared Data preferences:

1. Start OpenScript.
2. Open or create a new script.
3. Select **Script Properties** from the **Script** menu.

4. Select Modules in the left pane.
5. Select the **Shared Data** module and click **OK**.
6. Select **OpenScript Preferences** from the **View** menu.
7. Expand the OpenScript node and the Playback category.
8. Select **Shared Data**.
9. Set the Shared Data Preferences as follows:

   OATS Credentials: Specifies the authentication credentials to use to establish the communication between the shared queue and the Virtual User.

   - **Enable global shared data access credentials**: When selected, when selected, the shard data access credentials are enabled. Specify the **Address**, **User Name**, and **Password**.
   - **Address**: Specifies the address of the Oracle Load Testing for Web Application server to use for the shared data service.
   - **User name**: Specifies the user name to use for authentication. The default name is `oats` unless changed in the Oracle Application Testing Suite configuration.
   - **Password**: Specifies the password to use for authentication. This should be the same password specified in the Encryption setting of the General preferences if the **Encrypt script data** setting is selected.

   Actions on Shared Data: Specifies actions on shared data.

   - **Timeout**: Specifies the maximum number of seconds to wait for actions on shared data to occur before timing out.
10. Click **OK**.

Web Services Security Extensions and Attachments

The Web Services Module is an application module that supports testing of Web Services. Security extensions for username and password (either Password Text or Password Digest) can be added the Web Services scripts created using the Web Service module.

Adding Security Extensions

You can add security extensions to Web Services scripts.

To add security extensions to a Web Services script:

1. Create a Web Services script.
2. Expand the **Run** node.
3. Select the Web Services method node where you want to add the security and attachments.
4. Select the **Script** menu and then select **Other** from the **Add** menu.
5. Expand the HTTP node.
6. Select **Web Services Security Attachments** from the Web Services group and click **OK**.
7. If necessary click the **WS-Security** tab.

8. Enter a URL. If you selected a Web Services navigation node in the script tree, the URL will be automatically entered.
9. Select **User Username Token**
10. Enter the user name and password.

    Username: specifies the user name to use for the Username Token in the XML request.

    Password: specifies the password to use for the Username Token in the XML request.

    Confirm Password: confirms the password.
11. Select the password type: Password Text or Password Digest.

    Password Text: when selected, the password in the XML request is included as plain text. The URI attribute for the <wsse:Password> element is set to #PasswordText.

    Password Digest: when selected, the password is encrypted. The URI attribute for the <wsse:Password> element is set to #PasswordDigest.
12. Select or clear the **Add Created Header**, **Add Nonce** and **Add Timestamp** options.

    Add Created Header: when selected, a creation timestamp is included in the Username Token of the XML request for use in setting the server cache limit of used nonces.

    Add Nonce: when selected, a cryptographically random nonce value is included in the Username Token of the XML request to provide a countermeasure for replay attacks.

    Add Timestamp: when selected, a timestamp value is included in the Web Services security element of the XML request. The timestamp includes both Created and Expires elements. Specify the **Valid For** number of seconds.
13. Click **OK** to add the Security Attachment node to the script tree.

    In the Java Code view, the Security Attachment consists of the code executed in the `ws.addSecurityAttachments` method (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
  ws.security("userName", deobfuscate("password"),addCreatedHeader,
    addNonce, addTimestamp, validFor), null);
```

    If you add security and file attachments together, the `ws.addSecurityAttachments` method includes both the `ws.security` and `ws.attachments` methods (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
   ws.security("userName", deobfuscate("password"),true, true, true, 10),
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

Adding Attachments

You can add file attachments to Web Services scripts.

To add file attachments to a Web Services script:

1. Create a Web Services script.

2. Expand the **Run** node.
3. Select the Web Services method node where you want to add the security and attachments.
4. Select the **Script** menu and then select **Other** from the **Add** menu.
5. Select **Web Services Security Attachments** from the Web Services group.
6. If necessary click the **WS-Security** tab.
7. Enter a URL. If you selected a Web Services navigation node in the script tree, the URL will be automatically entered.
8. Click the **Attachments** tab.
9. Select the **Transfer Type**.
   - DEFAULT - uses the default transfer type specified by the Content-Type header.
   - SWA - Security SOAP Messages with Attachments
   - MTOM - SOAP Message Transmission Optimization Mechanism
   - DIME - Direct Internet Message Encapsulation
10. Click **Add**.
11. Enter the path and file name or click **Browse** to select a file.
12. If the Web Services method includes any Attachment Part object identifiers, select an Attachment Part from the list. If the Web Services method does not include any Attachment Part object identifiers, the list will be empty.
13. Click **OK** to add the Security Attachment node to the script tree.

    In the Java Code view, the Security Attachment consists of the code executed in the `ws.addSecurityAttachments` method (line breaks and spacing added for clarity) as follows:

```
ws.addSecurityAttachments("url", null,
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

    If you add security and file attachments together, the `ws.addSecurityAttachments` method includes both the `ws.security` and `ws.attachments` methods (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
   ws.security("userName", deobfuscate("password"),true, true, true, 10),
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

    The following example Web Services script method shows the `ws.addSecurityAttachments` method with a `ws.post` postdata method used to upload a file. The `ws.post` method specifies the SOAP Envelope postdata, Content-Type, and SOAP Action.

```
ws.method("upload");
{
  ws.addSecurityAttachments("http://myurl.com:8080/services/MTOMService",
    ws.security("username",deobfuscate("5blNah5kX/XuZnepYwInFw=="),
      true, true, true, 20),
    ws.attachments(AttachmentMechanism.MTOM,
```

```
        ws.attachment("C:\\OracleATS\\OFT\\test.txt", "<upload>776598931581")));

    ws.post(15, "http://myurl.com:8080/services/MTOMService",
      "<soapenv:Envelope
          xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
          xmlns:ser=\"http://service.interop.mtom.sample\">\r\n
        <soapenv:Header/>\r\n
        <soapenv:Body>\r\n
          <ser:upload>\r\n
            <!--Optional:-->\r\n
            <ser:fileName>string</ser:fileName>\r\n
            <!--Optional:-->\r\n
            <ser:contents>cid:776598931581</ser:contents>\r\n
          </ser:upload>\r\n
        </soapenv:Body>\r\n
      </soapenv:Envelope>",
      http.headers(http.header("Content-Type", "text/xml;charset=UTF-8",
          Header.HeaderAction.Modify),
        http.header("SOAPAction", "\"urn:upload\"",
          Header.HeaderAction.Modify)),
      true, null, null);
  }
  ws.endMethod();
```

Agent Command Line Encryption Properties

This section lists the command line options related to encryption settings. See the *Oracle Functional Testing OpenScript User's Guide* for additional information about command line options.

General Settings

The following table lists the General Encryption command line settings.

Setting	Description
`-ENCRYPTION_PROPERTIES_FILE` *path*`\encryption.properties`	Specify the full path and file name to use for encrypted password authentication. The file name is C:\Documents and Settings*username*\osworkspace\.metadata\.plugins\oracle.oats.scripting.utilities\encryption.properties.

Encryption Settings

The following table lists the Encryption command line settings:

Setting	Description
`-encrypt.password` *password* `-[`*child_alias*`].encrypt. password` *password* `-[`*grandchild_alias*`]@[`*child_alias*`].encrypt.password` *password*	Specify script passwords from the command line for encrypted scripts. Passwords specified as a parameter are different from playback settings, which are not kept or accessible by the `getSettings()` API, nor are they printable in scripts. `-encrypt.password` specifies the password of the master script. `-[`*child_alias*`].encrypt.password` specifies password of a child script. `-[`*grandchild_alias*`]@[`*child_alias*`].encrypt.password` specifies the password of a grandchild script. The hierachical alias names of child scripts are divided by the `@` character (the `@` character and period character are reserved characters of the alias name). Example usage: `-encrypt.password xxx -a.encrypt.password yyy -b@a.encrypt.password zzz` In this example, a "master" script has a child script "a", script 'a' has a grand-child script "b". ("a", "b" are the aliases of the child scripts).
`-encrypt.password.global`	Specify a global password that can be applied to any encrypted scripts. The `-encrypt.password` and `-`*alias*`.encrypt.password` takes precedence over `-encrypt.password.global`. The `-encrypt.password.global` password is applied only when the password of a script is not found in other arguments or is incorrect. Example usage: `-encrypt.password.global mypass`
`-encrypt.password.prompt`	Enables interactively entering password for encrypted scripts one-by-one using the console. Useful only when the CLI is started from the console. If specified, CLI will interactively ask to input the correct script password in the console (if the console exists). The password is not shown in the console. If you do not want to input the password of a script, press ENTER to skip to next encrypted script. Example usage: `-encrypt.password.prompt`

Setting	Description
`-streamProperties`	Enables a way of passing arguments to the CLI by reading from STDIN. If specified, the CLI reads arguments from STDIN until `\endStream` plus a line separator is entered. The line separator is any one of line feed (`\n`), carriage return (`\r`), or carriage return followed immediately by linefeed. The charset used to decode the bytes from STDIN can be specified using the `-streamCharset` argument. The default charset is UTF-8. `-streamProperties` provides a more secure way of passing arguments to CLI. It can be used to pass script passwords or to pass any other CLI arguments. If the CLI is started from a console, CLI will wait for arguments followed by `\endStream\r\n`. The arguments in `-streamProperties` will override the same arguments from the `-propertiesPath` and CLI arguments. The `-propertiesPath` will override the same arguments in the CLI. Example usage: `-streamProperties -streamCharset UTF-16LE [STDIN] -settingIds settingValues [...] \endStream\r\n`
`-streamCharset charset`	Specify a charset that will be used to decode the byte array read from STDIN to a string. Used only when `-streamProperties` is specified. If an illegal charset is specified, `-streamCharset` shows a warning message and uses UTF-8 instead. Example usage: `-streamProperties -streamCharset UTF-16LE`

Oracle Load Testing Security Features

This section has the following topics:

- Overview
- Password-Based Login
- Agent Authentication Manger
- SSL Version Preferences
- Encryption Error Recovery Settings

Overview

The Oracle Load Testing application includes the following security features:

- Password-based login. The login requires username and password configured using the Oracle application Testing Suite Administrator application.
- Agent Authentication Manager. The Oracle Load Testing Agent Authentication Manager lets you define authentication profiles for multiple load testing agent machines.

- SSL Version preferences. The Oracle Load Testing Java Client Preferences provides an option for selecting the Secure Socket Layer version to use for the proxy server.
- Encryption Service error recovery. The Oracle Load Testing OpenScript Error Recovery options specify the error recovery action when the password encryption service was not initialized of if Encrypting/Decrypting failed.

Password-Based Login

The Oracle Load Testing application user login credentials are configured using the Oracle Application Testing Suite Administrator. The Oracle Load Testing login feature is enabled as the default.

Agent Authentication Manger

The Oracle Load Testing Agent Authentication Manager lets you define authentication profiles for multiple load testing agent machines.

To start the Oracle Load Testing Agent Authentication Manager:

1. Select **Oracle Application Testing Suite** from the **Start** menu.
2. Expand the **Tools** folder and select **Oracle Load Testing Agent Authentication Manager**.
3. Click **New** to create new Authentication Profile.
4. Enter the Hostname, port, username, and password. Click **Help** for additional information.
5. Click **Save**.
6. Repeat steps 3-5 for each agent machine.

SSL Version Preferences

The Oracle Load Testing Java Client Preferences provides an option for selecting the Secure Socket Layer version to use for the proxy server. When recording a secure site in the browser, the user only sees the Proxy Recorder's certificate not the secure web site's certificate. The Browser, Proxy Recorder, and Secure Server each have their own private and public keys which are used to encrypt/decrypt data.

To set the SSL version preferences as the default for all scenarios:

1. Start the Oracle Load Testing Application.
2. Log in if required.
3. Select **Options** from the **Tools** menu.
4. Select **Scenario Defaults** in the left pane.
5. Set the **Secure Protocol** setting in the **Java Client Preferences** section.

To set the SSL version preferences for a specific Virtual User profile (script):

1. Start the Oracle Load Testing Application.
2. Log in if required.
3. Click the **Build Scenarios** tab.
4. Add a script to the **Configure parameters of the scenario** list.

5. Click the **Configure all parameters** button.
6. Set the **Secure Protocol** setting in the **Java Client Preferences** section.

Encryption Error Recovery Settings

The Oracle Load Testing OpenScript Error Recovery options specify the error recovery action when the password encryption service was not initialized of if Encrypting/Decrypting failed.

To set the OpenScript Error Recovery options as the default for all scenarios:

1. Start the Oracle Load Testing Application.
2. Log in if required.
3. Select **Options** from the **Tools** menu.
4. Select **Scenario Defaults** in the left pane.
5. Set the **Encryption Service not Initialized** setting in the **OpenScript Error Recovery - General** section.
6. Set the **Encrypting/Decrypting Failed** setting in the **OpenScript Error Recovery - General** section.

To set the SSL version preferences for a specific Virtual User profile (script):

1. Start the Oracle Load Testing Application.
2. Log in if required.
3. Click the B**uild Scenarios** tab.
4. Add a script to the **Configure parameters of the scenario** list.
5. Click the **Configure all parameters** button.
6. Set the **Encryption Service not Initialized** setting in the **OpenScript Error Recovery - General** section.
7. Set the **Encrypting/Decrypting Failed** setting in the **OpenScript Error Recovery - General** section.

Oracle Test Manager Security Features

This section has the following topics:

Overview

The Oracle Test Manager application includes the following security features:

- Password-based login. The user login can be Oracle Application Testing Suite database-based or LDAP server based. The users and LDAP server configuration are specified in the Oracle Application Testing Suite Administrator application.

- Support for running encrypted OpenScript scripts. OpenScript scripts can be used as test in the Oracle Test Manager application. Oracle Test Manager supports running password encrypted OpenScript scripts.

Password-Based Login

The user login can be Oracle Application Testing Suite database-based or LDAP server based. The users and LDAP server configuration are specified in the Oracle Application Testing Suite Administrator application.

Oracle Test Manager Role-Based Privileges

The Oracle Application Testing Suite Administrator specifies the role-based privileges for Oracle Test Manager. The Roles tab lists the defined roles and the permissions provided to each role. See Oracle Test Manager Role-Based Privileges in the OATS Administrator Security Features section for additional information.

Oracle Test Manager Project-Based Access

The Oracle Application Testing Suite Administrator defines the testing projects available to users. Oracle Test Manager Users can be assigned to specific projects to restrict project access to a defined set o f users and roles. See Oracle Test Manager Project-Based Access in the OATS Administrator Security Features section for additional information.

Encrypted OpenScript Scripts

OpenScript scripts can be used as test in the Oracle Test Manager application. Oracle Test Manager supports running password encrypted OpenScript scripts.

To add an Oracle OpenScript test:

1. Start Oracle test Manager and login.
2. Select the **Tests** tab.
3. Click the **Add** button.
4. Enter a name for the test.
5. Select **Oracle OpenScript** in the Type field.
6. Click **Find** to display the Select Script dialog box.
7. Select the Repository and Workspace.
8. Select the Oracle OpenScript script that you want to add.
9. Enter command line settings and OpenScript password if required. See the Command line settings in the *Oracle Functional Testing OpenScript User's Guide* or OpenScript online help for the available settings.
10. Click **OK**.
11. Select the owner and priority and enter any descriptive information in the **Functionality** and **Description** fields.
12. Click **Save**.

13. The OpenScript test can be executed in the Test Execution tab. See the *Oracle Test Manager User's Guide* or online help for more information about executing tests in Oracle Test Manager.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Oracle Application Testing Suite Security Guide Release 13.3.0.1
E74489-05