Siebel

Testing Siebel Business Applications Guide

December 2025

Siebel
Testing Siebel Business Applications Guide

December 2025

Part Number: F84089-06

Authors: Siebel Information Development Team

Contents

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at *https://docs.oracle.com/*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an email to: *oracle_fusion_applications_help_ww_grp@oracle.com*.

1 What's New in This Release

What’s New in This Release

This chapter tracks the changes in the documentation. It includes the following topics:

What's New in Testing Siebel Business Applications, Siebel CRM 25.12 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
StartLoop and EndLoop Logical Looping Constructs	New topic. This topic describes the new how to add loop constructs to repeat a contiguous set of Test Steps.

What's New in Testing Siebel Business Applications, Siebel CRM 25.8 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Test Scripts Dashboard	New topic. This topic describes the new View displaying Test Scripts, Data Sets and Master Suites summary data and chart.
Test Execution Dashboard	New topic. This topic describes the new View displaying Test Runs, Passed, Failed, Test Results pending Issue Type , and Issue Type summary pie chart.

Topic	Description
Executing the Automation Batch Run *Creating Test Results* *Configuring the Test Run*	Modified topics. The changes describe the enhancements to Test Execution View, about using Test Execution Profile, Failed Test Results button, and Parameters applet changes. Also a new section *Test Execution Profile* is added, it describes the new View for managing Test Execution parameters using Test Execution Profiles.
Test Results View	New topic. New fields Result Log, Issue, and Comments, are added to Test Results.
Create a Test Execution Record	Modified topics. The changes describe REST API example using Test Execution Profile.
Create Test Passes for a Test Execution Record	Modified topics. The changes describe automatic creation of Test Passes.
Installing Appium	New topic. Describes Installation steps.
Installing Xcode on the Xcode iOS Simulator	Modified topics. The changes describe new version numbers and changed steps.
Offline uploading of Test Results	New topic. This topic describes the utility provided as part of DISA to upload test results to Siebel.

What's New in Testing Siebel Business Applications, Siebel CRM 25.3 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Test Execution without Jenkins	New topic. This topic describes how to use the command for test execution without Jenkins.

What's New in Testing Siebel Business Applications, Siebel CRM 24.8 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Executing the Automation Batch Run	Modified topic. New subsection, *Copying a Test Execution record* added.

What's New in Testing Siebel Business Applications, Siebel CRM 24.7 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Condition Expression for Test Steps	New topic. Describes how to use Condition field in Test Steps Applet in Test Script View, to determine at run time if the Test Step should be run or if it should be skipped.

What's New in Testing Siebel Business Applications, Siebel CRM 23.7 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Dynamic Data Selection from Data Set	New topic. Describes how to use Condition field in Data Sets to determine at run time if a particular row of Data set is selected for automated Test Script iteration or if the row is to be skipped.
Associating a Data Set with a Test Set	New topic. Describes how to associate a Data Set to a Test Set, in order to use it with the Test Scripts associated to Test Set .
Copying a Test Set	New topic. Describes how to copy a Test Set replicating Test Script associations, Data Set association and iteration type .
Viewing Test Sets associated to a Data Set	New topics. Describes how to view Master Suites associated to a given Test Set, and to view Test Sets associated to a given Data Set.
Viewing Master Suites associated to a Test Set	
Configuring the Test Run	Modified topic. Notify and Run Reference fields are added to table. Notify describes how to enable Notifications for Status change of a Test Execution record. Run Reference describes how to redirect a Test Execution record to a given Jenkins node or a pool of nodes.
Configuring the Siebel Test Execution Job	Modified topic. Parameters Field and Description added to the table.
Setting Up the Jenkins Server	Modified Topics. Jenkins related steps corrected to remove obsolete steps and Multi job plug-in. Updated for --runReference parameter usage.
Setting Up and Configuring the Siebel Test Execution Plugin	Modified Topics. Jenkins related steps corrected to remove obsolete steps and Multi job plug-in. Updated for --runReference parameter usage.
Configuring the Siebel Test Execution Job	Modified Topics. Jenkins related steps corrected to remove obsolete steps and Multi job plug-in. Updated for --runReference parameter usage.

What's New in Testing Siebel Business Applications, Siebel CRM 22.11 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Configuring Multiple Batch Runs	New topic. Describes how to set up *parallel execution* for Siebel test automation so that you will be able to execute multiple batch runs on the same client at the same time.
InvokeREST	New topic. Describes how to use the InvokeREST keyword to test and issue REST API calls from within a test script (for example, to execute the API, verify and/or use the response) in desktop applications.
ToolsConfig	Modified topic. The information in this topic has been updated.

What's New in Testing Siebel Business Applications, Siebel CRM 22.4 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
About Report Generation	New topic. Reports are now generated in JSON format (in addition to existing HTML format) making it easy for report results to be consumed by other tools such as CI/CD pipelines, test management tools, and so on. New reports built using the JSON format provide a consolidated snapshot of results and a detailed failure analysis. You can also manually generate a combined results report.
Generating a Combined Report	New topic. Describes how to manually generate a combined results report from multiple Report.html files.

What's New in Testing Siebel Business Applications, Siebel CRM 21.9 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Siebel Test Automation Folder	Modified topic. The Extensions folder is a placeholder for CustomExtension JAR files.
Extending Keyword Automation Capabilities	New topic. You can use the CustomExtension keyword to enable support for custom requirements, such as SSO.
Plugin Configurations	Modified topic. You can use the DetailedReport check box (on the Popup that appears when you click Play) to specify whether or not to capture detailed test results and screenshots during unit mode/ single test script playback.

Topic	Description
Configuring the Siebel Test Execution Job	Modified topic. When configuring Siebel Test Execution jobs, you can use the Parameters field to specify whether or not to generate reports and capture screenshots during the test script automation batch run.
Keywords Reference	Modified topic. The note about importing test scripts to the database before using any keywords is new.
CustomExtension	New topic. CustomExtension is a new keyword, which runs a custom extension JAR file.
Reports	Modified topic. You can control whether or not to generate reports and capture screenshots and test results during test script execution.

What's New in Testing Siebel Business Applications, Siebel CRM 21.8 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Implementing Siebel Open UI Keyword Automation Testing *Impact of Usage Pattern Tracking Enhancements in Siebel CRM 18.9 Update*	Modified topic. Enabling UPT is not compatible with test automation.
Browser Configuration Settings	Modified topic. The geckodriver is required to run the test scripts in Firefox. Microsoft Edge is supported for running test scripts.
Creating Test Results	Modified topic. Test Plans must be associated with Test Sets before scheduling a Test Execution.
Setting Up the Jenkins Server	Modified topic. Describes how to install and set up the Jenkins server.
Setting Up the Automation Adapter	Modified topic. Added a new subtopic about deleting UPT action sets, runtime events and system preferences.

What's New in Testing Siebel Business Applications, Siebel CRM 21.6 Update

The following information lists the changes in this version of the documentation to support this release of the software.

Topic	Description
Setting Up the Automation Adapter *Enabling Automation for Developer Web Client*	Modified topics. As of Siebel CRM 21.2 Update, the **`applicationcontainer`** directory has been replaced by two directories, as follows: • **`applicationcontainer_external`** (for Siebel Application Interface) • **`applicationcontainer_internal`** (for all other Siebel Enterprise components) For more information, see *Siebel Installation Guide* .

2 Overview of Testing Siebel Applications

Overview of Testing Siebel Applications

This chapter provides an overview of the reasons for implementing testing in software development projects, and introduces a methodology for testing Oracle's Siebel Business Applications with descriptions of the processes and types of testing that are used in this methodology. This chapter includes the following topics:

About Testing Siebel Business Applications

This guide introduces and describes the processes and concepts of testing Siebel Business Applications. It is intended to be a guide for best practices for Oracle customers currently deploying or planning to deploy Siebel Business Applications for Siebel CRM 17.x or later.

Although job titles and duties at your company may differ from those described in the following table, the audience for this guide consists primarily of employees in these categories:

Job Title	Description
Application Testers	Testers responsible for developing and executing tests. Functional testers focus on testing application functionality, while performance testers focus on system performance.
Business Analysts	Analysts responsible for defining business requirements and delivering relevant business functionality. Business analysts serve as the advocate for the business user community during application deployment.
Business Users	Actual users of the application. Business users are the customers of the application development team.
Database Administrators	Administrators who administer the database system, including data loading, system monitoring, backup and recovery, space allocation and sizing, and user account management.
Functional Test Engineers	Testers with the responsibility of developing and executing manual and automated testing. Functional test engineers create test cases and automate test scripts, maintain regression test library and report issues and defects.

Job Title	Description
Performance Test Engineers	Testers with the responsibility of developing and executing automated performance testing. Performance test engineers create automated test scripts, maintain regression test scripts and report issues and defects.
Project Managers	Manager or management team responsible for planning, executing, and delivering application functionality. Project managers are responsible for project scope, schedule, and resource allocation.
Siebel Application Developers	Developers who plan, implement, and configure Siebel business applications, possibly adding new functionality.
Siebel System Administrators	Administrators responsible for the whole system, including installing, maintaining, and upgrading Siebel business applications.
Test Architect	Working with the Test Manager, an architect designs and builds the test strategy and test plan.
Test Manager	Manages the day-to-day activities, testing resources, and test execution. Manages the reporting of test results and the defect management process. The Test Manager is the single point of contact (POC) for all testing activities.

Note: On simple projects, the Test Architect and Test Manager are normally combined into a single role.

How This Guide Is Organized

This book describes the processes for planning and executing testing activities for Siebel business applications. These processes are based on best practices and proven test methodologies. You use this book as a guide to identify what tests to run, when to run tests, and who to involve in the quality assurance process.

The first two chapters of this book provide an introduction to testing and the test processes. You are encouraged to read *Overview of Testing Siebel Applications*, which describes the relationships between the seven high-level processes. The chapters that follow describe a specific process in detail. In each of these chapters, a process diagram is presented to help you to understand the important high-level steps. You are encouraged to modify the processes to suit your specific situation.

Depending on your role, experience, and current project phases you will use the information in this book differently. Here are some suggestions about where you might want to focus your reading:

- **Test manager.** At the beginning of the project, review Chapters 2 through 8 to understand testing processes.
- **Functional testing.** If you are a functional tester focus on Chapters 3 through 7 and 9. These chapters discuss the process of defining, developing, and executing functional test cases.
- **Performance testing.** If you are a performance tester focus on Chapters 3, 4, 7, and 10. These chapters describe the planning, development, and execution of performance tests.

At certain points in this book, you will see information presented as a best practice. These tips are intended to highlight practices proven to improve the testing process.

Additional Resources

- American Society of Quality: *http://www.asq.org/pub/sqp*.
- Bitpipe http: *http://www.bitpipe.com/tlist/Software-Testing.html*
- Economic Impact of Inadequate Infrastructure for Software Testing: *http://www.nist.gov/director/prog-ofc/report02-3.pdf*
- International Federation for Information Processing: *http://www.ifip.or.at/* (click on the "Search IFIP" link)
- StickyMinds: *http://www.stickyminds.com/testing.asp*

Introduction to Application Software Testing

Testing is a key component of any application deployment project. The testing process determines the readiness of the application. Therefore, it must be designed to adequately inform deployment decisions. Without well-planned testing, project teams may be forced to make under-informed decisions and expose the business to undue risk. Conversely, well-planned and executed testing can deliver significant benefit to a project including:

- **Reduced deployment cost.** Identifying defects early in the project is a critical factor in reducing the total cost of ownership. Research shows that the cost of resolving a defect increases dramatically in later deployment phases. A defect discovered in the requirements definition phase as a requirement gap can be a hundred times less expensive to address than if it is discovered after the application has been deployed. Once in production, a serious defect can result in lost business and undermine the success of the project.
- **Higher user acceptance.** User perception of quality is extremely important to the success of a deployment. Functional testing, usability testing, system testing, and performance testing can provide insights into deficiencies from the users' perspective early enough so that these deficiencies can be corrected before releasing the application to the larger user community.
- **Improved deployment quality.** Hardware and software components of the system must also meet a high level of quality. The ability of the system to perform reliably is critical in delivering consistent service to the users or customers. A system outage caused by inadequate system resources can result in lost business. Performance, reliability, and stress testing can provide an early assessment of the system to handle the production load and allow IT organizations to plan accordingly.

Inserting testing early and often is a key component to lowering the total cost of ownership. Software projects that attempt to save time and money by lowering their initial investment in testing find that the cost of *not* testing is much greater. Insufficient investment in testing may result in higher deployment costs, lower user adoption, and failure to achieve business returns.

Best Practice

Test early and often. The cost of resolving a defect when detected early is much less then resolving the same defect in later project stages. Testing early and often is the key to identifying defects as early as possible and reducing the total cost of ownership.

Application Software Testing Methodology

The processes described in this book are based on common test definitions for application software. These definitions and methodologies have been proven in customer engagement, and demonstrate that testing must occur throughout the project lifecycle.

Common Test Definitions

There are several common terms used to describe specific aspects of software testing. These testing classifications are used to break down the problem of testing into manageable pieces. Here are some of the common terms that are used throughout this book:

- **Business process testing.** Validates the functionality of two or more components that are strung together to create a valid business process.
- **Data conversion testing.** The testing of converted data used within the Siebel application. This is normally performed before system integration testing.
- **Functional testing.** Testing that focuses on the functionality of an application that validates the output based on selected input that consists of Unit, Module and Business Process testing.
- **Interoperability testing.** Applications that support multiple platforms or devices need to be tested to verify that every combination of device and platform works properly.
- **Negative testing.** Validates that the software fails appropriately by inputting a value known to be incorrect to verify that the action fails as expected. This allows you to understand and identify failures. By displaying the appropriate warning messages, you verify that the unit is operating correctly.
- **Performance testing.** This test is usually performed using an automation tool to simulate user load while measuring the system resources used. Client and server response times are both measured.
- **Positive testing.** Verifies that the software functions correctly by inputting a value known to be correct to verify that the expected data or view is returned appropriately.
- **Regression testing.** Code additions or changes may unintentionally introduce unexpected errors or regressions that did not exist previously. Regression tests are executed when a new build or release is available to make sure existing and new features function correctly.
- **Reliability testing.** Reliability tests are performed over an extended period of time to determine the durability of an application as well as to capture any defects that become visible over time.
- **Scalability testing.** Validates that the application meets the key performance indicators with a predefined number of concurrent users.
- **Stress testing.** This test identifies the maximum load a given hardware configuration can handle. Test scenarios usually simulate expected peak loads.
- **System integration testing.** This is a complete system test in a controlled environment to validate the end-to-end functionality of the application and all other interface systems (for example, databases and third-party systems). Sometimes adding a new module, application, or interface may negatively affect the functionality of another module.
- **Test case.** A test case contains the detailed steps and criteria (such as roles and data) for completing a test.
- **Test script.** A test script is an automated test case.
- **Unit testing.** Developers test their code against predefined design specifications. A unit test is an isolated test that is often the first feature test that developers perform in their own environment before checking changes

into the configuration repository. Unit testing prevents introducing unstable components (or units) into the test environment.

- **Usability testing.** User interaction with the graphical user interface (GUI) is tested to observe the effectiveness of the GUI when test users attempt to complete common tasks.
- **User acceptance test (UAT).** Users test the complete, end-to-end business processes, verifying functional requirements (business requirements).

Modular and Iterative Methodology

An IT project best practice that applies to both testing and development is to use a modular and incremental approach to develop and test applications to detect potential defects earlier rather than later. This approach provides component-based test design, test script construction (automation), execution and analysis. It brings the defect management stage to the forefront, promoting communication between the test team and the development team. Beginning the testing process early in the development cycle helps reduce the cost to fix defects.

This process begins with the test team working closely with the development team to develop a schedule for the delivery of functionality (a drop schedule). The test team uses this schedule to plan resources and tests. In the earlier stages, testing is commonly confined to unit and module testing. After one or more drops, there is enough functionality to begin to string the modules together to test a business process.

After the development team completes the defined functionality, they compile and transfer the Siebel application into the test environment. The immediate functional testing by the test team allows for early feedback to the development team regarding possible defects. The development team can then schedule and repair the defects, drop a new build of the Siebel application, and provide the opportunity for another functional test session after the test team updates the test scripts as necessary.

Best Practice

Iterative development introduces functionality to a release in incremental builds. This approach reduces risk by prompting early communication and allowing testing to occur more frequently and with fewer changes to all parts of the application.

Continuous Application Lifecycle

One deployment best practice is the continuous application lifecycle. In this approach, application features and enhancements are delivered in small packages on a continuous delivery schedule. New features are considered and scheduled according to a fixed release schedule (for example, once every quarter). This model of phased delivery provides an opportunity to evaluate the effectiveness of prebuilt application functionality, minimizes risk, and allows you to adapt the application to changing business requirements.

Continuous application lifecycle incorporates changing business requirements into the application on a regular timeline, so the business customers do not have a situation where they become locked into functionality that does not quite meet their needs. Because there is always another delivery date on the horizon, features do not have to be rushed into production. This approach also allows an organization to separate multiple, possibly conflicting change activities. For example, the upgrade (repository merge) of an application can be separated from the addition of new configuration.

Best Practice

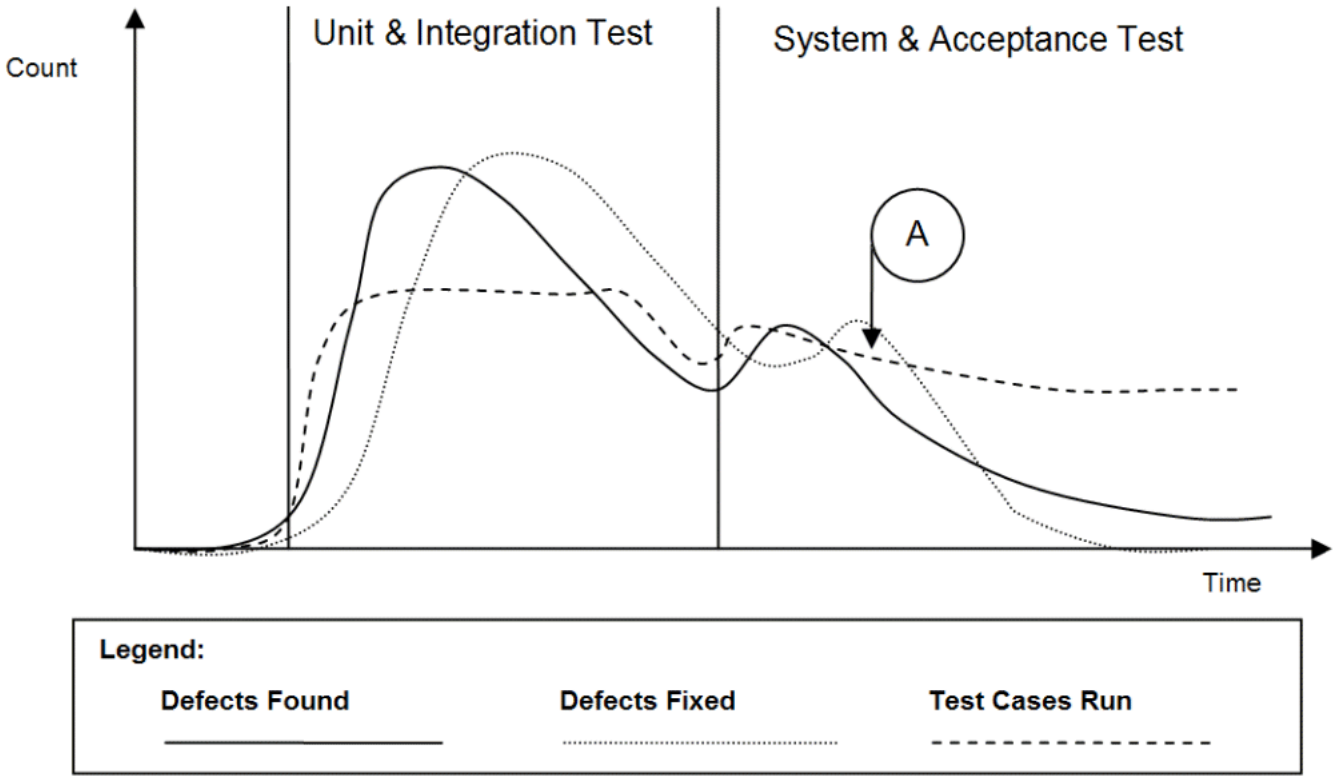
The continuous application lifecycle approach to deployment allows organizations to reduce the complexity and risk on any single release and provides regular opportunities to enhance application functionality.

Testing and Deployment Readiness

The testing processes provide crucial inputs for determining deployment readiness. Determining whether or not an application is ready to deploy is an important decision that requires clear input from testing.

Part of the challenge in making a good decision is the lack of well-planned testing and the availability of testing data to gauge release readiness. To address this, it is important to plan and track testing activity for the purpose of informing the deployment decision. In general, you can measure testing coverage and defect trends, which provide a good indicator of quality. The following are some suggested analyses to compile:

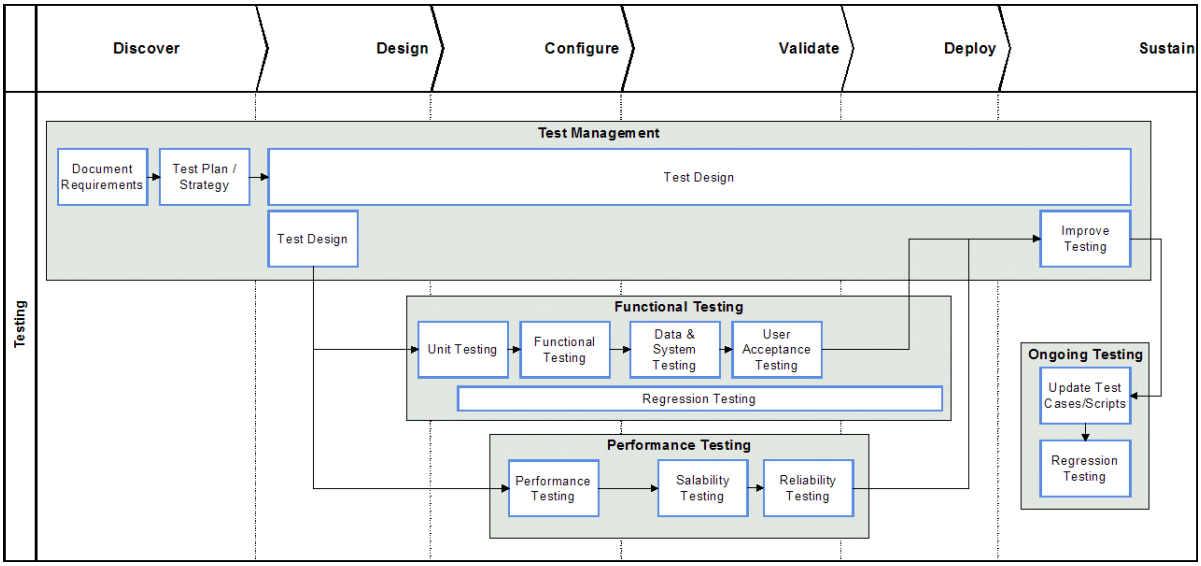
- For each test plan, the number and percentage of test cases passed, in progress, failed, and blocked. This data illustrates the test objectives that have been met, versus those that are in progress or at risk.
- Trend analysis of open defects targeted at the current release for each priority level.
- Trend analysis of defects discovered, defects fixed, and test cases executed. Application convergence (point A in the following image) is demonstrated by a slowing of defect discovery and fix rates, while maintaining even levels of test case activity.



Testing is a key input to the deployment readiness decision. However it is not the only input to be considered. You must consider testing metrics with business conditions and organizational readiness.

Overview of the Siebel Testing Process

Testing processes occur throughout the implementation lifecycle and are closely linked to other configuration, deployment, and operations processes. The following image presents a high-level map of the different testing processes.



Each of the seven testing processes described in this book are highlighted in this image, and described briefly in the following subtopics:

- *Plan Testing Strategy*
- *Design and Develop Tests*
- *Execute Siebel Functional Tests*
- *Execute System Integration Tests*
- *Execute Acceptance Tests*
- *Execute Performance Tests*
- *Improve and Continue Testing*

Plan Testing Strategy

The test planning process makes sure that the testing performed is able to inform the deployment decision process, minimize risk, and provide a structure for tracking progress. Without proper planning many customers may perform either too much or too little testing. The process is designed to identify key project objectives and develop plans based on those objectives.

It is important to develop a testing strategy early, and to use effective communications to coordinate among all stakeholders of the project.

For more information, see *Plan Testing Strategy*.

Design and Develop Tests

In the test design process, the high-level test cases identified during the planning process are developed in detail (step-by-step). Developers and testers finalize the test cases based on approved technical designs. The written test cases can also serve as blueprints for developing automated test scripts. Test cases should be developed with strong participation from the business analyst to understand the details of usage, and less-common use cases.

Design evaluation is the first form of testing, and often the most effective. Unfortunately, this process is often neglected. In this process, business analysts and developers verify that the design meets the business unit requirements. Development work should not start in earnest until there is agreement that the designed solution meets requirements. The business analyst who defines the requirements should approve the design.

Preventing design defects or omissions at this stage is more cost effective than addressing them later in the project. If a design is flawed from the beginning, the cost to redesign after implementation can be high.

For more information, see *Design and Develop Tests*.

Execute Siebel Functional Tests

Functional testing is focused on validating the Siebel business application components of the system. Functional tests are performed progressively on components (units), modules, and business processes in order to verify that the Siebel application functions correctly. Test execution and defect resolution are the focus of this process. The development team is fully engaged in implementing features, and the defect-tracking process is used to manage quality.

For more information, see *Execute Siebel Functional Tests*.

Execute System Integration Tests

System integration testing verifies that the Siebel application validated earlier, integrates with other applications and infrastructure in your system. Integration with various back-end, middleware, and third-party systems are verified. Integration testing occurs on the system as a whole to make sure that the Siebel application functions properly when connected to related systems, and when running along side system-infrastructure components.

For more information, see *Execute System Integration and Acceptance Tests*.

Execute Acceptance Tests

Acceptance testing is performed on the complete system and is focused on validating support for business processes, as well as verifying acceptability to the user community from both the lines of business and the IT organization. This is typically a very busy time in the project, when people, process, and technology are all preparing for the rollout.

For more information, see *Execute System Integration and Acceptance Tests*.

Execute Performance Tests

Performance testing validates that the system can meet specified key performance indicators (KPIs) and service levels for performance, scalability, and reliability. In this process, tests are run on the complete system simulating expected loads and verifying system performance.

For more information, see *Execute Performance Tests*.

Improve and Continue Testing

Testing is not complete when the application is rolled out. After the initial deployment, regular configuration changes are delivered in new releases. In addition, Oracle delivers regular maintenance and major software releases that may need to be applied. Both configuration changes and new software releases require regression testing to verify that the quality of the system is sustained.

The testing process should be evaluated after deployment to identify opportunities for improvement. The testing strategy and its objectives should be reviewed to identify any inadequacies in planning. Test plans and test cases should be reviewed to determine their effectiveness. Test cases should be updated to include testing scenarios that were discovered during testing and were not previously identified, to reflect all change requests, and to support software releases.

For more information, see *Improve and Continue the Testing Process* .

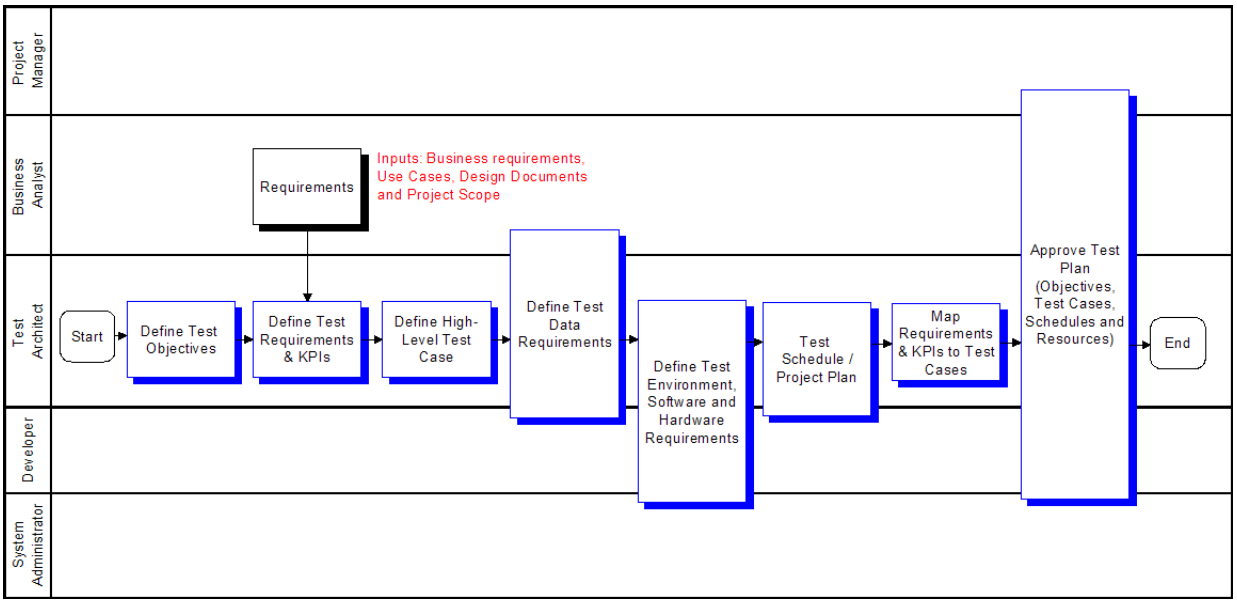
3 Plan Testing Strategy

Plan Testing Strategy

This chapter describes the process of planning your tests. It includes the following topics:

Overview of Test Planning

The objective of the test planning process is to create the strategy and tactics that provide the proper level of test coverage for your project. The following image illustrates one example of the test planning process.



As shown in this image:

1. The inputs to the test planning process are the business requirements and the project scope.
2. The outputs or deliverables for the test planning process include the following:
    - **Test objectives.** The high-level objectives for a quality release. The test objectives are used to measure project progress and deployment readiness. Each test objective has a corresponding business or design requirement.
    - **Test plans.** The test plan is an end-to-end test strategy and approach for testing the Siebel application. A typical test plan contains the following sections:

    - **Strategy, milestones, and responsibilities.** Set the expectation for how to perform testing, how to measure success, and who is responsible for each task
    - **Test objectives.** Define and validate the test goals, objectives, and scope
    - **Approach.** Outlines how to and when to perform testing
    - **Entrance and exit criteria.** Define inputs required to perform a test and the success criteria for passing a test
    - **Results reporting.** Outlines the type and schedule of reporting
  - **Test cases.** A test plan contains a set of test cases. Test cases are detailed, step-by-step instructions about how to perform a test. The instructions should be specific and repeatable by anyone who typically performs the tasks being tested. In the planning process, you identify the number and type of test cases to be performed.
  - **Definition of test environments.** The number, type, and configuration of test environments must be defined. Clear entry and exit criteria for each environment should be defined.

Test Objectives

The first step in the test planning process is to document the high-level test objectives. The test objectives provide a prioritized list of verification or validation objectives for the project. You use this list of objectives to measure testing progress, and verify that testing activity is consistent with project objectives.

Test objectives can typically be grouped into the following categories:

- **Functional correctness.** Validation that the application correctly supports the required business processes and transactions. List all of the business processes that the application is required to support. Also list any standards for which there is required compliance.
- **Authorization.** Verification that actions and data are available only to those users with correct authorization. List any key authorization requirements that must be satisfied, including access to functionality and data.
- **Service level.** Verification that the system will support the required service levels of the business. This includes system availability, load, and responsiveness. List any key performance indicators (KPIs) for service level, and the level of operational effort required to meet KPIs.
- **Usability.** Validation that the application meets required levels of usability. List the required training level and user KPIs required.

The testing team, development team, and the business unit agree upon the list of test objectives and their priority. The following image shows an example of a Test Objectives document.

ID	Type	Objective	Name	Author	Reviewed	Priority
1	Functional Correctness	Ability to identify a duplicate pending contract holder in the system	FC1	Jane Smith	Not Reviewed	3-Medium
2	Functional Correctness	Ability to keep historic pending contract holder in the system	FC2	Jane Smith	Not Reviewed	4-High
3	Functional Correctness	Ability to keep historic pending contract holder 'rejection reason' in the system	FC3	John Smith	Not Reviewed	5-Very High
4	Functional Correctness	Ability to track status of pending contract holder	FC4	John Smith	Not Reviewed	1-Low
5	Functional Correctness	Validate support for Manage Quotes Business Process	FC4	Jane Smith	Not Reviewed	3-Medium
6	Service Level	Verify system support for 3050 concurrent sales call center users	SLA1	Jane Smith	Not Reviewed	3-Medium
7	Functional Correctness	Verify proper restrictions to Account functionality and data based on role	FC5	John Smith	Not Reviewed	3-Medium
8	Service Level	Verify view paint response time <2 seconds for commonly used views	SLA2	John Smith	Not Reviewed	4-High
9	Usability	Verfiy a novice user can create a quote with 1 hour of training	U1	Jane Smith	Not Reviewed	4-High

A test case covers one or more test objectives, and has the specific steps that testers follow to verify or validate the stated objectives. The details of the test plan are described in *Test Plans*.

Test Plans

The purpose of the test plan is to define a comprehensive approach to testing. This includes a detailed plan for verifying the stated objectives, identifying any issues, and communicating schedules towards the stated objectives. The test plan has the following components:

- **Project scope.** Outlines the goals and what is included in the testing project.
- **Test cases.** Detail level test scenarios. Each test plan is made up of a list of test cases, their relevant test phases (schedule), and relationship to requirements (traceability matrix).
- **Business process script inventory and risk assessment.** A list of components (business process scripts) that require testing. Also describes related components and identifies high-risk components or scenarios that may require additional test coverage.
- **Test schedule.** A schedule that describes when test cases will be executed.
- **Test environment.** Outlines the recommendations for the various test environments (for example, Functional, System Integration, and Performance). This includes both software and hardware.
- **Test data.** Identifies the data required to support testing.

Business process testing is an important best practice. Business process testing drives the test case definition from the definition of the business process. In business process testing, coverage is measured based on the percentage of validated process steps.

Best Practice

Functional testing based on a required business process definition provides a structured way to design test cases, and a meaningful way to measure test coverage based on business process steps.

Business process testing is described in more detail in the topics that follow.

Test Cases

A test case represents an application behavior that needs to be verified. For each component, application, and business process you can identify one or more test cases that need verification. The following image shows an example of a test case list. Test plans typically contain multiple test cases.

Test Case ID	Test Case Description	Dependencies	TC Ready for Review	Functional C1	Functional C2	SIT C1	Performance	Requirements ID
TC1.0 – New Contact	Create new contacts: Login → Contacts →*New Contacts* → *Verify Data Elements*	N/A	Approved	X				34, 112, 212, 243, 244
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → Logout	TC1.0, TC1.1, Username, Password, Appropriate	Approved	X			X	24, 25, 36, 37, 40, 44, 59, 99, 107, 194, 196, 226, 233, 235, 244, 254, 256
TC 2.3 - Test a	Testing a Campaign: Login -> *Login* -> Program Plans -> *Query Program Plan* ->	TC1.0, TC1.1,	Not Reviewed	X		X		129, 150, 153,154, 159,
TC2.0 – New Contract Standard	Validates the creation of a new contract and the approval process: Login → Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* → *Approval process* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 112, 133, 135, 138, 193
TC3.0 – Contracts	Validates that this process will fail correctly (negative test): Login →New Contract Holder → *New Contract Holder* → Contracts → *New Contract* → Commissions → *Commissions* Approval Process → *Rejection* → *Resubmit* → *Logout*	TC1.x, LOVs and State model	Not Reviewed		X	X		24, 44, 59, 77, 79, 85, 98, 99, 100, 107, 112, 133, 135, 138, 193,
TC4.3 – Contracts	Validate the converted data in the Contracts fields are correct based on test inputs (using field names from the Design document).	TC1.x, TC2.x, Contracts data	Not Reviewed			X		51, 112, 143 198, 200, 201, 204, 265

The example shown in this image uses the following numbering schema for the Test Case ID:

- TC1.x – New records and seed data required to support other test cases
- TC2.x – Positive test cases
- TC3.x – Negative test cases
- TC4.x – Data Conversion testing
- TC5.x – System integration testing

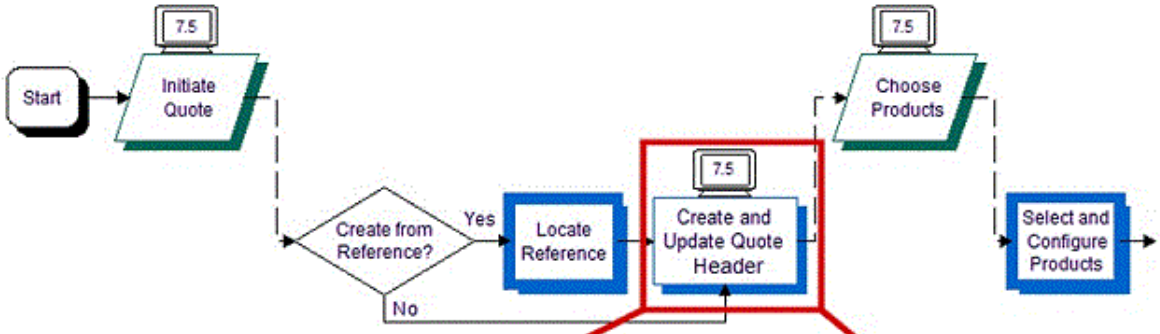
Notice how the test schedule is included in the example: TC1.0 – New Contact is performed during Functional Cycle 1 (Functional C1) of the functional testing. Whereas TC3.0 – Contracts occurs during Functional Cycle 2 (Functional C2) and during system integration testing.

During the *Design Phase* of a test plan, there are a number of test types that you must define as follows:

- **Functional test cases.** Functional test cases are designed to validate that the application performs a specified business function. The majority of these test cases take the form of user or business scenarios that resemble common transactions. Testers and business users should work together to compile a list of scenarios. Following the business process testing practice, functional test cases should be derived directly from the business process, where each step of the business process is clearly represented in the test case.

  For example, if the test plan objective is to validate support for the Manage Quotes Business Process, then there should be test cases specified based on the process definition. Typically, this means that each process or subprocess has one or more defined test cases and each step in the process is specified within the test case definition. The following image illustrates the concept of a process-driven test case. Considerations must also

be given for negative test cases that test behaviors when unexpected actions are taken (for example, creation of a quote with a create date before the current date).



Test Case: 12-ABCDEF
Test Case Name: Create Quote
Test Case Description: This test case creates a new quote header and tests defaulting behavior.
Application: Siebel Sales
Components under test: View Quote List View
Date Created: 5/2/2003
Created By: Joe Tester

Step ID	Process Step	Test Case Step	Expected Result
1	Create a high-tech quote	Login to Siebel Call Center - as sadmin/sadmin	
		Click the OK button to login.	Siebel application starts.
3	Create Quote Header	Click the Quotes tab.	Navigate to Quote environment.
4		Click the New button in Quote List applet	New Quote Line created.
5		Complete key fields (Account or Cantact Name and Bill To / Ship To addresses).	Fields saved appropriately.
6		Validate defaulted fields (Price List and Status).	Defaulting behavior consistent.
7	Select and Configure Products	Click the Add Items button.	"Pick Products" popup applet appears.
8		Locate desired product.	Product Located by scrolling or querying.
9		Add desired number to quantity field.	Data saved.
10		Click Add.	Line item details added to "Pick Products" applet.
11		Click OK.	Popup applet closes. Quote Line Item added.

- **Structural test cases.** Structural test cases are designed to verify that the application structure is correct. They differ from functional cases in that structural test cases are based on the structure of the application, not on a scenario. Typically, each component has an associated structural test case that verifies that the component has the correct layout and definition (for example, verify that a view contains all the specified applets and controls).
- **Performance test cases.** Performance test cases are designed to verify the performance of the system or a transaction. There are three categories of performance test cases commonly used:
  - **Response time or throughput.** Verifies the time for a set of specified actions. For example, tests the time for a view to paint or a process to run. Response time tests are often called *performance* tests.
  - **Scalability.** Verifies the capacity of a specified system or component. For example, test the number of users that the system can support. Scalability tests are often called *load* or *stress* tests.
  - **Reliability.** Verifies the duration for which a system or component can be run without the need for restarting. For example, test the number of days that a particular process can run without failing.

Test Phase

Each test case should have a primary testing phase identified. You can run a given test case several times in multiple testing phases, but typically the first phase in which you run it is considered the primary phase. The following describes how standard testing phases typically apply to Siebel business application deployments:

- **Unit test.** The objective of the unit test is to verify that a unit (also called a component) functions as designed. The definition of a unit is discussed in *Component Inventory*. In this phase of testing, in-depth verification of a single component is functionally and structurally tested.

  For example, during the unit test the developer of a newly configured view verifies that the view structure meets specification and validates that common user scenarios, within the view, are supported.
- **Module test.** The objective of the module test is to validate that related components fit together to meet specified application design criteria. In this phase of testing, functional scenarios are primarily used. For example, testers will test common navigation paths through a set of related views. The objective of this phase of testing is to verify that related Siebel components function correctly as a module.
- **Process test.** The objective of the process test is to validate that business process are supported by the Siebel application. During the process test, the previously-tested modules are strung together to validate an end-to-end business process. Functional test cases, based on the defined business processes are used in this phase.
- **Data conversion test.** The objective of the data conversion test is to validate that the data is properly configured and meets all requirements. This should be performed before the integration test phase.
- **Integration test.** In the integration test phase, the integration of the Siebel business application with other back-end, middleware, or third-party components are tested. This phase includes functional test cases and system test cases specific to integration logic. For example, in this phase the integration of Siebel Orders with an ERP Order Processing system is tested.
- **Acceptance test.** The objective of the acceptance test is to validate that the system is able to meet user requirements. This phase consists primarily of formal and ad-hoc functional tests.
- **Performance test.** The objective of the performance test is to validate that the system will support specified performance KPIs, maintenance, and reliability requirements. This phase consists of performance test cases.

Component Inventory

The Component Inventory is a comprehensive list of the applications, modules, and components in the current project. The component inventory is typically carried out at the project level and is not a testing-specific activity. There are two ways projects identify components. The first is to base component definition on the work that needs to be done (for example, specific configuration activities). The second method is to base the components on the functionality to be supported. In many cases, these two approaches produce similar results. A combination of the two methods is most effective in making sure that the test plan is complete and straightforward to execute.

The worksheet shown in the following image is one example of a component inventory. The worksheet contains the following columns and corresponding information:

- **CID.** For example: C1, C2, C3, and so on.
- **Component.** For example: Product Catalog, Configuration Rules, Quote View, and so on.
- **Type.** For example: Content, Rules, View, and so on.
- **Parent Module.** For example: Catalog, Catalog, Quotes, and so on.

- **Parent Application.** For example: Sales, and so on.
- **Description.** For example: All administered product data, Configuration rules, Quote view, and so on.
- **Risk Score.** For example: 2, 3, 1, and so on.

CID	Component	Type	Parent Module	Parent Application	Description	Risk Score
C1	Product Catalog	Content	Catalog	Sales	All administered product data	2
C2	Configuration Rules	Rules	Catalog	Sales	Configuration rules	3
C3	Quote View	View	Quotes	Sales	Quote View	1
C4	Order View	View	Orders	Sales	Order View	1
C5	Pricing Rules	Rules	Pricer	Sales	Price lists and pricing rules	4
C6	Order to SAP	Integration	SAP Integration	Integration	Integration to SAP for Orders	4

Risk Assessment

A risk assessment is used to identify those components that carry higher risk and may require enhanced levels of testing. The following characteristics increase component risk:

- **High business impact.** The component supports high business-impact business logic (for example, complex financial calculation).
- **Integration.** This component integrates the Siebel application to an external or third-party system.
- **Scripting.** This component includes the coding of browser script, eScript, or VB script.
- **Ambiguous or incomplete design.** This component design is either ambiguous (for example, multiple implementation options described) or the design is not fully specified.
- **Availability of data.** Performance testing requires production-like data (a data set that has the same shape and size as that of the production environment). This task requires planning, and the appropriate resources to stage the testing environment.
- **Downstream dependencies.** This component is required by several downstream components.

Taking these characteristics into consideration, the Risk Score column of the component inventory (shown in the image in *Component Inventory*) shows a risk score for each component. In this example, one risk point is given to a component for each of the criteria met. The scoring system should be defined to correctly represent the relative risk between components. Performing a risk assessment is important for completing a test plan, because the risk assessment provides guidance on the sequence and amount of testing required.

Best Practice

Performing a risk assessment during the planning process allows you to design your test plan in a way that minimizes overall project risk.

Test Plan Schedule

For each test plan, a schedule of test case execution should be specified. The schedule is built using four different inputs:

- **Overall project schedule.** The execution of all test plans must be consistent with the overall project schedule.
- **Component development schedule.** The completion of component configuration is a key input to the testing schedule.
- **Environment availability.** The availability of the required test environment needs to be considered in constructing schedules.

- **Test case risk.** The risk associated with components under test is another important consideration in the overall schedule. Components with higher risk should be tested as early as possible.

Test Environments

The specified test objectives influence the test environment requirements. For example, service level test objectives (such as system availability, load, and responsiveness) often require an isolated environment to verify. In addition, controlled test environments can help:

- **Provide integrity of the application under test.** During a project, at any given time there are multiple versions of a module or system configuration. Maintaining controlled environments can make sure that tests are being executed on the appropriate versions. Significant time can be wasted executing tests on incorrect versions of a module or debugging environment configuration without these controls.
- **Control and manage risk as a project nears rollout.** There is always a risk associated with introducing configuration changes during the lifecycle of the project. For example, changing the configuration just before the rollout carries a significant amount of risk. Using controlled environments allows a team to isolate late-stage and risky changes.

It is typical to have established Development, Functional Testing, System Testing, User Acceptance Testing, Performance Testing, and Production environments to support testing. More complex projects often include more environments, or parallel environments to support parallel development. Many customers use standard code control systems to facilitate the management of code across environments.

The environment management approach includes the following components:

- **Named environments and migration process.** A set of named test environments, a specific purpose (for example, integration test environment), and a clear set of environment entry and exit criteria. Typically, the movement of components from one environment to the next requires that each component pass a predefined set of test cases, and is done with the appropriate level of controls (for example, code control and approvals).
- **Environment audit.** A checklist of system components and configuration for each environment. Audits are performed prior to any significant test activity. The Environment Verification Tool can be used to facilitate the audit of test environments. For help with the Environment Verification Tool, see 477105.1 (Doc ID) on My Oracle Support. This document was previously published as Siebel Technical Note 467.
- **Environment schedule.** A schedule that outlines the dates when test cases will be executed in a given environment.

Performance Test Environment

In general, the more closely the performance test environment reflects the production environment, the more applicable the test results will be. It is important that the performance test environment includes all of the relevant components to test all aspects of the system, including integration and third-party components. Often it is not feasible to build a full duplicate of the production configuration for testing purposes. In that case, the following scaled-down strategy should be employed for each tier:

- **Web Servers and Siebel Servers.** To scale down the Web and application server tiers, the individual servers should be maintained in the production configuration and the number of servers scaled down proportionately. The overall performance of a server depends on a number of factors besides the number of CPUs, CPU speed,

and memory size. So, it is generally not accurate to try to map the capacity of one server to another even within a single vendor's product line.

The primary tier of interest from an application scalability perspective is the application server tier. Scalability issues are very rarely found on the Web server tier. If further scale-down is required it is reasonable to maintain a single Web server and continue to scale the application server tier down to a single server. The application server should still be of the same configuration as those used in the production environment, so that the tuning activity can be accurately reflected in the system test and production environments

- **Database server.** If you want to scale down a database server, there is generally little alternative but to use a system as close as possible to the production architecture, but with CPU, memory, and I/O resources scaled down as appropriate.
- **Network.** The network configuration is one area in which it is particularly difficult to replicate the same topology and performance characteristics that exist in the production environment. It is important that the system test includes any active network devices such as proxy servers and firewalls. The nature of these devices can impact not only the performance of the system, but also the functionality, because in some cases these devices manipulate the content that passes through them. The performance of the network can often be simulated using bandwidth and latency simulation tools, which are generally available from third-party vendors.

4 Design and Develop Tests

Design and Develop Tests

This chapter describes the process of developing the tests that you should perform during the development of your project. It includes the following topics:

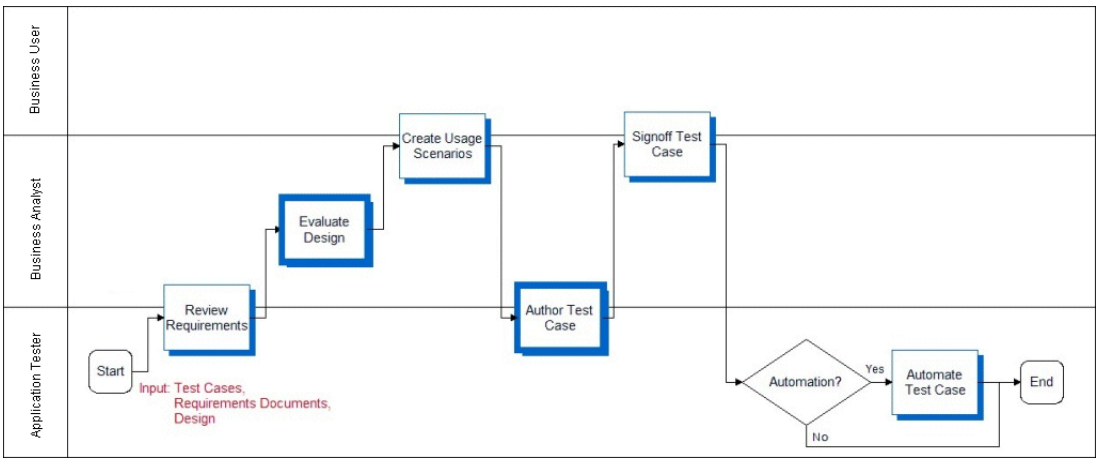
- *Overview of Test Development*
- *Design Evaluation*
- *Test Case Authoring*
- *Test Case Automation*

Overview of Test Development

It is important that you develop test cases in close cooperation between the tester, the business analyst, and the business user. To generate valid and complete test cases, they must be written with full understanding of the requirements, specifications, and usage scenarios.

The process in the following image illustrates some of the activities that should take place in the test development process, including the following:

1. Review requirements.
2. Evaluate design.
3. Create usage scenarios.
4. Author test cases.
5. Signoff test case.
6. (Optional) Automate test case.



As shown in this image, the deliverables for the test development process include:

- **Requirement gaps.** As a part of the design review process, the business analyst should identify business requirements that have incomplete or missing designs. This can be a simple list of gaps tracked in a

spreadsheet. Gaps must be prioritized and critical issues scoped and reflected in the updated design. Lower priority gaps enter the change management process.

- **Approved technical design.** This is an important document that the development team produces (not a testing-specific activity) that outlines the approach to solving a business problem. It should provide detailed process-flow diagrams, UI mock-ups, pseudo-code, and integration dependencies. The technical design should be reviewed by both business analysts and the testing team, and approved by business analysts.
- **Detailed test cases.** Step-by-step instructions for how testers execute a test.
- **Test automation scripts.** If test automation is a part of the testing strategy, the test cases need to be recorded as actions in the automation tool. The testing team develops the functional test automation scripts, while the IT team typically develops the performance test scripts.

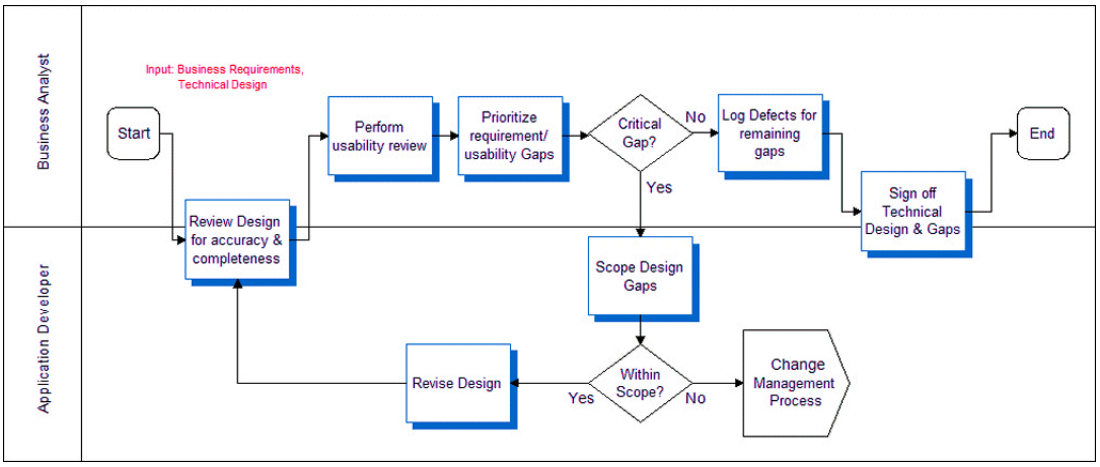
Design Evaluation

The earliest form of testing is design evaluation. Testing during this stage of the implementation is often neglected. Development work should not start until requirements are well understood, and the design can fully address the requirements. All stakeholders should be involved in reviewing the design. Both the business analyst and business user, who defined the requirements, should approve the design.

An example design evaluation process is illustrated in the following image and includes the following steps:

1. Review Design for accuracy and completeness.
2. Perform usability review.
3. Prioritize requirement/usability gaps.
4. If there are critical gaps, then scope the design gap and either revise the design (if time) or set up a change management process (to handle later).
5. If there are non-critical gaps, then log defects for same.

For more information, see *Reviewing Design and Usability*.



Reviewing Design and Usability

Two tools for identifying issues or defects are the Design Review and Usability Review. These early stage reviews serve two purposes. First, they provide a way for development to describe the components to the requirement solution. Second, they allow the team to identify missing or incomplete requirements early in the project. Many critical issues are often introduced by incomplete or incorrect design. These reviews can be as formal or informal as deemed appropriate.

Many customers have used design documents, whiteboard sessions, and paper-based user interface mock-ups for these reviews.

Once the design is available, the business analyst should review it to make sure that the business objectives can be achieved with the system design. This review identifies functional gaps or inaccuracies. Usability reviews determine design effectiveness with the UI mock-ups, and help identify design inadequacies.

Task-based usability tests are the most effective. In this type of usability testing, the tester gives a user a task to complete (for example, create an activity), and using the user interface prototype or mock-up, the user describes the steps that he or she would perform to complete the task. Let the user continue without any prompting, and then measure the task completion rate. This UI testing approach allows you to quantify the usability of specific UI designs.

The development team is responsible for completing the designs for all business requirements. Having a rigorous design and review process can help avoid costly oversights.

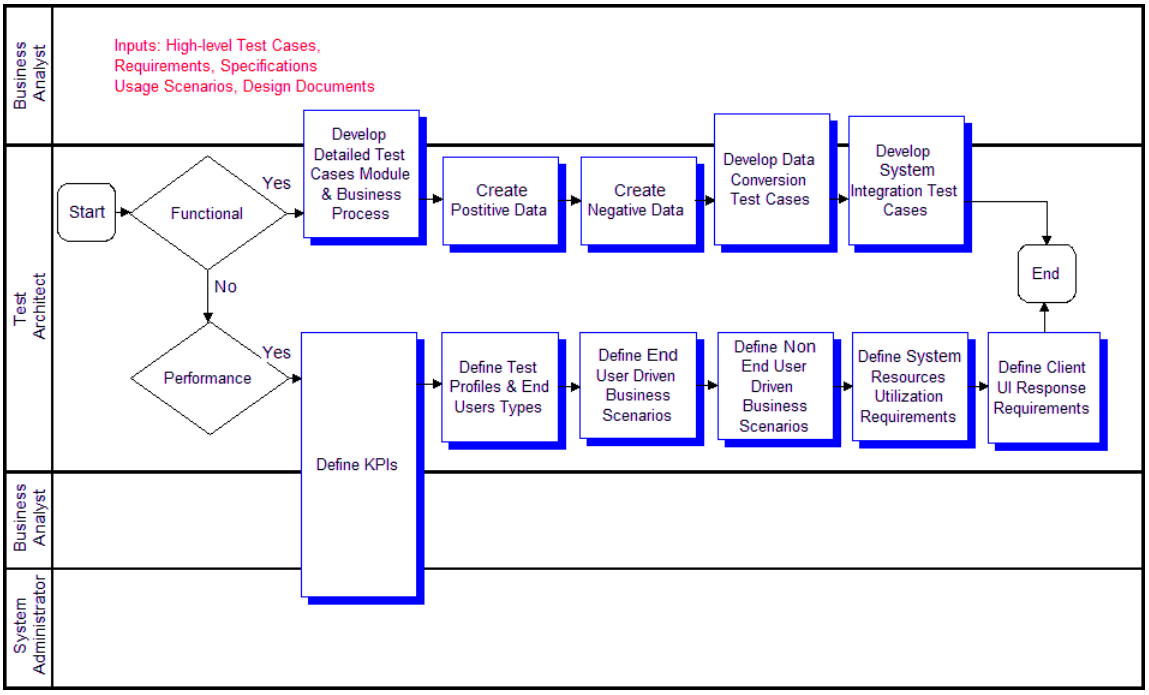
Test Case Authoring

Based on the test case objective, requirements, design, and usage scenarios, the process of authoring test cases can begin. Typically this activity is performed with close cooperation between the testing team and business analysts.

The following image illustrates the process for authoring test cases and includes the following stages:

1. **Author functional test cases.** *Functional Test Cases* test a common business operation or scenario to make sure that it does what it is supposed to do.
2. **Author system test cases.** *System Test Cases* are typically used in the system integration test phase to make sure that a component or module is built to specification, making sure that the structure of the application is correct.

   Functional tests focus on validating support for a scenario whereas system tests make sure that the structure of the application is correct.
3. **Author performance test cases.** *Performance Test Cases* typically simulate system activity using automated testing tools.



Functional Test Cases

Functional test cases test a common business operation or scenario. The following table describes the different types of test phases in Functional Test Cases.

A functional test case may verify common control navigation paths through a view. Functional test cases typically have two components: test paths and test data.

Test Phase	Example
Unit test	Test common control-level navigation through a view. Test any field validation or default logic. Invoke methods on an applet.
Module test	Test common module-level user scenarios (for example, create an account and add an activity). Verify correct interaction between two related Siebel components (for example, Workflow Process and Business Service).
Process test	Test proper support for a business process.
User interface	Verify that a view has all specified applets, and each applet has specified controls with correct type and layout.
Data entity	Verify that a data object or control has the specified data fields with correct data types.

Test Case

A test case describes the actions and objects that you want to test. A case is presented as a list of steps with the expected behavior at the completion of a step. The following image shows an example of a test case.

TC1.0 - Create Contact							
Project Name	Project ABC					Date	3/3/2004
Test Case Description	The purpose of this Test Case is to test the creation of a new contact. Login --> Contacts --> *New Contact* --> Logout					Requirements	3, 54, 123, 45
Function / Module Under Test	Contact					Test Type	Manual
Written by							
Goals	To create a contact in CMS as a basis for building relationship, opportunity, and account information in the system						
Test Setup	N/A						
Dependencies	Appropriate username, password and role						
ID	Process	Detailed Step	Expected Results	User	Pass/Fail (Criteria)	Data Input	Siebel Reference
1	Login	Type User ID into the User Name field	Field is populated			User Name	Login
2		Type Password into the Password field	Field is populated			Password	
3		Click the Login Button	Siebel Launches				
4	Navigate_Contact	Click on Contacts tab	Contacts - Rolodex View opens	What type of user or username goes here	View opens without error.		Contacts
5	Search_Contact_By_Name	Click on Search view tab	Contacts-Search view opens				
6		Select "Name" search method in Search By field	Field is populated, appropriate form controls display on applet				
7		Type [First Name] in First field	Field is populated			First Name	
8		Type [Last Name] in Last field	Field is populated			Last Name	
9	Create_Contact	Click the New button	New record displays in Contacts list applet				

In the Detailed Step column, there are no data values in the step. Instead you see a parameter name in brackets as a place holder. This parameterization approach is a common technique used with automation tools, and is helpful for creating reusable test cases.

Test Data

Frequently, you can use a single path to test many scenarios by simply changing the data that is used. For example, you can test the processing of both high-value and low-value opportunities by changing the opportunity data entered, or you can test the same path on two different language versions of the application. For this reason, it can be helpful to define the test path separately from the test data.

System Test Cases

System test cases are typically used in the system integration test phase to make sure that a component or module is built to specification. Functional tests focus on validating support for a scenario whereas system tests make sure that the structure of the application is correct. The following table shows some examples of typical system tests.

Object Type	Example
Interface	Verify that an interface data structure has the correct data elements and correct data types.
Business Rule	Verify that a business rule (for example, assignment rule) handles all inputs and outputs correctly.

Object Type	Example

Performance Test Cases

You accomplish performance testing by simulating system activity using automated testing tools. Oracle has several software partners who provide load testing tools that have been validated to integrate with Siebel business applications. Automated load-testing tools are important because they allow you to accurately control the load level, and correlate observed behavior with system tuning. This topic describes the process of authoring test cases using an automation framework.

When you are authoring a performance test case, first document the key performance indicators (KPIs) that you want to measure. The KPIs can drive the structure of the performance test and also provide direction for tuning activities. Typical KPIs include resource utilization (CPU, memory) of any server component, uptime, response time, and transaction throughput.

The performance test case describes the types of users and number of users of each type that will be simulated in a performance test. The following image shows a typical test profile for a performance test and includes the following information: Test Case #, Test Case Name, Test Case Description, Application, KPIs, User Type, Number Users, and Total Number of Users and Business Transactions.

Test Case: T13
Test Case Name: 3050 User Callcenter Load
Test Case Description: Verifies peak callcenter load of 3050 users
Application: Siebel Call Center
KPIs: CPU, Memory, Transaction response times
Date Created: 5/28/2003
Created By: Joe Tester

User Type	Num Users
Incoming Call Creates Opportunity, Quote and Order	957
Campaign Call Creates Opportunity	652
Call Creates a Service Request	534
Agent Follows Up On Service Request	907
Total Number of Users and Business Transactions	**3,050**

Test cases should be created to mirror various states of your system usage, including:

- **Response time or throughput.** Simulate the expected typical usage level of the system to measure system performance at a typical load. This allows evaluation against response time and throughput KPIs.
- **Scalability.** Simulate loads at peak times (for example, end of quarter or early morning) to verify system scalability. Scalability (stress test) scenarios allow evaluation of system sizing and scalability KPIs.
- **Reliability.** Determine the duration for which the application can be run without the need to restart or recycle components. Run the system at the expected load level for a long period of time and monitor system failures.

User Scenarios

The user scenario defines the type of user, as well as the actions that the user performs. The first step in authoring performance test cases is to identify the user types that are involved. A user type is a category of typical business user. You arrive at a list of user types by categorizing all users based on the transactions they perform. For example, you may have call center users who respond to service requests and call center users who make outbound sales calls. For each

user type, define a typical scenario. It is important that scenarios accurately reflect the typical set of actions taken by a typical user because scenarios that are too simple or too complex skew the test results. There is a trade-off that must be balanced between the effort to create and maintain a complex scenario and accurately simulating a typical user. Complex scenarios require more time-consuming scripting while scenarios that are too simple may result in excessive database contention – because all simulated users will simultaneously try to access the small number of tables that support a few operations.

Most user types fall into one of two usage patterns:

- **Multiple-iteration users** tend to log in once, and then cycle through a business process multiple times (for example, call center representatives). The Siebel application has a number of optimizations that take advantage of persistent application state during a user session, and it is important to accurately simulate this behavior to obtain representative scalability results. The scenario should show the user logging in, iterating over a set of transactions, and then logging out.
- **Single-iteration scenarios** emulate the behavior of occasional users such as e-commerce buyers, partners at a partner portal, or employees accessing ERM functions such as employee locator. These users typically execute an operation once and then leave the Siebel environment, and so do not take advantage of the persistent state optimizations for multiple-iteration users. The scenario should show the user logging in, performing a single transaction, and then logging out.

The following image shows the user wait times for the following scenario:

- **User Type:** Incoming Call Creates Opportunity and Quote.
- **Iteration:** Multiple Iteration.

For example, the Go_New_Call operation has a wait time of 5 seconds, the New_Contact operation has a wait time of 60 seconds, and the New_Oppty operation has a wait time 45 seconds. It is important that wait times be distributed throughout the scenario, and reflect the time that an actual user takes to perform the operation.

User Type: Incoming Call Creates Opportunity and Quote
Iteration: Mutiple Iteration

Operation Name	Operation	Think Time (sec)	System Response KPI (sec)	
Go_New_Call	Click on "**Retrieve Call**" icon on CTI bar	5	1	
Find_Corp_Cont	Click Find (Binocular) Button	2	1	
	Query for non-existing "**Corporate Contact**", e.g. 'Kim'	10	1.5	
New_Contact	Enter new contact	60	1	
Go_Cont_Opty	Navigate to Contact - Opportunities View	5	1	
New_Opty	Enter new opportunity	45	1	
Go_Opty_Cont	Drilldown on opportunity name to Opportunity - Contacts View	5	2	
Go_Opty_Prod	Navigate to Opportunity Products	2	1	
New_Product (2)	Enter two new products	45	1	
Go_Opty_Quote	Navigate to Opportunities - Quotes View	3	1	
Click_Auto Quote	Click "Auto Quote" button to generate quote	5	3	
Enter_Quote_Info	Enter Quote Name, Price List and Discount	30	2	
Go_Quote_Line	Drilldown on the quote name to go to Quote - Line Items View	5	2	
Quote_Reprice	Click "Reprice All" button	2	2	
	Communicate the results of "Reprice All" to prospect (no navigation required)	30	0	
Quote_Upd_Opty	Click "Update Opty" button	1	2	
Go_Quote_Order	Navigate to Quotes - Orders View	2	2	
Click_Auto Order	Click on "Auto Order" button to automatically generate order	2	3	
	Wrap up call (no navigation required)	10	0	
Go_Thread_Opty	Navigate back to Opty	3	1	
	Wrap up call (no navigation required)	10	0	
Go_Release_Call	Click on "**Release Call**" icon on CTI bar	2	1	
	Total Business Transaction Values	**284**	**29.5**	**313.5**

Data Sets

The data in the database and used in the performance scenarios can impact test results – because this data impacts the performance of the database. It is important to define the data shape to be similar to what is expected in the production system. Many customers find it easiest to use a snapshot of true production data sets to do this.

Test Case Automation

Oracle partners with the leading test automation tool vendors, who provide validated integrations with Siebel business applications. Automation tools can be a very effective way to execute tests. In the case of performance testing, automation tools are critical to provide controlled, accurate test execution. When you have defined test cases, you can automate them using third-party tools.

Functional Automation

Using automation tools for functional or system testing can cost less than performing manual test execution. You should consider which tests to automate because there is a cost in creating and maintaining functional test scripts. Acceptance regression tests benefit the most from functional test automation technology.

For functional testing, automation provides the greatest benefit when testing relatively stable functionality. Typically, automating a test case takes approximately five to seven times as long as manually executing it once. Therefore, if a test case is not expected to be run more than seven times, the cost of automating it may not be justified.

Performance Automation

Automation is necessary to conduct a successful performance test. Performance testing tools virtualize real users, allowing you to simulate thousands of users. In addition, these virtual users are less expensive, more precise, and more tolerant than actual users. The process of performance testing and tuning is iterative, so it is expected that a test case will be run multiple times to first identify performance issues, and then verify that any tuning changes have corrected observed performance issues.

Performance testing tools virtualize real users by simulating the HTTP requests made by the client for the given scenario. The Siebel Smart Web Client Architecture separates the client-to-server communication into two channels, one for layout and one for data. The protocol for the data channel communication is highly specialized; therefore Oracle has worked closely with leading test tool vendors to provide their support for Siebel business applications. Because the communication protocol is highly specialized and subject to change, it is strongly recommended that you use a validated tool.

At a high level, the process of developing automated test scripts for performance testing has four steps - refer to the instructions provided by your selected tool vendor for more detailed information:

- **Record scripts for each of the defined user types.** Use the automation tool's recording capability to record the scenario documented in the test case for each user. Keep in mind the multiiteration versus single iteration distinction between user types. Many tools automatically record user wait times. Modify these values, if necessary, to make sure that the recorded values accurately reflect what was defined in the user type scenario.
- **Insert parameterization.** Typically, the recorded script must be modified for parameterization. Parameterization allows you to pass in data values for each running instance of the script. Because each virtual user runs in parallel, this is important for segmenting data and avoiding uniqueness constraint violations.
- **Insert dynamic variables.** Dynamic variables are generated based on data returned in a prior response. Dynamic variables allow your script to intelligently build requests that accurately reflect the server state. For example, if you execute a query, your next request should be based on a record returned in the query result set. Examples of dynamic variables in Siebel business applications include session ids, row ids, and timestamps. All validated load test tool vendors provide details on how dynamic variables can be used in their product.

- **Script verification.** After you have recorded and enhanced your scripts, run each script with a single user to verify that it functions as expected.

Oracle offers testing services that can help you design, build, and execute performance tests if you need assistance.

Best Practice

Using test automation tools can reduce the effort required to execute tests, and allows a project team to achieve greater test coverage. Test Automation is critical for Performance testing, because it provides an accurate way to simulate large numbers of users.

5 Execute Siebel Functional Tests

Execute Siebel Functional Tests

This chapter describes the process of executing Siebel functional tests. It includes the following topics:

- *Overview of Executing Siebel Functional Tests*
- *Track Defects Subprocess*

Overview of Executing Siebel Functional Tests

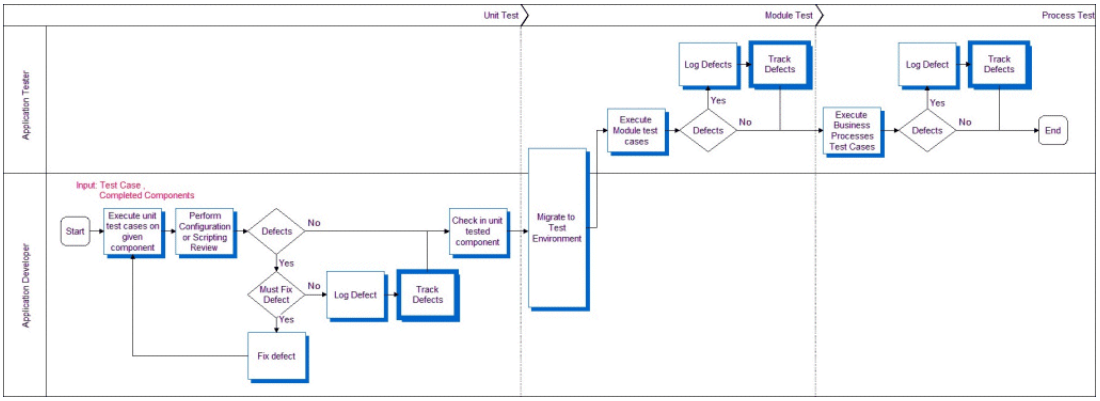
The process of executing Siebel functional tests is designed to provide for delivery of a functionally validated Siebel application into the system environment. For many customers the Siebel application is one component of the overall system which may include other back-end applications as well as integration infrastructure and network infrastructure. Therefore, the objective of executing Siebel functional tests is to verify that the Siebel application functions properly before inserting it into the larger system environment.

Application developers test their individual components for functional correctness and completeness before checking component code into the repository. The unit test cases should have been designed to test the low-level details of the component (for example, control behavior, layout, data handling).

Typical unit tests include structural tests of components, negative tests, boundary tests, and component-level scenarios. The unit test phase allows developers to fast track fixes for obvious defects before checking in. A developer must demonstrate successful completion of all unit test cases before checking in their component. In some cases, unit testing identifies a defect that is not critical for the given component; these defects are logged into the defect tracking system for prioritization.

Once unit testing has been completed on a component, that component is moved into a controlled test environment, where the component can be tested along side others at the module and process level.

The following image illustrates the process of executing Siebel functional tests.



As shown in this image, there are three phases in the process of executing Siebel functional tests:

1. **Unit test.** The unit test validates the functionality of a single component (for example, an applet or a business service). Typical steps involved in unit testing include the following:
    a. Execute unit test cases on a given (completed) component.
    b. Perform configuration or scripting review.
    c. Regarding defects:
        - Fix critical defects that must be fixed, and return to step a.
        - Log and track other defects, if there are any, and continue to step d.
    d. Check in unit tested component.
2. **Module test.** The module test validates the functionality of a set of related components that make up a module (for example, Contacts or Activities). Typical steps involved in module testing include the following:
    a. Migrate the (unit tested) component to a Test environment.
    b. Execute module test cases on the component.
    c. Log and track defects if there are any.
3. **Process test.** The process test validates that multiple modules can work together to enable a business process (for example, Opportunity Management or Quote to Order). Typical steps involved in module testing include the following:
    a. Execute business process test cases on the component.
    b. Log and track defects if there are any.

Reviews

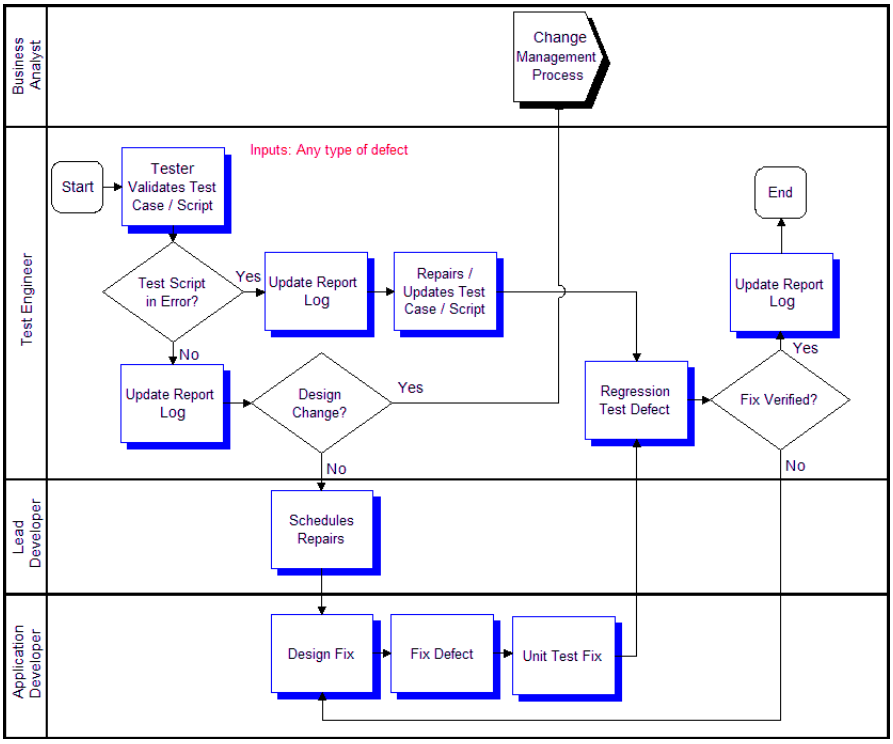
There are two types of reviews done in conjunction with functional testing: configuration review and scripting code review.

- **Configuration review.** This is a review of the Siebel application configuration using Siebel Tools. Configuration best practices should be followed. Some common recommendations include using optimized, built-in functionalities rather than developing custom scripts and using primary joins to improve MVG performance.
- **Scripting code review.** Custom scripting is the source of many potential defects. These defects are the result of poor design or inefficient code that can lead to severe performance problems. A code review can identify design flaws and recommend code optimization to improve performance.

Checking in a component allows the testing team to exercise that component along side related components in an integration test environment. Once in this environment, the testing team executes the integration test cases based on the available list of components. Integration tests are typically modeled as actual usage scenarios, which allow testers to validate that a user can perform common tasks. In contrast to unit test cases, these tests are not concerned with specific details of any one component, but rather the way that logic is handled when working across multiple components.

Track Defects Subprocess

The Track Defects subprocess is designed to collect the data required to measure and monitor the quality of the application, and also to control project risk and scope. The process, illustrated in the following image, is designed so that those with the best understanding of the customer priorities are in control of defect prioritization.



As shown in this image:

- The business analyst monitors a list of newly discovered issues using a defect tracking system like the Siebel Quality module. These users monitor, prioritize, and target defects with regular frequency. This is typically done daily in the early stages of a project, and perhaps several times a day in later stages.
- The level of scrutiny is escalated for defects discovered after the project freeze date. A very careful measurement of the impact to the business of a defect versus the risk associated with introducing a late change must be made at the project level. Commonly, projects that do not have appropriate levels of change management in place have difficulty reaching a level of system stability adequate for deployment. Each change introduced carries with it some amount of regression risk. Late in a project, it is the responsibility of the entire project team, including the business unit, to carefully manage the amount of change introduced.
- Once a defect has been approved to be fixed, it is assigned to development and a fix is designed, implemented, unit tested, and checked in. The testing team must then verify the fix by bringing the affected components back to the same testing phase where the defect was discovered. This requires regression testing (re-execution of test cases from earlier phases). The defect is finally closed and verified when the component or module successfully passes the test cases in which it was discovered. The process of validating a fix can often require the re-execution of past test cases, so this is one activity where automated testing tools can provide significant savings. One best practice is to define regression suites of test cases that allow the team to re-execute a relevant, comprehensive set of test cases when a fix is checked in.

- Tracking defects also collects the data required to measure and monitor system quality. Important data inputs to the deployment readiness decision include the number of open defects and defect discovery rate. Also, it is important for the business customer to understand and approve the known open defects prior to system deployment.

Best Practice

The use of a defect tracking system allows a project team to understand the current quality of the application, prioritize defect fixes based on business impact, schedule resources, and carefully control risk associated with configuration changes late in the project.

6 Execute System Integration and Acceptance Tests

Execute System Integration and Acceptance Tests

This chapter describes the process of executing integration and acceptance tests. It includes the following topics:

Overview of Executing Integration and Acceptance Tests

The processes of executing integration and acceptance tests are designed to verify that the Siebel application can properly communicate with other applications or components in the system, support end-to-end business processes, and will be accepted by the user community. This is a very busy and exciting phase of any project, because it marks a point where the system is nearing deployment.

The three major pieces involved in executing integration and acceptance tests processes are as follows:

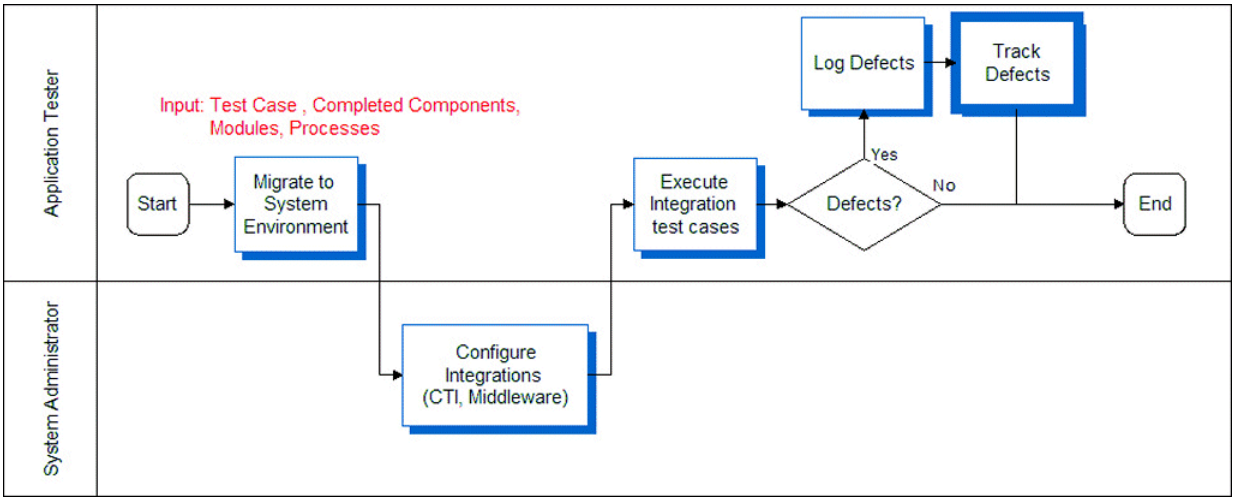
- **Testing integrations with the Siebel application.** In most customer deployments, the Siebel application integrates with several other applications or components. Integration testing focuses on these touch points with third-party applications, network infrastructure, and integration middleware.
- **Functional testing of business processes.** Required business processes must be tested end-to-end to verify that transactions are handled appropriately across component, application, and integration logic. It is important to push a representative set of transaction data through the system and follow all branches of required business processes.
- **Testing system acceptance with users.** User acceptance testing allows system users to use the system to perform simulated work. This phase of testing makes sure that users will be able to use the system effectively once it is live.

Execute Integration Tests

Completion of the Siebel Functional Testing process verifies that the Siebel application functions correctly as a unit. In Integration Testing you verify that this unit functions correctly when inserted into the complete, larger system. In this process, your test cases should be defined to test the integration points between the Siebel application and other applications or components. Typical components include back office applications, integration middleware, network infrastructure components, and security infrastructure. Tests in this process should focus on exercising integration logic, and validating end-to-end business processes that span multiple systems.

The following image illustrates the process involved in executing integration tests and includes the following typical steps:

1. Migrate to system environment.
2. Configure integrations (CTI, Middleware).
3. Execute integration test cases.
4. Log and track defects (if there are any).

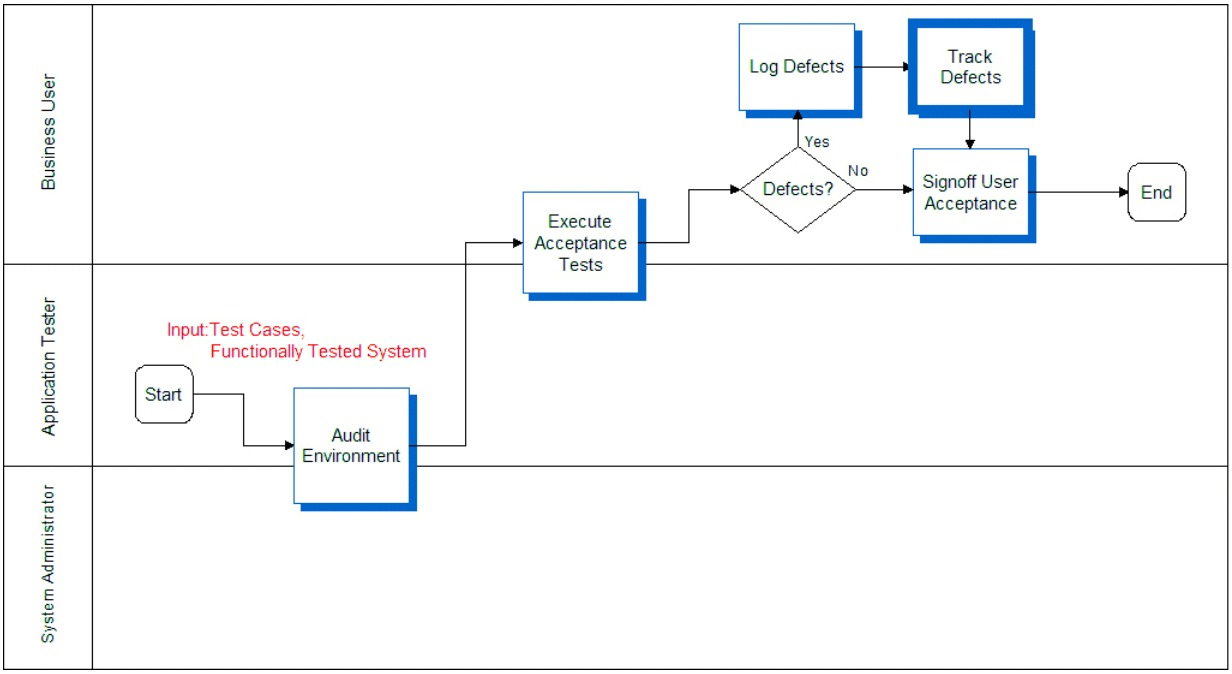


Execute Acceptance Tests

Once the system as a whole has been validated, you must make sure that the functionality provided is acceptable to the business users. Hopefully, the business user has been engaged all along, approving at each phase of the project to make sure that there are no surprises. In the User Acceptance testing process, open the system up to a larger community of trained users and ask them to simulate running their business on the system. User Acceptance testing should be designed to simulate live business as closely as possible. Complete this process by having the user community representative (business user) approve the acceptance test results.

The following image illustrates the process involved in executing acceptance tests and includes the following typical steps:

1. Set up the audit environment.
2. Execute acceptance cases.
3. Log and track defects (if there are any).
4. Signoff user acceptance tests.


Business User
Application Tester
System Administrator
Log Defects
Track Defects
Yes
No
Defects?
Signoff User Acceptance
End
Execute Acceptance Tests
Input:Test Cases,
Functionally Tested System
Start
Audit Environment

7 Execute Performance Tests

Execute Performance Tests

This chapter describes the process of executing performance tests. It includes the following topics:

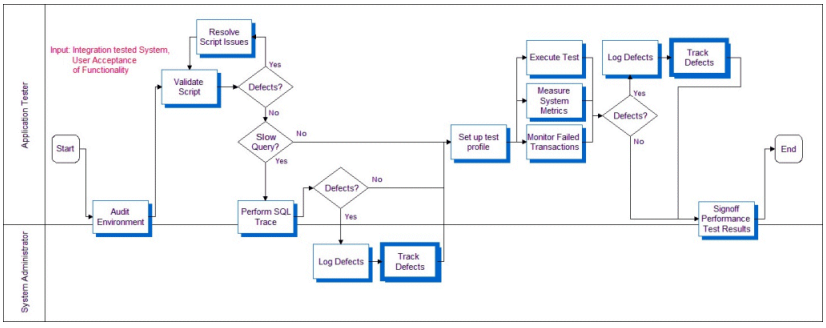
- *Overview of Executing Performance Tests*
- *Executing Tests*
- *Performing an SQL Trace*
- *Measuring System Metrics*
- *Monitoring Failed Transactions*

Overview of Executing Performance Tests

As described earlier, there are three types of performance test cases that are typically executed: response time, stress, and reliability testing. It is important to differentiate between the three because they are intended to measure different KPIs (key performance indicators). Specialized members of the testing and system administration organizations, who have ownership of the system architecture and infrastructure, typically manage performance tests.

The process of executing performance tests involves validating recorded user-type scripts in the system test environment. The following image illustrates the typical steps involved in executing performance tests:

1. Set up the audit environment.
2. Validate scripts.
3. Regarding defects:
    - Resolve script issues if there are defects, and return to step 2.
    - Otherwise perform SQL trace, log and track defects as necessary, and continue to step 4.
4. Set up the test profile, then:
    a. Execute the test.
    b. Measure system metrics.
    c. Monitor failed transactions.
5. Log and track defects as necessary.
6. Signoff performance test acceptance.



Executing Tests

Execute each script for a single user to validate the health of the environment. A low user-load baseline should be obtained before attempting the target user load. This baseline allows you to measure system scalability by comparing results between the baseline and target loads.

Users must be started at a controlled rate to prevent excessive resource utilization due to large numbers of simultaneous logins. This rate depends on the total configured capacity of the system. For every 1000 users of configured system capacity, you add one user every three seconds. For example, if the system is configured for 5000 users, you add five users every three seconds.

Excessive login rate causes the application server tier to consume 100% CPU, and logins begin to fail. Wait times should be randomized during load testing to prevent inaccuracies due to simulated users executing transactions simultaneously. Randomization ranges should be set based on determining the relative wait times of expert and new users when compared to the average wait times in the script.

Performing an SQL Trace

Because poorly formed SQL or suboptimal database-tuning causes many performance issues, the first step to improve performance is to perform an SQL trace. An SQL trace creates a log file that records the statements generated in the Siebel object manager and executed on the database. The time required to execute and fetch on an SQL statement has a significant impact on both the response time seen by end users of a system, and on the system's resource utilization on the database tier. It is important to discover slow SQL statements and root cause, and fix issues before attempting scalability or load tests, as excessive resource utilization on the database server will invalidate the results of the test or cause it to fail.

To obtain an SQL trace

1. Set a breakpoint in the script at the end of each action and execute the script for two iterations.
2. Enable EvtLogLvl (ObjMgrSqlLog=5) to obtain SQL traces for the component on the application server that has this user session or task running.
3. Continue executing the script for the third iteration and wait for the breakpoint at the end of action.
4. Turn off SQL tracing on the application server (reset it to its original value, or 1).
5. Complete the script execution.

The log file resulting from this procedure has current SQL traces for this business scenario. Typically, any SQL statement longer than 0.1 seconds is considered suspect and must be investigated, either by optimizing the execution of the query (typically by creating an index on the database) or by altering the application to change the query.

Measuring System Metrics

Results collection should occur during a measurement period while the system is at a steady state, simulating ongoing operation in the course of a business day. Steady state is achieved once all users are logged in and caches (including

simulated client-side caches) are primed. The measurement interval should commence after the last user has logged in and completed the first iteration of the business scenario.

The measurement interval should last at least one hour, during which system statistics should be collected across all server tiers. We recommend that you measure the following statistics:

- CPU
- Memory
- System calls
- Context switches
- Paging rates
- I/O waits (on the database server)
- Transaction response times as reported by the load testing tool

Note: Response times will be shorter than true end-user response times due to additional processing on the client, which is not included in the measured time.

The analysis of the statistics starts by identifying transactions with unacceptable response times, and then correlating them to observed numbers for CPU, memory, I/O, and so on. This analysis provides insight into the performance bottleneck.

Monitoring Failed Transactions

Less than 1% of transactions should fail during the measurement interval. A failure rate greater than 1% indicates a problem with the scripts or the environment.

Typically, transactions fail for one of the following three reasons:

- **Timeout.** A transaction may fail after waiting for a response for a specified timeout interval. A resource issue at a server tier, or a long-running query or script in the application can cause a timeout.

  If a long-running query or script is applicable to all users of a business scenario, it should be caught in the SQL tracing step. If SQL tracing has been performed, and the problem is only seen during loaded testing, it is often caused by data specific to a particular user or item in the test database. For example, a calendar view might be slow for a particular user because prior load testing might have created thousands of activities for that user on a specific day. This would only show as a slow query and a failed transaction during load testing when that user picks that day as part of their usage scenario

  Long-running transactions under load can also be caused by consumption of all available resources on some server tier. In this case, transaction response times typically stay reasonable until utilization of the critical resource closely approaches 100%. As utilization approaches 100%, response times begin to increase sharply and transactions start to fail. Most often, this consumption of resources is due to the CPU or memory on the Web server, application server, or database server, I/O bandwidth on the database server, or network bandwidth. Resource utilization across the server tiers should be closely monitored during testing, primarily for data gathering purposes, but also for diagnosing the resource consumption problem.

  Very often, a long-running query or script can cause consumption of all available resources at the database server or application server tier, which then causes response times to increase and transactions to time out.

While a timeout problem may initially appear to be resource starvation, it is possible that the root cause of the starvation is a long-running query or script.

- **Data issues.** In the same way that an issue specific to a particular data item may cause a timeout due to a long-running query or script, a data issue may also cause a transaction to fail. For example, a script that randomly picks a quote associated with an opportunity will fail for opportunities that do not have any associated quotes. You must fix data if error rates are significant, but a small number of failures do not generally affect results significantly.
- **Script issues.** Defects in scripts can cause transaction failures. Common pitfalls in script recording include the following:
  - Inability to parse Web server responses due to special characters (quotes, control characters, and so on) embedded in data fields for specific records.
  - Required fields not being parameterized or handled dynamically
  - Strings in data fields that are interpreted by script error-checking code as indicating a failed transaction. For example, it is common for a technical support database to contain problem descriptions that include the string. The server is down or experiencing problems

8 Improve and Continue the Testing Process

Improve and Continue the Testing Process

This chapter describes the steps you can take to make iterative improvements to the testing process. It includes the following topic.

- *Improve and Continue Testing*

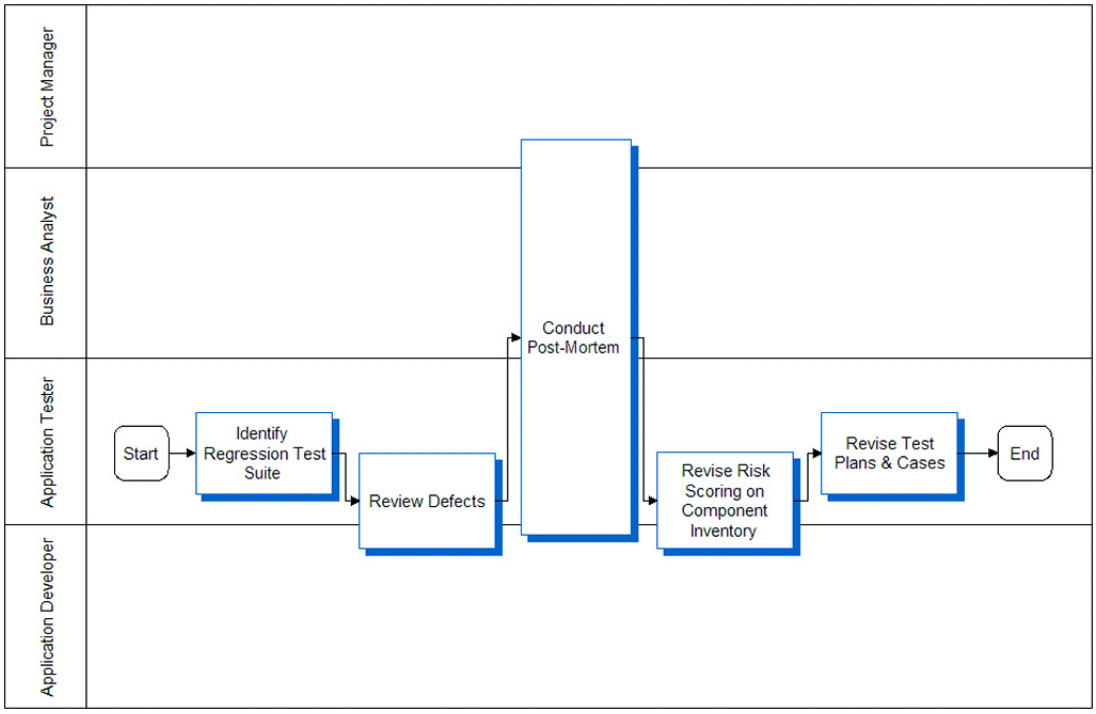
Improve and Continue Testing

After the initial deployment, regular configuration changes are delivered in new releases. In addition, Oracle delivers regular maintenance and major software releases. Configuration changes and new software releases must be tested to verify that the quality of the system is sustained. This is a continuous effort using a phased deployment approach, as discussed in *Modular and Iterative Methodology*.

This ongoing lifecycle of the application is an opportunity for continuous improvement in testing. As illustrated in the following image, the steps you can take to make iterative improvements to the testing process include the following:

1. **Identify Regression Test Suite.** First, a strategy for testing functionality across the life of the application is built by identifying a regression test suite. This test suite provides an abbreviated set of test cases that can be run with each delivery to identify any regression defects that may be introduced. The use of automation is particularly helpful for executing regression tests. By streamlining the regression test process, organizations are able to incorporate change into their applications at a much lower cost.
2. **Review Defects.** The testing strategy and its objectives should be reviewed to identify any inadequacies in planning. A full review of the logged defects (both open and closed) can help calibrate the risk assessment performed earlier. This review provides an opportunity to measure the observed risk of specific components (for example, which component introduced the largest number of defects).

   The testing strategy and its objectives should be reviewed to identify any inadequacies in planning. A full review of the logged defects (both open and closed) can help calibrate the risk assessment performed earlier. This review provides an opportunity to measure the observed risk of specific components (for example, which component introduced the largest number of defects).
3. **Conduct Post-Mortem.** A project-level final review meeting (also called a post-mortem) provides an opportunity to have a discussion about what went well, and what could have gone better with respect to testing.
4. **Revise Risk Scoring on Component Inventory.**
5. **Revise Test Plans and Cases.** Test plans and test cases should be reviewed to determine their effectiveness. Update test cases to include testing scenarios exposed during testing that were not previously identified.


Project Manager
Business Analyst
Application Tester
Application Developer
Start
Identify Regression Test Suite
Review Defects
Conduct Post-Mortem
Revise Risk Scoring on Component Inventory
Revise Test Plans & Cases
End

9 Implementing Siebel Open UI Keyword Automation Testing

Implementing Siebel Open UI Keyword Automation Testing

This chapter provides an overview of implementing Siebel Open UI keyword automaton testing in software development projects. It includes the following topics:

- *Overview of Siebel Open UI Keyword Automation Testing*
- *Process of Implementing Siebel Open UI Keyword Automation Testing*
- *Enabling Oracle Business Intelligence Publisher for Test Automation*
- *Siebel Test Automation Folder*
- *Extending Keyword Automation Capabilities*

Note: Enabling UPT is not compatible with test automation. To ensure that test scripts are correctly recorded (Unit Mode is typically used by test script authors to record test scripts), make sure there are no events with Action Set Name set to UPT (`'Action Set Name' = UPT`) and that the following system preferences are not used: UPTEEE1, UPTEEE2, and UPTEndToEndEventList.

Overview of Siebel Open UI Keyword Automation Testing

Siebel Open UI keyword automation testing is based on the Keyword Driven Framework. The Keyword Driven Framework is an automation testing framework which uses action words for testing.

The four main parts to each keyword automation step are:

1. Action to perform (Keyword)
2. Object to act upon
3. Input data
4. Choice of closing action.

A test case or script is made up of a set of sequenced test steps as shown, for example, in the following table.

Action or Keyword	Target Object	Inputs	Choice of closing action
Click a button	Repository name of an applet and button.	Any required closing action, such as clicking OK or Cancel, (after the button is clicked).	Click new

Action or Keyword	Target Object	Inputs	Choice of closing action
Input value	Repository name of applet and button.		Enter a value in the Name field.

Process of Implementing Siebel Open UI Keyword Automation Testing

Perform the following tasks to implement Siebel Open UI keyword automation testing for the following test scenario: *I want to automate a test script using keywords that will, for example, create a new product and add it to Quote.*

1. *Creating a Test Script*
2. *Adding Test Steps to Test Scripts*
3. *Capturing Automation Attributes for Test Steps*
4. *Grouping Test Scripts into a Test Set*
5. *Grouping Test Sets into a Master Suite*
6. *Configuring the Test Run*

Prerequisites

1. **Release** values
   a. Navigate to **Sitemap**, search for **Projects**. Click on **List**.
   b. Click + for new project.
   c. In the form applet, select **Organization Type** as **Service**.
   d. Enter a name, select **Status** as **Active**.
   e. Save the record.
   f. In the name value of the new record, select **Team Workbook**.
   g. Add all the User Ids that are required to create or use Test Scripts, Test Plans, Test Passes, etc.
   h. Go to **Test Scripts** view. The new project value should be available for selection in Release field.
2. **Team** values
   a. Navigate to **Sitemap**, search for **Administration - Group**.
   b. Select **Internal Divisions**.
   c. Create as many 'Divisions' as needed. These can be selected on Test objects, as Team.

Creating a Test Script

The following procedure describes how to create a new test script. Make sure that the Siebel application is up and running before starting this task.

This task is a step in *Process of Implementing Siebel Open UI Keyword Automation Testing*.

To create a test script

1. Navigate to Sitemap, Release screen, then the Test Scripts view.
2. Click the plus (+) icon to create a new test script.
3. In the form that appears, complete the fields with the values shown in the following table.

Only fields marked with an asterisk (*) in the UI are mandatory.

Field Name	Value	Description
Name*	xyz	Name of the test script.
Test Script Description	Test	Description of the test script.
Status*	Active	Status of the test script
Reference ID	xyz_abc	The reference id of the test script.
Team name	ABC	The name of the team.
Release	IP2017	The name of the specific release.
Test Environment	All, Demo etc	Specifies the test environment.
Pre Condition	Text	Notes on preconditions for the test.
Steps to execute	Text	Notes steps to be covered in the test.
Input Data	Text	Notes in input data.
Expected Output	Text	Notes on expected results.
Post Processing	Text	Notes on post processing.

4. From the Test Scripts menu, choose Save Record.

   A new test script is created.

Adding Test Steps to Test Scripts

The following procedure describes how to add test steps to test scripts. Make sure that the Siebel application is up and running before starting this task.

This task is a step in *Process of Implementing Siebel Open UI Keyword Automation Testing*.

To add test steps to test scripts

1. Navigate to Sitemap, Release screen, then the Test Scripts view.
2. Drill down on the name of the test script to which you want to add test steps.
3. Click the plus (+) icon to create a new test step.
4. In the form that appears, complete the fields with the values shown in the following table.

   Note the following:

   - The first test step (or test sequence number 1) must be the Launch keyword.
   - After you select a Keyword, the fields are rendered dynamically as required for the keyword.

Field Name	Value	Description
Test Step Sequence	1	Sequence number for the test step.
Description	Login	Description of the Keyword
Keyword	Launch	Enter this keyword to log into the application.
Paste Attributes	Not Applicable	Not Applicable.
Component Alias	Can be any name	ID associated with the test script.
User Name	CCHECNG	Name of the user.
Clear Browser	Y	Enter this value to clear the browser cache.
Screenshot Required	Not Applicable	Select this option to capture the corresponding step performed in the test step.

5. Save the record.

Note: The Keyword Reference lists all the keywords and provides you with information on how to use keywords with examples.

Note: IPH1;IPH2;IPH3 are read-only place holder notations for the Input field.

Note: All the references to user names, user ids or passwords in this Testing guide are meant for Testing and test automation purposes only. It is responsibility of the users to ensure that production or real user ids and passwords are not stored in Test Automation records or artifacts, such as files, database, etc.

Test Setps View

The Test Steps View is an applet similar to the Test Steps applet under Test Scripts.

This view exposes the Test steps at the first level. The records are in read-only mode and you cannot create any records here. You can only query and view the records.

The Test Steps View exposes the inputs and RNs given in the keyword as separate columns. Import/Export is supported for this applet.

In case the input for a keyword is changed, users can export records for that particular keyword, modify the input value and import it. Using the Test Sets view, you can also see how frequently each keyword is used – for example, by querying on the keyword, you get the number of records for that keyword.

To edit a Test Step, navigate to Sitemap, Release screen, then the Test Steps view.

Capturing Automation Attributes for Test Steps

The following procedure shows you how to use the attributes pop-up window to capture automation attributes for test steps.

This task is a step in the *Process of Implementing Siebel Open UI Keyword Automation Testing*.

Before capturing automation attributes for test steps, the following prerequisites apply:

- Make sure that the following behavior settings are configured (click Tools, select User Preferences and then Behavior).
  - Set Navigation Control to Tab.
  - Set Theme to Aurora.
- Make sure that the Siebel application is running in AutoOn mode.
- Make sure that the URL command is set to the AutoOn mode, for example, as follows: `http://machinenumber:domainname:portname/siebel/app/callcenter/enu?SWECmd=AutoOn.`

  Being in AutoOn mode helps you to capture the automation attributes when navigating through the application.

To capture automation attributes for test steps

1. Start the Siebel application in a new browser.

2. To capture the automation attribute for a specific object or control, do the following:
   a. Position the mouse over the object or control for which the automation attributes are needed (for example, the plus (+) icon).
   b. Press the CTRL key on your keyboard and right-click your mouse button simultaneously.

      The Siebel Automation Information dialog box appears.
   c. Copy the string value from the COPY STRING field and paste it into the Paste Attributes field of the Test Step.
   d. Tab out of the Paste Attributes field in the Test Step.

      The data will be automatically populated in the target object fields. For example: the AppletRN and ItemRN fields.

Grouping Test Scripts into a Test Set

The following procedure shows how to group test scripts into a test set.

This task is a step in the *Process of Implementing Siebel Open UI Keyword Automation Testing*.

To group test scripts into a test set

1. Navigate to Sitemap, Release screen, then the Test Set view.
2. Click the plus (+) icon to create a new Test Set.
3. In the form that appears, complete the fields with the values shown in the following table.

Field	Description	Sample Value
Name	Name of the test set.	Xyz
Description	Description of the test set.	Test
Test Plan Name	Name of the test plan.	Test_xyz
Release	Name of the release.	1718
Team	Name of the team.	Transformers
Status	Status of the test set.	Active
External ID	The external ID associated with the test set.	Testplan_1

4. Drill down on a value in the Name field and then click Add Test Scripts (or the plus (+) icon) to associate test scripts with the selected Test Set.

5. Update the Sequence field for each associated test script.

Note: As of Siebel CRM 20.9 Update, the Add Iteration, Remove Iteration and Scenario Iteration buttons pertain to data driven testing. For more information on the different iteration types, see *Data Driven Testing*.

Grouping Test Sets into a Master Suite

The following procedure shows how to group test sets into a Master Suite.

This task is a step in the *Process of Implementing Siebel Open UI Keyword Automation Testing*.

To group test scripts into a Master Suite

1. Navigate to Sitemap, Release screen, then the Master Suites view.
2. Click the plus (+) icon to create a new test set.
3. In the form that appears, complete the fields with the values shown in the following table.

Field	Description	Sample Value
Name	Name of the test set	xyz
Description	Description of the test set	Test
Status	Status of the test set	Active
Test Plan ID	ID associated with the test set	Test1_001
Team	Specifies the corresponding name of the team.	Titans, Transformers, and so on.
Release	Specifies the appropriate release version.	17.0

4. Drill down on a value in the Name field, then click Add Test Sets (or the plus (+) icon) to associate test sets with the selected Master Suite.
5. If the master suite contains the following keywords, you must add the required files in the respective folders.
    - Folder Name: Resources
    - Keywords:
        - Inboundwebservicecall
        - Invokeperl
        - Serverconfig
        - fileupload

After including the files, you can zip the folder (that is, Resources.zip) and attach Resources.zip in the MasterSuite Attachments Applet.

Viewing Master Suites associated to a Test Set

To view master suites associated to a Test Set, follow the steps:

1. Navigate to **Test Sets** List View and drill down on a record.
2. In the next level, select or click **Master Suites**.
3. This applet displays all the Master Suites to which the Test Set is associated to.

Configuring the Test Run

Configuring a test run involves creating a test execution record and running the test scripts.

This task is a step in the *Process of Implementing Siebel Open UI Keyword Automation Testing*.

Creating the Test Execution Record

1. Navigate to Site Map, Release screen, then the Test Execution view.
2. Click the plus (+) icon to create a new configuration record.
3. Complete the fields with the values shown in the following table.

Field	Description	Sample Value
Master Suite id Master Suite	Select a Master Suite record from the drop-down list.	The Row id and the name of the selected Master Suite appear in the respective fields.
Application Version	The version number of the application. To create the application version records, navigate to Site Map, Quality screen, then the Release Product Administration view. **Note:** You can create records in this view, and the same would be available as Application Version values in the Test Execution view.	1.2.0
Notify	Check this box to enable Notifications on Test Execution Status changes. For example, when Test Execution Status changes to Completed, following record will show on Notifications Summary. `Test Run #: <Run id of Test Execution> ( <name of Master Suite> ) Test`	Checked or Unchecked

Field	Description	Sample Value
	`Execution Status: <Status value> mm/dd/yyyy` **Example**: `Test Run #: 88-ZFU21 ( MySuite01 ) Test Execution Status: Completed 07/17/2023`	
Run Reference	Enter value of Label from Jenkins Node or group of Nodes. Ensure the value is an exact match with Node Labels, including case. Ensure there are no spaces in the Label text. Test Execution record will be redirected to run on a matching Node only. This field is optional - default is blank. Refer to section *Configuring the Siebel Test Execution Job*. **Note:** When Run Reference Field is used, ensure to provide Parameter `--runReference=<Same label value as in Run Reference field>` on Test Execution job (or build step) configuration.	Client_Pool_A, myclientxyz

4. In the Server Credentials applet, select the Application Type (such as Desktop_Chrome), the URL details, and the OS (Operating System) and Language.
5. Click Schedule Run, and the status will be updated to Requested.

   Assuming that the Jenkins Server setup is ready, the status will be updated to Scheduled.

   The attachment files are created in the Attachments applet. For example: batchconfig.xml, mastersuite.csv, and Resources.zip.

Enabling Oracle Business Intelligence Publisher for Test Automation

To enable Oracle Business Intelligence Publisher (BIP) for Test Automation, you must enter the BIP specific parameters. As a prerequisite, ensure that BIP is integrated with Siebel prior to test execution.

The format for entering the parameters in Batch mode and Unit mode are given in the following subsections.

Batch Mode

Enter the information in the following table in the Test Execution screen.

Field	Description	Sample Value
BIP Outbound WS URL	BIP outbound Web service.	http://slcxxxxxx:9502/xmlpserver/services/publicreportservice_v11
BIP Server Machine	Name of the BIP server.	slcxxxxxx
BIP Server Port	Port number for BIP server.	9502
BIP XMLP Server Machine	Name of the XMLP server.	None.
BIP XMLP Server Port	XMLP port number.	None.
BIP Server User	BIP server user name.	None.
BIP Server Password	BIP server password in plaintext.	None.

Unit Mode

Update the following parameters in the unitconfig.xml file:

1. `<BIP-OUTBOUNDWEBSERVICE>http://slcxxxxxx:9502/xmlpserver/services/publicreportservice_v11</BIP-OUTBOUNDWEBSERVICE>`
2. `<BIP-SERVERNAME>slcxxxxx:9500</BIP-SERVERNAME>`
3. `<BIP-SERVERXMLP>slcxxxxx:9502</BIP-SERVERXMLP>`
4. `<BIP-USERNAME>SADMIN</BIP-USERNAME>`
5. `<BIP-PASSWORD>ldap</BIP-PASSWORD>`

Note: The BIPserver.jar file is packaged in DISA and available in the following (folder) location: `C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\lib`.

After installing DISA, the bip.pl will be available in the following location:

```
<DISA Install location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\perl
```

To perform BIP Automated script execution, you need to move the bip.pl to the following (folder) location:

```
<DISA Install location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Resources\invokeperl
```

Using BIP Automation, you can verify all Interactive, Scheduled, and Parameter based report generation.

Siebel Test Automation Folder

The Siebel Test Automation folder structure contains the required folders for executing the csv script. The Framework folder contains all the required JARS and library files. The following folder structure is available when you install DISA.

Folder	File Type	Comment	Execution
Framework\	Exe	Do not modify	Contains necessary Exe files which are used in the SiebelTestAutomation Framework
Framework\	Lib	None	Contains necessary Library files (JAR files) which are used in the SiebelTestAutomation Framework
Framework\	Perl	None	Contains necessary Perl scripts which are used in SiebelTestAutomation Framework
Framework\	SiebelTestAutomation.jar	None	Compiled executable JAR which drives KWD execution
Extensions	.jar	CustomExtension KeywordActionHandler implementations	Placeholder for CustomExtension JAR files.
Reports	N/A	Post- execution file	The folder gets created post execution, generates execution results for review. **Note:** For batch runs, this folder is created under the SiebelTestAutomation\TestExecutions\ <RunId>\ folder.
Resources\	N/A	Contents of Resources.zip are unzipped into this folder.	**Note:** For batch runs, this folder is created under the SiebelTestAutomation\TestExecutions\ <RunId>\ folder.
Resources\	fileupload	Script authoring file	Folder containing user developed files needed for fileupload.
Resources\	inboundwebservicecall	Script authoring file	Folder containing user developed files needed for inboundwebservicecall.
Resources\	invokeperl	Script authoring file	Folder containing user developed files needed for invokeperl.
Resources\	toolsconfig	Script authoring file	Folder containing user developed files needed for toolsconfig.
Resources\	serverconfig	None	Folder containing user developed files needed for serverconfig keywords.

Folder	File Type	Comment	Execution
Scripts\	Samplescript.csv	None	Place holder for csv scripts. **Note:** For batch runs, this folder is created under the SiebelTestAutomation\TestExecutions\ <RunId>\ folder.
TestExecutions\	N/A	Batch runs.	Folder where individual batch run folders are created – each folder is named according to the "Run Id" (row id) of the test execution record.
SiebelTestAutomation \TestExecutions \ <RunId> \batchconfig.xml	XML (.xml) file.	Pre-setup file	XML File which contains the configuration details of the execution (URL, username password, script location). The batchconfig.xml file will be copied by the Jenkins scheduler if it is an STE run. The unitconfig.xml file will be used for the UPT run.
SiebelTestAutomation \TestExecutions\ <RunId>\User_opted_ operation.txt	Text (.txt) file.	Text file	Text file which contains information like Pause, Resume and abort. The user has the option to Pause/Resume/ Abort the current execution by updating the value in the User_opted_operation.txt file.
SiebelTestAutomation \TestExecutions \ <RunId>\Log_ randomnumber.log	Log (.log) file.	Log file	Text file which contains log messages by Framework for debug purposes.

Extending Keyword Automation Capabilities

Keyword Automation, by default, supports only a typical login screen using the Launch keyword. Because Siebel deployments often use Single Sign-On (SSO) and/or Multi-Factor Authentication and since SSO login screens are company-specific, you can now use the CustomExtension keyword to handle SSO login. Similarly, you can extend the keyword capabilities to address other, specific processing needs of your organization through CustomExtension.

You can use the CustomExtension keyword, which runs a custom extension JAR file, to enable support for custom requirements, such as the following:

- Single Sign-On
- Non-OpenUI controls
- Embedded iFrames
- Integration dependency processing

By using the CustomExtension keyword along with custom scripting, you can implement automation as part of the Siebel test script execution (ExecuteOperation). This helps to extend automation capabilities and increases overall test coverage.

The following subtopic shows how to build a CustomExtension plugin. For more information about using the CustomExtension keyword, see *CustomExtension*.

Building a CustomExtension Plugin

The typical instructions to build a CustomExtension plugin (for Login) are as follows:

1. Locate the following files, and then add both JAR file paths to the class path while compiling the CustomExtension project:
   - Go to the `<DISA_HOME>/DesktopIntSiebelAgent/plugins/SiebelTestAutomation/Framework` folder and locate the SiebelTestAutomation.jar file.
   - Go to the `<DISA_HOME>/DesktopIntSiebelAgent/plugins/SiebelTestAutomation/Framework\lib` folder and locate the selenium_xxx.jar file (where xxx indicates the version number).
2. Create a new Java Package with the new class implementing the interface KeywordActionHandler and other related files if any. To avoid any potential conflict, it is recommended that the custom extension class have a unique package name.
3. Implement method ExecuteOperation in the new class with a return type boolean (true/false value) and the following parameters: InputVariables and OutputVariables.

   For example, the following sample login Java class implements the KeywordActionHandler interface as specified - the Java class is designed to handle a Web page with the following fields:
   - Input field called `username`
   - Input field called `password`
   - Anchor element with the value `Login`

   The ExecuteOperation function identifies the `username` field and uses the sendKeys API from Selenium to fill in the `username` for the login page, and similarly for the password field. Further, it identifies the anchor element with `Login` value and performs a click operation on it. Note that the keyword framework injects the relevant WebDriver object as it executes the plugin based on the keyword execution from the test script.

```
package com.siebel.automation.keywordFrameworkcust;

import com.siebel.automation.keywordFramework.KeywordActionHandler;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import java.util.ArrayList;

public class login implements KeywordActionHandler {

 public boolean ExecuteOperation(WebDriver driver,ArrayList<String> inputvariables,ArrayList<String>
 outvariables)
 {
 driver.findElement(By.xpath("//input[@name='username' ]")).sendKeys("username");
 driver.findElement(By.xpath("//input[@name='password' ]")).sendKeys("password");
 driver.findElement(By.xpath("//a[@value='Login' ]")).click();
 return true;
 }
}
```

4. Set META-INF for the newly created plugin. If META-INF does not exist:
   - Create a folder named META-INF in the plugin source folder (the root folder of package folders).
   - Create a new text file named meta.txt.

   `Class-Path: ../../Framework/SiebelTestAutomation.jar ../../Framework/lib/selenium-serverstandalone-xxxx.jar` where XXXX is the appropriate selenium server standalone version from your DISA installation.
5. Compile the class and pack the META-INF folder in to a JAR file.
6. Deploy the JAR file to the following folder:

   `<DISA_HOME>/DesktopIntSiebelAgent/plugins/SiebelTestAutomation/Extensions`
7. Add a new CustomExtension test step with the appropriate classname extension to the test script and validate the functionality.

10 Usage Pattern Tracking and Conversion to Keyword Scripts

Usage Pattern Tracking and Conversion to Keyword Scripts

This chapter includes information about usage pattern tracking (UPT) in Siebel Business Applications. It includes the following topics:

About Usage Pattern Tracking

Usage pattern tracking allows administrators to review details about when, and how often, users' access features in a Siebel application. To capture this information, administrators configure a component job to run at a scheduled date and time, or at time intervals that they specify.

For more details, refer to About Usage Pattern Tracking in *Siebel Applications Administration Guide* .

Setting Up the Automation Adapter

Before you get started with the task of recording a functional flow, you must complete the procedures in this topic to set up the system preferences and profile configuration parameters for Usage Pattern Tracking. You must also delete UPT action sets, runtime events and system preferences.

To set the system preferences for Usage Pattern Tracking

1. Navigate to the Administration - Application screen, then the System Preferences view.
2. Set the system preferences as shown in the following table:

Name	Value
Enable UPT	True
Enable UPT Context	True
UPTMax Record Cache	100

3. To implement the system preferences, you must restart the Siebel Gateway and Siebel Services.

To set the profile configuration parameters for Usage Pattern Tracking

1. Navigate to the Administration - Server Configuration screen, then the Enterprises view.
2. In the Enterprises Server applet, select the enterprise server you want to configure.
3. Click the Profile Configuration tab. Query for the profile AutomationSubSys.

Name	Alias	Datatype	Value
Class Path	CLASSPATH	String	N/A
Container URL	CONTAINERURL	String	The value for the jbs. For example: `CONTAINERURL=http://localhost:<ContainerPort>/siebel/jbs` The `<ContainerPort>` value should be taken from the following: `<Siebel_Install_Location>\ses\applicationcontainer_external\conf\server.xml<Connector port="4730" protocol="HTTP/1.1"` For Example: `http://localhost:4730/siebel/jbs`.

4. To implement the profile configuration changes, you must restart the Siebel Gateway and Siebel Services.

To configure the AutomationSubSys profile via the server manager command

- Enter the following server manager command to configure the profile for AutomationSubSys:

```
change param CONTAINERURL=http://localhost:<ContainerPort>/siebel/jbs for named subsystem
 AutomationSubSys
```

The following table lists the runtime events configured out of the box.

Sequence	Object Type	Event	Action Set Name	Conditional Expression
2	Application	ViewActivate	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	ViewDeactiv	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	WebLogin	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	WebLogout	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Applet	PreInvokeMe	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	Recording	KWD	`GetProfileAttr("Enable Recording") = "True"`
2	Application	InvokeServic	KWD	`GetProfileAttr("Enable Recording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	UPTClientSc	KWD	`GetProfileAttr("EnableRecording") = "True" AND GetProfileAttr("RecordingOn") = "True"`
2	Application	InvokeMetho	KWD	`GetProfileAttr("EnableRecording") = "True" AND GetProfileAttr("RecordingOn") = "True"`

Note: Any other UPT Events should be disabled or inactivated before using the UPT feature. Refer to the UPT documentation for Event Details.

Note: If 18.9 UPT enhancements are applied via Repository Upgrade, refer to *Siebel Applications Administration Guide* for UPT enhancements.

To delete UPT action sets, runtime events and system preferences

1. To delete events set to UPT:
   - Navigate to the Administration - Runtime Events screen, then the Events view.
   - Delete all the Events with Action Set Name set to UPT: `'Action Set Name' = UPT`.
   - Click Applet Menu and then click Reload Runtime Events.
2. To delete action sets set to UPT:
   - Navigate to the Administration - Runtime Events screen, then the Action Sets view.
   - Delete all the Events with Action Set Name set to UPT: `'Action Set Name' = UPT`.
   - Click Applet Menu and then click Reload Runtime Events.
3. To delete UPT system preferences:
   - Navigate to the Administration Application screen, then the System Preferences view.
   - Delete the following system preferences: UPTEEE1, UPTEEE2, and UPTEndToEndEventList.
4. For changes to take effect, log out of the application and then back in again.

   Restart Enterprise Server for system preference changes to take effect.

Configuring the UPT and KWD Log Directory for Multiple Servers

This topic provides you with the information on configuring the UPT and KWD log directory for multiple servers.

The following procedure shows you how to configure Automation Application Directories in a multiple server or hybrid environment (for example, Microsoft Windows and UNIX).

To configure the UPT and KWD log directory

1. Create a shared Common Internet File System (CIFS) between the Windows and Unix servers.

   On Windows for example, a folder called automation needs to be created (for example, `c:\automation`) and the Siebel Servers are installed in `<Siebel Install Location>\ses\siebsrvr`.

   On UNIX for example, a folder called automation mounted as `/somepath/automation` (which points to the Windows folder) and Siebel Servers are installed under `<Siebel Install Location>\ses\siebsrvr`.
2. Create two subfolders for UPT and KWD.

- `mkdir c:\automation\upt`
- `mkdir c:\automation\kwd`

3. Use the mklink in Windows to link the `siebsrvr\upt` and `siebsrvr\kwd` to the shared folder. `c:\automation\upt` and `c:\automation\kwd` respectively as follows:
   - `mklink /D <Siebel Install Location>\ses\siebsrvr\upt c:\automation\upt`
   - `mklink /D <Siebel Install Location>\ses\siebsrvr\kwd c:\automation\kwd`
4. In UNIX, link the `siebsrvr/upt` and `siebsrvr/kwd` to the shared folder `/somepath/automation/upt` and `/somepath/automation/kwd` by using the following commands:
   - `ln -s /somepath/automation/upt <Siebel Install Location>/ses/siebsrvr/automation/upt`
   - `ln -s /somepath/automation/kwd <Siebel Install Location>/ses/siebsrvr/automation/kwd`

Using the Automation Toolbar

The test automation framework provides the ability to record a functional flow in Siebel using Automation Toolbar. The toolbar features buttons for each of the following actions:

1. Start Recording,
2. Pause/Resume Recording,
3. Stop Recording,
4. Generating the KWD Script.

Impact of Usage Pattern Tracking Enhancements in Siebel CRM 18.9 Update

If the usage pattern tracking enhancements in Siebel CRM 18.9 Update are applied via Repository Upgrade, then note the following:

- Based on Unit Mode and Bulk Mode settings, the user may or may not see the Recorder button on the Application Tool bar.
- If the 'Input Capture' Action is set to False to mask sensitive information, KWD scripts will be generated without any input values, causing them to fail on play back. Input values can be added in Test Script View.
- Enabling UPT is not compatible with test automation. To ensure that test scripts are correctly recorded (Unit Mode is typically used by test script authors to record test scripts), make sure there are no events with Action Set Name set to UPT (`'Action Set Name' = UPT`) and that the following system preferences are not used: UPTEEE1, UPTEEE2, and UPTEndToEndEventList.

For more information about usage pattern tracking, see *Siebel Applications Administration Guide* .

Recording the Functional Flow

The test automation framework allows you to record a functional flow in Siebel using the Automation Toolbar. Make sure that the Siebel Application is in AutoOn mode before recording a functional flow, as shown in the following URL:
`https://<URL>:<port>/siebel/app/callcenter/enu?SWECmd=AutoOn.`

To record a functional flow

1. Click the Camcorder icon on the Automation toolbar to open the recording panel.
2. Click Start to start a recording session.
3. Click Pause at any time to pause the script recording, then click Resume to resume recording again.
4. Click Stop to end the recording session.

   The Generate button becomes active only when you stop the recording session (that is, after you click Stop).
5. Click Generate to generate the KWD script.
6. Click Scripts on the Automation toolbar to open the Scripts Pane, where you can see the scripts that have been generated.

Note: The buttons on the Automation toolbar are enabled or disabled based on the actions that you perform during a recording session. For example, when you click Start, the Pause and Stop buttons are enabled. When you click Pause, the Resume and Stop buttons are enabled. When you click Stop, the Start and Generate buttons are enabled.

Renaming the Scripts

After generating the scripts, you can rename the scripts in the script pane.

Note: You can provide any name to the generated scripts.

To rename the script

1. Open the Script Pane
2. Update the File name field of the script to rename it.
3. Save the record.

You can also choose to delete the scripts if that are not required.

To delete the script

1. Open the Script Pane
2. Select script o be deleted.
3. Click Delete.

Setting Up DISA

You must download and install the latest version of DISA before proceeding with the task of playing the test scripts. For more information on installing and configuring DISA, refer to *Desktop Integration Siebel Agent Guide* .

Plugin Configurations

After installing the Plugin for Siebel Automation with DISA, you must update the unitconfig.xml file. The unitconfig.xml file is available in the following location:

```
<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\ unitconfig.xml
```

Update the XML tags under Mandatory_Params as shown in the following table.

XML Tag	Description
URL	A fully qualified URL of the application will be run while playing the script [AutoOn mode is not required here]. The URLs will be similar, for example, to the following: **`http://xyzcjzg.us.oracle.com:14440/siebel/app/fins/enu`** When you click Play, this URL will be prepopulated in the Popup that appears. The Popup also contains a DetailedReport check box, which you can select (check) or deselect (uncheck) as follows: • Select the DetailedReport check box to capture detailed test results and screenshots during unit mode/single test script playback. • Deselect the DetailedReport check box to turn off detailed test results and screenshot capture during unit mode/single test script playback. For information on how to configure report generation and screenshot capture during the test script automation batch run, see *Configuring the Siebel Test Execution Job*.
USERNAME	User name to login to the application URL provided in the URL xml tag.
APPLICATIONTYPE_BROWSER	The platforms and supported browsers for playback are: • Desktop_FireFox: Playback will be on Windows platform with Firefox browser. • Desktop_Chrome: Playback will be on Windows platform with Chrome browser. • Desktop_IE: Playback will be on Windows platform with IE browser. • Mobile_Safari: Playback will be on MAC IOS simulator platform with mobile safari browser. You must enter the optional Parameters in the unitconfig.xml. • Mobile_Chrome: Playback will be on the Android device with chrome browser. You must enter the optional Parameters in the unitconfig.xml. • Mobile_SM_IOS: Playback will be on MAC IOS simulator platform with siebel mobile client. You must enter the optional Parameters in the unitconfig.xml. • Mobile_NativeBrowser: Playback will be on Android emulator with default native browser. You must enter the optional Parameters in the unitconfig.xml scripts.

To play the functional flow in Mobile or Simulators

- Download the java-client-4.1.1.jar (https://search.maven.org/#search|gav|1|g%3A%22io.appium%22%20AND%20a%3A%22java-client%22 download version 4.1.1) and copy it to the `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\lib` folder.
- Download the *plink.exe* and pscp.exe and copy it to the `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\exe` folder.
- Add the appropriate values for the OPTIONAL_PARAMS parameter in the unitconfig.xml file.
    - MOBILE-IP: IP address of Android device, simulator, emulator
    - MOBILE-OS: Android or iOS based upon browser.
    - MOBILE-PORT: 4444 (The port number specified here is for example purpose).
    - MAC-USERNAME: Mac Machine User Name
    - MAC-PASSWORD: Mac Machine Password

Browser Configuration Settings

The following browsers are supported for running test scripts: Firefox, Chrome, Microsoft Edge, and Internet Explorer. It is recommended that you use the latest version of one of these browsers for which a compatible Web driver is available.

To run the test scripts in different types of browsers, you must update the browser settings.

> **Note:** Ensure to disable the monitor or screen Touch mode feature for the targeted browser before proceeding with the automation.

Firefox

To run the test scripts using Firefox, download the geckodriver.exe file from *https://github.com/mozilla/geckodriver/releases* and copy it to the following location: `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Drivers`.

Before playing test scripts in the Firefox browser, you must add the Security Exception window to the Location field and add the following address: `https://localhost:18443`.

> **Note:** The port 18443 must be changed to the port number which you configured for disa.exe in the config.properties file.

Chrome

To run the test scripts using Chrome, download the chromedriver.exe file from *https://sites.google.com/a/chromium.org/chromedriver/downloads* and copy it to the following location: `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Drivers`.

> **Note:** It is recommended that you download the driver file after checking the browser version.

Microsoft Edge

To run the test scripts using Microsoft Edge, download the appropriate Microsoft Edge Driver version for Windows from *https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/* and copy it to the following location: `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Drivers.`

Internet Explorer

To run the test scripts using Internet Explorer (IE), download the IEdriverserver.exe file and copy it to the following location: `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Drivers.`

You must disable the Drag and Drop option, update the Compatibility View settings and Security Settings.

Note: *Drag and Drop* is a method of moving something (for example, files or images) from one place on the UI to another using a mouse or similar device.

To disable the Drag and Drop option

1. Navigate to the Internet Options in IE 11.
2. Navigate to the Security tab.
3. Click Custom Level.
4. Navigate to the Miscellaneous Section and disable the following options:
    - Allow dragging of content between domains into separate windows.
    - Allow dragging of content between domains into the same window.

To update the Compatibility View Settings

1. Navigate to Compatibility View Settings option in IE 11.
2. Deselect the Display intranet sites in compatibility View check box.
3. Deselect the Use Microsoft compatibility lists check box.
4. Click Close.

To update the Security Settings

1. Navigate to the Internet Options in IE 11.
2. From the Tools Menu, click Internet Options.
3. Click Security Tab.
    - Choose Internet Zone and select Enable Protected Mode.
    - Choose Local Internet Zone and select Enable Protected Mode.
    - Choose Trusted Sites Zone and select Enable Protected Mode.
    - Choose the Restricted Sites Zone and select Enable Protected Mode.
4. Click OK after selecting Enable Protected Mode option for each of the zones.

Note: Copy the latest IE Driver "IEDriverServer" from `http://www.seleniumhq.org/download/` and place it in the `<DISA Installation Location>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Drivers` folder.

Validating the Scripts

Script validation is done before you can start playing the scripts from the Scripts Pane. Script validation ensures that all the steps in the script file are syntactically correct and the required attributes of the keyword are recorded properly.

If there are any attributes missing in the script step, the validation fails and an error message is displayed along with validation log file name.

The Validation log file contains the list of errors. Each line in the validation log file has the following details.

- Line Number. This is line number in a script file.
- Missing attributes. Comma separated list of attributes which are missing in Action, Target object Input or End Action.

The Validation log file will be created in the following location: `<SIEBEL_INSTALLATION_PATH>\ses\siebsrvr\log`.

Playing the Scripts

The scripts that are generated after a recording (see *Recording the Functional Flow*) are available in the Scripts Pane. You must download and install DISA (Desktop Integration for Siebel Applications) on the client machine to play the scripts.

To play a script from the Scripts Pane:

1. Click the Camcorder icon on the toolbar, then click Scripts to open the Scripts Pane.

   This Scripts Pane shows the list of scripts that have been generated.
2. Select the script you want to play and then click Play.

   > **Note:** You can play only one script at a time. The Play button will be disabled if you select multiple scripts simultaneously or if there is some issue with the DISA setup.

3. The Play Status will be updated to Success or Fail based upon the play result.

   > **Note:** If a script is renamed while playing it, the play status will not be reflected.

The play process is initiated in a separate browser window and the results are displayed when finished. The same script can be played more than once instead of using the replay process.

To play an imported or modified script from the Release Screen

1. Navigate to the Sitemap, Release screen, then the Test Scripts view.
2. Drill down to script.

   The play button will be available in the Form Applet.
3. Select the script you want to play and then click Play.

   The play process is initiated in a separate browser window and the results are displayed when finished.

Condition Expression for Test Steps

In order to selectively use a Test Step of a Test Script at run time, use **Condition** field in your **Test Script** > **Test Steps Applet**. This **Condition** field takes simple expression and evaluated using ECMAScript compliant expression engine, at run time for each Test Step. If the expression evaluates to true, the Test Step is run, else it is skipped.

Use **Condition expression** to run or skip a Test Step at run time. Default value is blank, and it evaluates to True.

1. Add **Conditions** to your **Test Script** > **Test Steps**, that determines if a particular Test Step is to be run or not.
2. At run time the actual values of variables are replaced in the expression, and is evaluated for true or false.

Note: Syntax of expression is important and no prior validation of expression is performed.

Note: By default, if **Condition** field is blank, then the Test Step is not skipped.

Rules for specifying Condition

1. Max length of field is 500. Ensure that the variable name does not have spaces.
   For example, variable `@Test Variable` is invalid, but `@TestVariable` or `@testvariable` or `@test_variable` are valid.
2. Single space is the delimiter between the operands and operators used in condition expression. For example, `@test > 10, @name = 'abc' OR @name = 'xyz'.`

   Note: Text values to be enclosed within single quotes, as in the examples. Double quotes are not supported.
3. Following values in **Condition** will evaluate to TRUE: `true or TRUE or no value` (blank), whereas `false or FALSE or 0(zero)` evaluate to FALSE.
4. Following operators are supported, `>,<,=,<>,>=,<=, AND, OR.`
5. Ensure the variables used in **Condition** are initialized with appropriate values, else the script fails with an error. Examples for Condition:
   - `@env = 'DEV'`
   - `@env = 'UAT'`
   - `@conf > 0 AND @conf < 50`

To initialize variables before a Test Script starts to run, use Parameters applet on Test Execution (or add Parameters section in `unitconfig.xml` for single Test Script Play from Test Scripts View). Add Name and Value pairs.

For example: To initialize a variable `@balance` with value `100`, add a Parameter with name as balance and value of 100. Framework creates variables for each of the **Name** in **Parameters applet**, and initializes with corresponding Value. Such parameters or variables can be directly used in Test Step Condition and/or in Data Set Condition expression for comparison or input value.

```
<PARAMETERS>
 <PARAMETER>
 <NAME>env</NAME>
 <VALUE>DEV</VALUE>
 </PARAMETER>
 <PARAMETER>
 <NAME>ROI</NAME>
```

```
<VALUE>20</VALUE>
</PARAMETER>
</PARAMETERS>
```

StartLoop and EndLoop Logical Looping Constructs

This topic explains both StartLoop (begins the loop; no inputs) and EndLoop (ends the loop; takes inputs in the Inputs column).

Looping is confined to a single test script. Nested or overlapping loops are not supported.

Repeat a block of test steps until a condition becomes true or a maximum iteration count is reached. This eliminates long static waits and supports asynchronous or event-driven scenarios.

Using Logical Looping Constructs

- Poll for backend or workflow-driven updates (for example, query a record until a status changes) and then continue the test.
- Replace brittle sleep/wait statements with controlled, condition-based repetition.

Keyword Usage and Inputs

- StartLoop
  - Inputs: A variable name, to get iteration count, automatically initialized to 1 and incremented each iteration.
  - Can be the first step in a test script.
- EndLoop
  - Inputs (provided in the Inputs column, not the Condition column):
    - Condition: A Boolean expression that, when it evaluates to true, exits the loop.
    - Max_loop_iterations: A positive integer cap; the loop exits when this count is reached if the condition has not become true.
  - Script-level inputs take precedence over profile defaults.

The loop exits immediately when the Condition evaluates to true; otherwise, it runs up to Max_loop_iterations.

Defaults and Guardrails

- You can configure a test execution profile-level loop iterations limit to cap loops if EndLoop parameters aren't set.
- If neither EndLoop parameters nor a profile cap are provided, a small default cap applies to prevent infinite loops.

Data-driven tests: Steps inside the loop can consume dataset values; each iteration uses the current dataset row according to existing behavior.

Reruns: Looping does not change rerun behavior; on rerun, the script starts from iteration 1.

Validation and Restrictions

- Nested or overlapping loops are not permitted. Attempting to start a new loop before ending the previous one results in an error.
- If StartLoop's test step condition (if used) evaluates to false, the entire loop block is skipped, EndLoop is ignored, and execution continues after the block.
- StartLoop and EndLoop must be in the same test script; cross-test-script loops are not supported.

Error Handling

- EndLoop condition expression:
  - Syntax errors or use of uninitialized variables cause a clear failure message at runtime.
- Invalid Max_loop_iterations (for example, negative or non-numeric):
  - Treated as invalid input; the loop does not execute and an error is reported.
- Orphan EndLoop (no preceding StartLoop):
  - Reported as a failure; the script continues with subsequent steps, and the test script is marked failed.
- Multiple StartLoops before a matching EndLoop:
  - Reported as an error (nested/overlapping loops are not supported).

Reporting

- Per-iteration logging:
  - Log "Start Iteration: n" at loop begin and "End of Iteration: n. Expression evaluated: <value>" at loop end.
- Results import:
  - Only the latest iteration's results (pass/fail) are imported into the Test Results view.
- Mid-iteration failures:
  - Log all executed steps up to the point of failure; remaining steps in that iteration are not executed.

Configuration

1. Test Execution Profile:
   a. Set "Loop iterations limit" value, that will take effect in absence of a value for Max. iterations in EndLoop
2. If and when provided in EndLoop Test Step, it will override profile value.

Best Practices

- Always specify at least one exit guard (Condition or Max_loop_iterations); preferably specify both.
- Include a refresh or re-query step inside the loop so the condition evaluates updated data each iteration.
- Keep iteration caps and timeouts reasonable to avoid long-running tests.
- Validate condition expressions in unit or batch runs before production runs, and initialize variables in profiles or earlier steps.

Examples

Example 1: Loop until condition

- StartLoop
- Query record or perform a refresh action
- EndLoop - Condition: @Status = "Completed"

Behavior: Repeats steps until @Status becomes Completed; exits immediately when true. @counter is initialized with 1 and incremented for iteration. Use the variable within or outside the loop.

Example 2: Loop with condition and max iterations

- StartLoop
- Poll service request status
- EndLoop Inputs: Condition = @SRState = "Closed"; Max_loop_iterations = 5

Behavior: Exits when SRState is Closed or after 5 iterations, whichever occurs first.

Example 3: StartLoop as first step in the test script

- StartLoop
- Launch App (for example, uses @user)
- Perform steps
- EndLoop Inputs: Condition = @Ready = true

Behavior: The section between StartLoop and EndLoop repeats; launch and steps execute each iteration as scripted.

Troubleshooting

Error	Fixes
Loop exits too early or never exits.	Verify the condition logic, ensure data is refreshed within the loop, and set a reasonable iteration cap.
Tests run longer than expected.	Tighten the exit condition, reduce the cap, or use the profile limit.
Runtime expression failures.	Fix syntax, initialize all variables used in the condition, and retest in unit/batch modes.

Note:

- No nested or overlapping loops.
- Loop scope is limited to one test script; EndLoop must follow StartLoop in the same test scriptscript.

Enabling Automation for Developer Web Client

Automation is available only for Developer Web Clients. Automation is not supported in Mobile Web clients. While installing, you must select the developer web client option only if automation is needed.

After installing the Developer Web Client, Tomcat will automatically be deployed in the following directory: `<webclient_install_location>\applicationcontainer_external`. The CONTAINERURL for AutomationSubSys section gets updated with the HTTP port provided during the Developer Web Client installation.

1. Enable the system preference by navigating to Administration - Application screen, then the System Preference.

Name	Value
Enable UPT	TRUE
Enable UPT Context	TRUE
UPT Max Record Cache	100

2. Make the following changes to the Developer Web Client's cfg files:
   - Update the [AutomationSubSys] section by setting the following:

     Set ContainerURL as follows: `CONTAINERURL = http://localhost:<Connector Port>/siebel/jbs`

     Set the `#Port` number according to the port specified in the following: `C:\Siebel\Client \applicationcontainer_external\conf\server.xml <Connector port="9001" protocol="HTTP/1.1".`
   - Update the [InfraUIFramework] section by setting EnableAutomation to TRUE:

     ```
     EnableAutomation=TRUE
     ```

   Playing the script through Developer Web Client is supported. Use the Siebel Thin Client URL to play back the script.

   - Multiple users using the same User ID is not supported, since it brings in ambiguity during conversion.
   - A case where recording is started and is not stopped, the session is logged out. Stop Recording is injected and considered for conversion.
   - Sticky sessions are not supported.
   - A functional flow is considered for conversion only with the confines of Start and Stop Recording.
   - Single users spanned across sessions are supported.

Approach for generating the Script involving switching of users on Portal and consumer applications

- Start the UPT-KWD recording of new user registration as an anonymous user. Proceed with the recording till a registration followed by login as a newly registered user and if there are any further scenarios for a newly registered user.

  > **Note:** At this point, since there is a user switch (due to new user login), an anonymous user UPT csv file will be generated in an `<anonymous_username>` folder under the `<SiebelServerBuild>\ses\siebesrvr\UPT` folder. Copy this anonymous UPT csv based on timestamp.

- Stop the UPT-KWD recording as a new user and log out of the application.

  > **Note:** Since the new user has logged out of the application, the new user UPT csv file will be generated in a `<NewUsername>` folder under the `<SiebelServerBuild>\ses\siebesrvr\UPT` folder.

- Now paste the anonymous UPT csv copied in Step 1 into the `SiebelServerBuild>\ses\siebesrvr\UPT \<NewUsername>` folder. The `<NewUsername>` folder will now have the following: *'anonymous user UPT csv'* and *'new user UPT csv file'*.
- Log in to the Siebel application as a new user and generate the KWD Script. This script will have both steps involving anonymous user followed by the new user registration or login and further steps as applicable.

Exporting the Test Scripts

You can export the generated scripts into an xml file.

To export the test scripts

1. Navigate to the Sitemap, Release screen, then the Test scripts view.
2. Query for the test script to be exported.
3. Click the Export button.

Importing the Test Scripts

You can follow the procedures in this topic to do the following:

- Activate the workflow for importing test scripts.
- Import the generated test scripts from the Script Pane to the database.
- Import the generated test scripts from the Release screen to the database.

Activating the Workflow for Importing Test Scripts

To activate the workflow for importing test scripts, make sure the Workflow Process named Testscript Import Workflow is active. See *Siebel Business Process Framework: Workflow Guide* to activate it.

Importing the test scripts from Script Pane to Database

To import the test scripts from the Script Pane to the database:

1. Click the Camcorder icon on the toolbar, then the script button.
2. Select the scripts to be imported from the Script pane.

   You can choose to select a single script or multiple scripts to import them into the database.
3. Click Import.

   A pop up message is displayed with the number of scripts that were imported successfully and the scripts that were not imported successfully.

   > **Note:** The import process will fail when you try to import a script with the same name if it is already available in the database. The log details of a failed import process will be available in the Siebel Logs folder.

Importing test scripts from Release Screen to Database

To import the test scripts from the Release screen to the database:

1. Navigate to the Sitemap, Release screen, then the Test Scripts view
2. Query for the test script to be exported.
3. Click Export.

   The test script is downloaded, in xml format, to the download folder.

   Use this xml file to import a test script into the database.
4. Navigate to the Sitemap, Release screen, then the Test Scripts view.
5. In the Test Scripts list applet, click Import.
6. In the File Upload popup window that appears, select the xml file to be imported.
7. Click Load.

The script will be imported into the database. If you have a script with the same name in the database, it will be replaced with the existing script. In this way the scripts can be exported and imported to different environments.

Post Import Options

You can perform the following tasks after importing scripts from the Script Pane:

- Navigate to the Sitemap, Release screen, then the Test Scripts view where you can add verification points to the existing scripts and use the play option to playback the edited script.
- Navigate to the Sitemap, Release screen, then the Test Scripts view where you can export the script as an xml file and import it to a different environment.
- Associate test scripts to Test sets.

11 Siebel Test Automation Execution

Siebel Test Automation Execution

This chapter includes information about the Siebel test automation execution process. It includes the following topics:

Setting Up the Jenkins Server

The Siebel Test Execution process is performed on the Jenkins server using the custom plugin designed for this process. The following procedure describes how to install and set up the Jenkins server.

To install and set up the Jenkins server

1. Navigate to Jenkins webpage.
2. In the LTS release section, download a Windows package, for Jenkins version that is compatible with DISA's Java version.
3. The Jenkins zip file will be downloaded.
4. Refer to Installing Jenkins, *https://www.jenkins.io/doc/book/installing/*.
5. Extract the Jenkins.msi file from the zip file.
6. Double click the .msi file to launch the Jenkins installer and follow the instructions to complete the installation.
7. Click Finish.

   Jenkins will be launched in your default browser.
8. After completing the installation, copy the Administrator password from the following path.

   ```
   c:\Users\intbuild\.jenkins\secrets\initialAdminPassword
   ```

9. Click Skip Plugin Installations.
10. Click Save and Finish.

    The Jenkins setup is ready.
11. Click Start to use Jenkins.

    The following welcome message appears: *"Welcome to Jenkins!" Your Jenkins server is up and running.*

Once the Jenkins setup is done, the Jenkins application will be launched by default on port 8080. However, if some other application is already running on this port, then you must change the port number in the jenkins.xml file.

For example, if the installed location of Jenkins is `C:\Program Files (x86)\Jenkins\`, then change the port number to 8090 (it could be any port number) in the xml file and restart the Jenkins server.

You can start or stop the Jenkins server using the Windows service running, for example, by the name `'Jenkins'`.

Setting Up and Configuring the Siebel Test Execution Plugin

The Jenkins custom plugin allows the user to interact with Siebel Master Server. The following procedure shows how to setup and configure the Siebel Test Execution plugin. The prerequisites for this procedure are:

- Before installing the Siebel Test Execution (STE) plugin, you must set up the proxy server configuration.

  The STE.hpi plugin is available in the following location: `SIEBEL_SERVER_ROOT/plugins`.

To set up the proxy server configuration

1. Navigate to Manage Jenkins, then Manage Plugins and select the Advanced tab.
2. Add the following HTTP proxy server configuration settings:
   - Server: `www.<your company proxy server>.com`.
   - Port Number: `<NN>`.

> **Note:** Check with your IT administrator for the proxy settings.

To install the custom plugin

1. From the Upload Plugin pane, navigate to the plugin location, select the STE plugin and then click Upload.

   The status of the STE plugin installation is displayed in the screen.
2. Click Installed to check if the STE plugin has been installed successfully.

Setting up the Jenkins Secondary Nodes

Before setting up the Jenkins secondary nodes, you must configure the Master or primary server.

To configure the Master or primary server

1. Navigate to Jenkins, click Manage Jenkins, and select Configure Global Security.
2. Set TCP Port for Agents to Random.

To append the secondary node machines

1. Navigate to Manage Jenkins, Manage Nodes, and select New Node from the Jenkins home page.
2. Enter the Node name `[machine_name.domainname.com]` and check the Permanent Agent option.
3. Click OK and then set the fields shown in the following table.

Field Name	Value
Name	`machine_name.domainname.com`
Description	N/A
Remote Root Directory	`C:\jenkins-slave`
Labels	Machine or pool name
Usage	It is recommended to utilize this node to the maximum extent.
Launch Method	Choose suitable option from dropdown and configure accordingly, or leave it default as *Launch agent by connecting it to the master*.
Availability	Keep this secondary node online.

4. Click Save.
5. Go to **Manage Jenkins**, and select Manage Nodes (in the Jenkins home page).
6. To disable, drill down on master node and click a button to make the node temporarily offline.

> **Note:** Since running the Test Automation on master node works in invisible mode, we need to disable master and create another node explicitly.

7. Once master node is made offline, create a node for the same machine as detailed in the section below for secondary node machines.
8. Navigate to **Manage Jenkins > Manage Nodes and Clouds**. Click on the secondary node created. If not already running and online, follow the options to launch the agent on the node. Typically, copying the `agent.jar` to the machine and running the command which looks like following:

```
"java -jar agent.jar -jnlpUrl http://abc.xyz.com:8080/computer
/<node name>/slave-agent.jnlp -secret a4e3897e8e7720f07495e05332525a62e4d896e32147083cfe2e2dd461ece350
-workDir "c:\jenkins-slave""
```

Configuring the Siebel Test Execution Job

You can configure the Siebel Test Execution job from the Configuration Screen.

To create a job to be executed on single machine, you must select the Freestyle project and create one job.

You can configure the Siebel Test Execution job from the Configuration Screen.

Select the following check box: "Restrict where this project can be run and provide the label expression similar to the one that was defined while configuring the secondary node machine, e.g. machine or pool name".

To configure the STE Job

1. Navigate to Build, then Add build Setup.

   > **Note:** The Siebel Test Execution option will be automatically displayed if the custom plugin provided by Siebel is already installed.

2. Select the Siebel Test Executor from the drop-down list.

   The Build Test Execution screen opens with the fields shown in the following table.

Field Name	Value
DISA Location	`c:\DISA`
Siebel Server URL	`http://xyzmlb.domainname.com/1660/Siebel/`
User Id	SADMIN
Password	`*******`
Parameters	`""` Note that you can configure the Parameters field as follows: o Set Parameters to `--reportLevel` to disable report generation and screenshot capture during the test script automation batch run. o Leave the parameters field blank (which is the default setting) to enable report generation and screenshot capture during the test script automation batch run. Other parameters that can be used: o `--parallel=<n>` Refer section *Configuring Multiple Batch Runs*. o `--runReference=<Node or node group Labels value in same case>` (will be matched with Run Reference field on Test Execution record).

Field Name	Value
	○ Example: `--runReference=Client_Pool_A` Refer section, *Creating the Test Execution Record* for Run Reference field, under *Configuring the Test Run* . For information on how to configure detailed test results and screenshot capture during unit mode/single test script playpack, see *Plugin Configurations*.

When you install the STE custom plugin, the command field value is populated by default (`java -jar %DISA_DIR %\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\KWDPresetup.jar`)

3. Click the help text shown against each field. Enter the details for all the fields.

Note: Enter `https://xyx.udomainname.com:16660/siebel` as the field value for the Siebel Server URL.

To run the STE Job

To run a STE job configured for a certain machine or pool of machines, drill down on the job name from Jenkins dashboard.

1. Click on **Build Now** to run the job.

   If the relevant node or nodes are online, job will be run on the available node.
2. From Build History panel, drill down on the latest running job and click Console Output, to view the job output log.

Executing the Automation Batch Run

This topic describes the steps involved in executing the automation batch run. You can proceed with this task after adding the test scripts to the test sets and creating a Master Suite with the test scripts.

To execute the batch run

1. Navigate to Sitemap, Release screen, then the Test Execution view.
2. Click New (or the plus (+) icon) on the Form applet.

   The Test Run # field value is auto populated and the default value for the Status field is Hold.
3. Select the Master Suite using the Master Suite Id field and select the Application Version.
4. Save the record.
5. Check if the values for the Server Credentials and User List are populated.
6. Select Windows as the Client Operating system (if this is a normal desktop run).
7. Enter the Client IP address if you have chosen the operating system as iOS.
   a. If the Operating System (OS) is iOS, enter the value for the Client IP Address, MAC Machine Username, MAC Machine Password and Mobile Port fields.

b. Before running the scripts on a Mobile platform, ensure that the plink.exe and psexec.exe files are available in the following location: `"..\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\exe".`
c. After running the scripts, the results will be displayed in the Firefox browser. The recommended version of Firefox browser is the latest version for which a compatible Web driver is available.

8. Update the Application Type, OS and URL in the Server Credentials applet.
9. Update the password in the User List applet.
10. Review the record.
11. Click Schedule Run.

    After you click the Scheduled Run button, the record status will change to Requested and the following files will be available in the Attachments applet:

    - batchconfig.xml
    - MasterSuite.csv
    - Resources.zip (if the Resources.zip is available in the Master Suite Attachments applet).

    Once the scheduled run is completed, the Siebel Test Execution records are updated with the appropriate status and the resulting XML files are added as attachments.

Note: You can use the Master Suite and test scripts available in the sample database for a batch run. For more information about the scripts available in the sample database and the actions performed by the scripts in the sample database, see *Sample Database Test Scripts*.

You can now proceed with the task of updating the other fields.

- Client Operating System: Windows (if it is a desktop run).
- Client Operating System: iOS (if it a mobile iOS run).
- Client Operating System: Android (if it is a mobile Android run).
- If the Master Suite contains Invokeperl and Serverconfig as keywords, then the Perl Path is mandatory (for example: `c:\\perl\\bil\\perl.exe`).
- If the Master Suite contains the Toolsconfig keyword, then update the following fields:

Field	Description	Sample Value
Siebel Tools Machine	Machine Name	slcxxxxxx.domain.com
Siebel Tools Path	Tools install folder path	`C:\Siebel\Tools`
Siebel Tools Machine Username	Login id for machine	`Domain/User1` or just user name
Siebel Tools Machine Password	Password for machine	`<xyzabc>`
Siebel Tools DSN	DSN name	SiebelDSN

Field	Description	Sample Value
Siebel Tools User	Tools user id	sadmin
Siebel Tools Password	Tools password	`<xyz>`

- If the Master Suite contains the `Inboundwebservicecall` keyword, then update the following fields:

Field	Description	Sample Value
EAI Machine Name	Machine Name	slcxxxxxx.domain.com
EAI Port Number	HTTPS Port Number	`16661`
EAI Server User	EAI user name	`user123`
EAI Server Password	Password for EAI user	`<abcxyz>`

- If the Master Suite contains the `ServerConfig` keyword, then you must update the following fields in the Server Credentials applet.

Field	Description	Sample Value
Server Home Path	Siebel Server install folder path.	`C:\Siebel\`
Siebel Server Machine	Siebel Server machine name.	slcxxxxxx.domain.com
Siebel Server User	Server machine user name.	`user123`
Siebel Server Password	Password for Server machine user.	`<abcxyz>`
Siebel Gateway Machine	Siebel Gateway machine name.	slcxxxxxx.domain.com
TLS Port	TLS port number.	`443`
AI Server Port	HTTPS port number	`16661`

Note: If you receive the error message, HTTP Status 405: Method not allowed, this could be due to a required Business Service not having the required access permission in Siebel. To resolve this issue, navigate to the Administration Application screen, then the Business Service Access view, create a record selecting Automation Rest Service, provide the access permissions, and then restart the Siebel Server.

Parameters Applet:

- Click New to add a record. Provide a Parameter Name and corresponding Value. Ensure to use exact name and case in your Test Scripts.
- Once **Schedule Run** button is clicked, parameters are added to `batchconfig.xml`, and New, Delete buttons are disabled.
- At the start of the batch run, before starting Test Script execution, Keyword framework, automatically creates variables for each of the parameter with same Name and initializes each with respective Value. These variables can be directly used in Test Scripts and/or Data Set Condition expressions.
- Example: For a parameter with `Name = ENV`, `Value = UAT` in Parameters applet, a variable `@ENV` is created with value `UAT`
- Example: For a parameter with `Name = SSOUser`, `Value=UserXYZ`, a variable `@SSOUser` is created with value `UserXYZ`.
- Adding Parameters is also supported in REST API.

Add and Delete buttons in Server Credentials Applet:

- If there are no records in Server Credentials (which means that Launch keyword has not been used), Keyword Framework will create a Chrome webdriver instance and the handle is made available for CustomExtension keyword. If a browser other than chrome is required, ensure to have a record in Server Credentials Applet reflecting the choice of browser. In either of the case, ensure appropriate browsers and corresponding drivers are available on the client machine.
- You may add a record for EAI Server details for alias name used in InvokeREST keyword.
- Ensure not to delete the records that are automatically added by application upon saving the Test Execution record.

Copying a Test Execution record

- In the Test Execution View, select a Test Execution record. Click on Applet Menu → Copy Record (CTRL+B) menu item.
  This action copies the selected record, along with various values from the original record.
- Following are the field values copied and new values set:
  Test Execution Form Applet:
  The fields Test Run #, Status, schedule, Test Harness Machine, and Runs Completed, are set to their default values.
  Rest of the field values are copied as-is over to new record.
  Server Credentials List Applet:
  All values are copied for latest Application Alias entries found on Test Scripts.
  Users List Applet:

This applet is repopulated with unique user ids from Test Scripts.

Parameters List Applet:

Values from this applet are not copied, as these are not stored in database tables. The entries need to be created manually.

Attachments List Applet:

No attachments are copied.

Test Execution Profile

Test Execution Profile View enables creation and management of profiles used with Test Execution records. A profile stores frequently used inputs and parameters required for Test Execution.

It avoids repeated manual inputs for each Test Execution record.

Go to site map, search for Test Execution Profile, and click the View link.

Fields

- **Name:** Profile name.
- **Description:** Profile description.
- **Default Profile Checkbox:** Marks selected profile as default one to be picked on new Test Execution records.
- Only one default profile allowed.
- Multiple unique profiles supported.
- Setting a new one as default unchecks the previous one.

System and Custom Parameters

- System parameters are editable; Name and Description are read-only.
- Click + to add Name and Value pairs in Custom Parameters applet.

Parameter	Default Value	LOV Values	Notes
Web Server OS	Windows	Windows, Unix	Used in Test Passes
Web Server App	Tomcat Apache	Tomcat Apache, IBM HTTP Server	Used in Test Passes
Type	Ad Hoc		Used in Test Passes
Server OS	Windows	Windows, Unix	Used in Test Passes
Database	MSSQL	ORACLE, MSSQL, DB2	Used in Test Passes
Client OS	Windows	Windows, Android, iOS	Used in Test Passes
Browser	Chrome	Chrome, Edge, Firefox, Safari, Siebel Mobile APP	Used in Test Passes
Default Application URL	N/A	N/A	Updated in Server Credentials Applet

Parameter	Default Value	LOV Values	Notes
Default REST URI	N/A	N/A	Updated in Server Credentials Applet
Import Test Step Results	Y	Y, N	Used for importing Test Step Results
Parallel	1	1-5	Information only
Report Level	N/A	0-3	Information only
iOS Version	N/A		For Mobile Runs
iOS Device	N/A		For Mobile Runs
iOS Device Orientation	Landscape	Landscape, Portrait	For Mobile Runs
Siebel App Path	N/A	N/A	For Mobile Runs Ex: `/Users//Siebel.app`

Note: If no Test Execution Profiles exist, or if no Test Execution Profile is used for creating Test Execution records, the default values indicated in the table are used automatically.

Test Execution without Jenkins

You may run a Master Suite without using Jenkins, directly via command line option. This option is useful in case you want to control execution in a CI/CD pipeline, or run a batch on a specific client, such as a VM or a laptop.

On a DISA machine, open Command prompt to execute in Administrator mode. Run following command, after ensuring folder paths are correct.

```
java -jar <DISA_HOME>\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\
Framework\KWDPresetup.jar /d <DISA_HOME> /j <User id> /g <password> /r
<Siebel Web Server URI> /c <EAI Component> /n <DISA machine name> /p "--parallel=<nn> --reportLevel=<0 or 1
 or 2 or 3> --runReference=<Run Reference used in Test Execution>"
```

- `/d <DISA_HOME>`- Location of DISA installation directory. Ex: C:/DISA
- `/j <User id>`- Siebel User ID for REST/SOAP operation.
- `/g <password>`- Password to authenticate the UserId on the Siebel Server for REST/SOAP operation
- `/r <Siebel Web Server URI>`- Refers to the EAI enabled Siebel Server URI for REST/SOAP operation. Please provide the URI path till /<Application Context Name>

  Example: `https://servername.com:port number/<Application Context Name>>`
- `/n <DISA machine name>` - HostName of the machine where DISA is installed
- `/c <EAI Component>`- (Optional) Ability to provide Custom EAI component.

  Example: `default-> /app/eai/enu/, Custom -> /app/my_eai/deu` here **my_eai** is user defined EAI Component
- `/p`-

- --parallel=<nn> - Optional - Ability to run multiple master suites in parallel on a given Client (DISA) on same or different browsers.

  Example: “parallel=1” One test execution record will be executed on the DISA machine.

  “parallel=3” Three different test execution records will be executed on the same DISA machine.

  By default (if --parallel=nn is not specified), one test execution record will be queried.
- --reportLevel=<0 or 1 or 2 or 3> - Optional - Based on the reportLevel, screenshot will be displayed.
  - “reportLevel=0” Detailed report is completely turned off that is individual report won’t be available
  - “reportLevel=1” Detailed report available with Screenshots turned off that is, Not capturing any screenshots for failed or passed steps
  - “reportLevel=2” Detailed report available with selected Screenshots i.e. screenshots will be taken only if the screenshot column in the test step is set to Y
  - “reportLevel=3” Detailed report available and reports with only Application Screenshots. If unspecified, default output is detailed reports with screenshots
- --runReference=<label_name>- Optional – picks a Test Execution record in Scheduled Status, where Run Reference field value matches with the label name.

  Example: runReferecne=UAT Running the above command will start the batch run on the machine. Once run is completed, result reports are zipped and uploaded to the Test Execution record and mark its status as 'Completed'.

Examples

Example 1:

```
java -jar C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\
Framework\KWDPresetup.jar /d C:\DISA /j SADMIN /g ldap /r https://siebelserver.xy.abcd.com:16690/siebel/ /
c /app/eai/enu /n
machinename.xy.abcd.com /p parameters are optional - /p and parameters can be omitted altogether. Default:
 parallel=1 runReference= and reportLevel=
```

Example 2:

```
java -jar C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\
Framework\KWDPresetup.jar /d C:\DISA /j SADMIN /g ldap /r https://siebelserver.xy.abcd.com:16690/siebel/ /
c /app/eai/enu /n
machinename.xy.abcd.com /p
"--parallel=1 runReference=abcdxyz"
```

Note: The double quotes around /p parameters which are separated by spaces.

Automated Rerun of Test Scripts

After a batch run has finished and it is found that some of the test scripts failed due to test environment issues, then it may be useful to rerun only the failed scripts, the predecessor scripts they depend on (Skip-on-prior-abort in Test Sets), and the non-executed scripts.

After such a rerun, all the test results from prior runs are consolidated with the rerun results. You use the Rerun button in the Test Execution view to perform reruns.

To rerun test scripts

1. From the Site Map, navigate to the Release screen, then the Test Execution view.
2. Select a Test Execution record, with a Status of Completed, that you want rerun.

   Note that when a batch run is completed, the Test Execution record is updated as follows:

   - A Reports zip and XML file are automatically created and uploaded to the Attachments applet of the Test Execution record.
   - The Status of the Test Execution record automatically changes to Completed.
   - The Rerun button in the UI is enabled.
3. Click Rerun in the Test Execution view.

   - The Status of the selected Test Execution record changes to Requested.
   - When a Siebel Test Execution job is invoked on the Jenkins server, the Test Execution record with Status of Requested is picked up and run on an available DISA client.
4. After the rerun completes:

   - The rerun results are consolidated with that of prior runs, and the overall consolidated test results are uploaded to the Test Execution record.
   - The Status of the Test Execution record changes to Completed.
   - The Runs Completed field in the Test Execution record increments by 1. For example, the value in the Runs Completed field would be 2 after the first rerun completes.
   - An XML file (with a <Runs Completed> suffix) containing a summary of the results is attached to the Attachments applet of the Test Execution record. For example:

     ```
     <Test Run ID>_<Mastersuite Name>_MasterSuite_2<Runs completed>.xml
     ```

     You must select this attachment record to Create Test Passes - for more information, see *Creating Test Results*.

Creating Test Results

After the batch run finishes for a Master suite, the results will be updated as an Attachment for each Test Run Id in the Test Execution view.

From 25.8 onwards, the Test Passes are created automatically, if the Test Sets are mapped to valid Test Plan records.

The Attachment name will begin with the Test Run number (#) mentioned in the parent list applet. Once this file is available, you can select the attachment record and click Create Test Passes to create the test results in Siebel.

The prerequisites for Create Test Passes to be successful are as follows:

- The LOV value `completed` must be available for the Status field in the Test Pass view.

- Test Sets must be associated with Test Plans.

  Test Plans must be associated with Test Sets before scheduling a Test Execution (creating a test pass).

  To create a Test Plan and associate it with a Test Set, at least one Project record (which contains the Release field) must exist.

If these conditions are not present, then Create Test Passes will not be successful (and will fail). Also, note the following:

- If reruns are done, an XML file will be available in the Attachments applet for each rerun with a suffix indicating the number of runs. For example: `88-1WCI9B_MS1_MasterSuite_2.xml`.
- The naming convention for XML files is:

  ```
  <Test Run ID>_<Mastersuite Name>_MasterSuite_<nth Runs>.xml
  ```

- Select the appropriate XML file before clicking Create Test Passes.

The following procedure shows how to Create Test Passes. You must associate a Test Plan with a Test Set before scheduling a Test Execution (creating a test pass).

To Create Test Passes Manually

From 25.8 onwards, the Test Passes are created automatically along with uploading of Reports zip file to Attachments. In case you need to re-create Test Passes, perform following steps. Create Test Passes button is typically disabled, unless "Import Test Step Results" parameter was set to "N".

1. From the Site Map, navigate to the Release screen, then the Test Execution view.
2. Select a Test Execution record, with a Status of Completed.
3. Go to the Attachments applet and select the appropriate XML file for which you want to create test passes.
4. Click the Create Test Passes button to create the test results in Siebel.

   After the test results are successfully created, the following message appears:

   ```
   Test Pass Created Successfully. You can query in the Test Pass view starting with <Test Run #>.
   ```

   Test Pass records are created with the prefix Test Run #.
5. In the Test Pass view, query for the Test Run # in the Test Pass # field, then drill down on each Test Pass record to view the test results for the respective test run.

> **Note:** The earlier Test Passes and Test Results, if present, are deleted and new ones are created.

Viewing Test Results

Click on **Failed Test Results** button on **Test Execution Form** applet. You will be navigated to **Test Results View**, listing Test Result field with values 'Fail'. You may categorize the fail in "Issue" field and add "Comments".

The **Failed Test Results** button is enabled when the Status is set to "Completed".

Clicking the **Failed Test Results** button will display:

1. The Automation Test Result View with failed test scripts if available.

2. An alert pop-up with the message "No Test Passes found for the Test Run#" if test passes are not found.
3. An alert pop-up with the message "No Failed Test Results" if no test scripts have failed.

By using the **Failed Test Results** button, you can quickly identify and review failed test scripts for a specific run ID.

> **Note:** Result Log field will be empty if Report Level was set to 0. Individual reports are not generated after run is completed, and consequently, the Result Log field will remain blank.

Test Step Results: The results of individual Test Steps are automatically uploaded after the batch run is completed.

Results zip file: A **Results zip file** button is available, which allows users to download a zip file containing the complete test reports for a specific Run Id.

The downloaded zip file contains the following:

- Test reports for the specified Run Id
- Logs and screenshots

After the Test pass is created successfully, you can view the test results. The following procedure shows how to view the test results.

To view the test results

1. Navigate to the Release screen, then the Test Pass view.
2. Query for a Test Run # in the Test Pass # field to get the results for that particular test run.

   The results that appear show the number of test cases that passed, failed, were aborted, or not executed.

   > **Note:** For data driven test script results, the Iteration number indicates the corresponding Sequence number of the Data Set row. The default value is 1 for all other test script results.

Test Results View

The Test Result List Applet has three fields aimed at improving test result management and analysis: **Result Log**, **Issue**, and **Comments**.

1. **Result Log**
   a. Stores the result log data of the first failed test step.
   b. This will hold the data of Test Step Result's Result Log.
   c. For Report Level = 0, individual reports are not generated; hence, the field remains blank.
2. **Issue**
   a. An unbounded picklist to categorize issues encountered during testing.
   b. **Available Options:**
      - Test Script Issue
      - Test Environment Issue
      - Functional Regression
      - Application Issue
3. **Comments**
   a. Add any text comments

Configuring Multiple Batch Runs

Siebel test automation supports parallel execution where you can execute one batch run at the same time as multiple other batch runs on the same (DISA) client. Each batch run runs as a parallel but isolated browser session on the same client, without impacting other sessions running in parallel. The parallel execution of multiple automation batch runs maximizes the utilization of compute resources and minimizes the overall cycle time.

To run multiple batch runs in parallel on a given client on the same or different browsers, complete the steps in the following procedure.

To configure multiple batches to run in parallel

1. Execute and schedule your batch runs as required – for more information, see *Executing the Automation Batch Run*. Make sure that the latest STE.hpi is installed on Jenkins and verify that it has a "Parameters" field.
2. Configure the `--parallel=nn` parameter on the Jenkins Siebel Test Execution, where -- is a *double hyphen* and `nn` is the maximum number of batches to run on the client. This parameter is provided by the Siebel Test Execution plugin on Jenkins, in the "Parameters" field.
   - For example, `--parallel=nn` will initiate a query for test execution records, limiting the number to 3.
   - If 4 or more test execution records are available, only 3 batch runs will be initiated in parallel.
   - By default (if `--parallel=nn` is not specified), one test execution record will be queried.
3. After the `--parallel=nn` parameter is configured, the following describes how multiple parallel batch runs work on a virtual machine:

   a. When the keyword framework is initiated on a virtual machine, it starts the number of threads as specified by the `--parallel=nn` parameter.

      - For 8 GB RAM, 2 to 3 parallel suites are recommended.
      - For 16 GB RAM, 3 to 5 parallel suites are recommended.

   b. Each thread is assigned a test execution record, identified by Run Id (the Row Id of the test execution record), and a browser session starts that is independent of any other browser session.

   c. Each test execution record or Master Suite runs from start to finish according to the sequence of specified test scripts, and results are collated into a report that is stored in a folder named by the "Run Id" of the test execution record under SiebelTestAutomation\TestExecutions.

   d. The following subfolders are created under the Run Id folder, along with any relevant files:

      - Reports
      - Resources
      - Scripts

   e. After a master suite run completes, the reports are zipped up and uploaded to the Siebel Test Execution record as an attachment.

   f. New runs cannot be initiated until all the threads in a parallel batch run are complete.

Test Scripts Dashboard

Test Scripts Dashboard provides summary of Test Scripts data. You can filter the data on Team, Release, or Status fields.

To navigate to the Test Scripts dashboard:

- Go to site map, search for Dashboard, select Test Scripts Dashboard.
- You can also navigate to it by searching for Release and clicking on the screen link.

Following infolets display data on dashboard:

- **Test Scripts**: Total number of Test Scripts based on filters applied.
- **Test Scripts-Automated**: Total number of Test Scripts where 'Auto' flag is true, based on filters applied.
- **Data Sets**: Total number of Data Sets based on filters applied.
- **Data Sets-Data Rows**: Total number of rows of data in Data Sets, indicating number of iterations, based on filters applied.
- **Master Suites**: Total number of Master Suites based on filters applied.
- **Master Suites-Test Scr**: Total number of Test Scripts used in Master Suites based on filters applied.

Note:

- Chart shows number of Test Scripts by Release or by Team Name, based on filters applied.
- Quick Links provide hyperlinks to different Views. It also provides link to Test Execution Dashboard View.

Test Execution Dashboard

Test Execution Dashboard provides summary of Test runs and Test Results data. You can filter the data on Build, Run Id fields.

1. Build is **Application Version** value used on Test Execution records, and Run Id is `'Run #'` field value or row id of a Test Execution record.
2. Go to **Sitemap**, search for **Dashboard**, select **Test Execution Dashboard**.

   You can also navigate to it by clicking on link on Test Scripts Dashboard.

Following infolets display data on dashboard:

- **Test Runs-Completed**: Total number of Test Executions in Completed Status, based on filters applied
- **Test Runs-Last 24 hours**: Total number of Test Executions Completed in the last 24 hours, based on filters applied
- **Test Runs-Last Week**: Total number of Test Executions Completed in the last 7 days, based on filters applied
- **Test Runs-Last Month**: Total number of Test Executions Completed in the last 30 days, based on filters applied
- **Test Results-Total**: Total number of Test Result records, based on filters applied
- **Test Results-Passed**: Total number of Test Result records with Test Result field value as 'Pass', based on filters applied

- **Test Results-Failed**: Total number of Test Result records with Test Result field value as 'Fail', based on filters applied
- **Test Results-Not Exec**: Total number of Test Result records with Test Result field value as 'Not Started' or 'Aborted', based on filters applied
- **Pending Issue Type**: Total number of Test Result records with Test Result field value as 'Fail' and 'Issue' field value being empty, based on filters applied. This indicates pending classification of the Fail result

Note:

- Pie Chart shows break up of Test Result records with Test Result field value as 'Fail' and 'Issue' field not empty, based on dashboard filters applied.
- Quick Links provide hyperlinks to different Views. It also provides link to Test Scripts Dashboard View.

12 Setting Up Keyword Automation Testing on iOS

Setting Up Keyword Automation Testing on iOS

This chapter shows you how to set up iOS mobile devices for keyword automation testing. It includes the following topics:

- *About Running Keyword Automation Testing*
- *Installing Xcode on the Xcode iOS Simulator*
- *Installing Appium*

About Running Keyword Automation Testing

To run keyword automation testing on iOS Xcode simulator, the following software is required and must be installed:

Xcode. For more information, see *Installing Xcode on the Xcode iOS Simulator* .

The Siebel Mobile requires the following software to be installed and setup accordingly for keyword automation testing:

1.
   - Xcode. For more information, see Installing Xcode on the Xcode iOS Simulator .
   - Appium
   - Siebel.app. Refer to Bookshelf ; Mobile Guide: Connected for more information.

The Siebel.app must be placed in the following path on the MAC desktop: `/Users/<MAC_USERNAME>/Desktop/Siebel.app`.

Installing XCode on the XCode iOS Simulator

Download Xcode

- Open Safari on your iOS device and navigate to *http://developer.apple.com/xcode/downloads*.
- Sign in with your Apple ID.

Install Xcode

- Locate Xcode version 16.2 and download the .xip file.
- Double-click the .xip file to initiate the installation process.
- Move the Xcode application to the Applications folder on your iOS device.

Launch Xcode and Accept the License Agreement

- Open Xcode from the Finder/Applications folder.
- On the Xcode and iOS SDK License Agreement screen, click Agree.
- Xcode and its required components and tools will be downloaded.

Add the iOS Simulator

- Perform these steps only when required iOS version is not available.
- Navigate to **Xcode** > **Settings** > **Platforms**.
- Click the + button and select iOS 18.3 Simulator.
- Click Get to download the iOS 18.3 Simulator.

Configure the Simulator and Install Certificates

- Install the required certificates.
- Install the React Native app on the chosen simulator.

Installing Appium

Prerequisites

- MAC operating system **Sequoia 15.5**
- Xcode **16.2**
- Homebrew to install node

Installation Steps

Download Appium

1. Run: npm install -g appium@1.22.3
2. Run: npm install wd
3. Rename package-lock.json under /Users/<username> to package.json and then re-execute the command.
4. Execute the below command in the terminal to point the required Xcode for execution.

```
sudo xcode-select -switch /Applications/Xcode.app
```

Following are the steps for workaround to support automation on iOS 16.0 onwards:

1. cd `#appiumNodeDirectory#` Eg: `/usr/local/lib/node-modules/appium.`
2. Open `pacakge.json` inside appium directory.
3. Search for "appium-xcuitest-driver".
4. Replace with "appium-xcuitest-driver": "4.35.0" (updated to 4.35.0).
5. Save file and Quit `webdriveragent xcode` project if it is open.
6. npm install (make sure you are at appium folder)

7. npm install -g appium-webdriveragent
8. Now replace the folder in **/usr/local/lib/node_modules/appium/node_modules/appium-webDriverAgent/** with the newly generated `appium-WebDriverAgent` and open `WebDriverAgent.xcodeproj` in Xcode project.
9. Open `webdriverAgentRunner` in Xcode, compile and deploy.

> **Note:** Click on run button with option Test to deploy.

13 Data Driven Testing

Data Driven Testing

This chapter describes data driven testing. It includes the following topics:

Overview of Data Driven Testing

Data driven testing is a test design and execution strategy. In the context of Siebel Test Automation, it is the ability to separate the data from test scripts and to manage it using data sets. It also extends ability to iterate the same test script through variations in data.

A data set in Siebel represents a two dimensional table with fields as columns and values for these fields as rows. You can import the data set into Siebel from flat files (tab separated text files)

Creating a Data Set

You can create any number of data sets and use them across test scripts. A data set can be created from the User Interface (UI), and it is recommended to import the data set from a flat file (tab separated text file).

To create a data set, do the following:

1. Navigate to Release, then the Data Set View
2. Click the plus (+) icon from the Data Set Definition view to create a new record.

   You can also add a new record using the Add Fields button.
3. Enter the value for the Name Field.

4. Click the plus (+) icon in the Values Applet and enter the values.
5. Save the record.

> **Note:** Copying a data set performs a deep copy of all fields and values as well.

Importing a Data Set

You can create or update a data set by importing flat files (tab separated text file). When imported, a new data set is created, the name of which is its file name. If the name already exists, you are prompted to overwrite with a new file name or not.

Prepare the data in spreadsheet and save it in a tab separated format. Ensure that the first column name is Sequence, and that it has the sequence numbers in the order you want.

To import a data set, do the following:

1. Navigate to Release, then the Data Set View A Data Set is created with same name as the file name and data is imported as Fields and Values into it.

   If the name already exists, you are prompted to overwrite with a new file name or not.
2. Click the Import button and select the file to import.

   A data set is created, the name of which is its file name. If the name already exists, you are prompted to overwrite with a new file name or not.

Exporting a Data Set

You can export a data set to a flat file.

To export a data set, do the following:

1. Navigate to Release, and then to the Data Set view.
2. Select the data set you want to export, and click Export.
3. Click Save.

> **Note:** If required, you can update the values in the file and export the data set again.

Associating Test Scripts with a Data Set

You can associate a test script with only one data set.

To associate a test script with a data set, do the following:

1. Navigate to Release, and then to the Test Scripts view.
2. Select the required test script record, and then select the required data set.
3. Save the record.

Associating a Data Set with a Test Script

A data set can be associated with any number of test scripts.

To associate a data set with a test script, do the following:

1. Navigate to Release, then the Test Scripts view.
2. Select the required test script record, and then select the required data set.
3. Save the record.

Referencing Data Set Fields in Test Scripts

Apart from associating test scripts and data sets, data set fields also need to be referenced in test scripts. When you create a data set, you can name a data set using the Add Fields button as follows:

```
Applet name|Field name
```

When you do this, instead of using the input value, you can reference the data set as follows in the test script:

```
#
```

You need to ensure that the target object matches the Applet name|Field name naming format. It's recommended to use a meaningful short name when naming a Data Set column in the input, such as: `#<name of column>`. For example, where Age is the name of the data field in a data set, you name it `#Age`.

To reference a data set field in a test script

1. Navigate to the Release screen, then the Test Scripts view.
2. Select a record, and drill down on the record name.
3. Click a Test Step record.
4. Use the following examples to edit the Test Step record:
   - In Launch Test Step, if the associated data set has a User Name field with the value Employee, update this field to `#Employee`.
   - In Input Value Test Step, if the associated data set has a Value/Variable field with the value Revenue, update this field to `#Revenue`.

     You can apply the same changes to the ItemRN and AppletRN fields.

Recommendations to consider when you reference a data set in a test script include the following:

- The application performs no validation on referenced or unreferenced fields.
- You can append the dollar sign ($) to your referenced field name to include a time stamp at run time. For example, the field `#Name$` appends a time stamp to values from the Name column in data sets.

  However, if a data set value already has a dollar sign appended to it, no timestamp is provided; and if the data set is balnk, then a field such as `#Name$` results in just the time stamp value.
- `#@<variable name>` is supported to dynamically reference Data Set columns, which resolves to column name matching the value of the variable. For example, `#@Var @Var` can contain Data Set field name.

- Data set field names cannot begin with `'@'`.
- From the data set, you can reference the User Id in Launch Action. For example, `#UserId` where User Id is a field name in the data set.

Iterations Types Available with Data Sets and Test Scripts

Test scripts can be either iterated over data set rows individually, or as a sequence of test scripts or test scenarios for each row of the Data Set. Use the following procedure to view the avalable iteration types.

To view the available iteration types

1. Navigate to the Release screen, then the Test Sets View.
2. Drill down on a Test Set record.

   The three iteration options are as follows:

   - **No iteration.** If any iteration was selected previously, select the required records and click Remove Iteration. Only the first row from the data set is used for test scripts associated with the data set, and if the fields are referenced in the test scripts.
   - **Individual Iteration.** Select the required records and click Add Iteration. The field name Scenario Type reflects the individual iteration for the selected test scripts. The selected test script is run for all rows of the data set, followed by the next test script in the test set.
   - **Scenario Iteration.** Click Scenario Iteration, and then click OK when the confirmation prompt appears. The field name Scenario Type reflects the scenario iteration for all test scripts in the test set. A full test script sequence is executed for the first rows of the respective associated data sets, before picking up the next row of data sets. This applies to all test scripts in the test set.

Dynamic Data Selection from Data Set

In order to selectively use a Data Set row at run time, use **Condition** field in your Data Set for the respective Data Set row. If the expression evaluates to true, the data in the row is used for the iteration, or else the iteration is skipped.

Use Condition expression to use or skip a Data Set row at run time. Default is to use all rows.

1. Add a field **Condition** to your Data Set (either in UI or tab separated text file to import).
2. Define the conditions that determine if a particular row is relevant or not.
3. At run time the actual values of variables are replaced in the expression, and is evaluated for true or false.

Note: Syntax of expression is important and no prior validation of expression is performed.

Note: By default: If Condition field is not present, then the data set row is not skipped. Also, if Condition field is present, and the value is blank, then the data set row is not skipped.

Rules for specifying Condition:

1. Ensure that the variable name does not have spaces. For example, variable `@Test Variable` is invalid, but `@TestVariable` or `@testvariable` or `@test_variable` are valid.
2. Single space is the delimiter between the operands and operators used in condition expression. For example: `@test > 10`, `@name = 'abc' OR @name = 'xyz'`.
3. Following values in Condition will evaluate to TRUE: `true or TRUE or no value` (blank), whereas `false or FALSE or 0`(zero) evaluate to FALSE.
4. Following operators are supported: `>,<,=,<>,>=,<=, AND, OR`.
5. Ensure the variables used in Condition are initialized with appropriate values, else the script fails with an error.

Examples for Condition:

- `@env = 'DEV'`
- `@env = 'UAT'`
- `@conf > 0 AND @conf < 50`

> **Note:** To initialize variables before a Test Script starts to run, use Parameters applet on Test Execution. Add Name and Value pairs. For example: To initialize a variable `@balance` with value `100`, add a Parameter with name as balance and value of 100. Framework creates variables for each of the Name in Parameters applet, and initializes with corresponding Value. Such parameters or variables can be directly used in Test Script and/or in Data Set Condition expression for comparison or input value.

Associating a Data Set with a Test Set

A Data Set can be associated with a Test Set also, which is a group of Test Scripts.

To associate a Data Set with a Test Set, do the following:

1. Navigate to the **Test Sets View**.
2. Select a Test Set record, and then select a required Data Set to associate to.
3. Save the record. A prompt appears with following message:

   Selected Data Set will override the individual Test Script associations if any.

   Click OK to override Test Script associations of Data Set. Ensure that the overriding Data Set has referenced fields and appropriate values for Test Scripts.

The Data Set association can be changed any time, by either replacing with another, or by clearing the field. Once Data Set association at Test Set is removed, the Data Sets associated to Test Scripts will take effect, if any.

Copying a Test Set

There are two types of copy feasible for a Test Set. One is to copy the header information only to new Test Set.

Second is to copy the Test Script associations and iteration type as well.

To copy a test set

1. Select Copy from the **Test Set** List Applet menu.
2. A pop up is displayed with *Click OK to copy all Test Script associations*, *Click Cancel to copy Test Set record only*.
3. Click OK or Cancel.

Viewing Test Sets associated to a Data Set

To view test sets associated to a data set, follow the steps:

1. Navigate to **Data Set** List View and click on the record.
2. In the next level, select or click on **Test Sets**.
3. This applet displays the Test Sets to which the Data Set is associated directly.

14 Setting Up Android Mobile Devices for Automation Testing

Setting Up Android Mobile Devices for Automation Testing

This chapter shows you how to set up Android mobile devices for keyword automation testing. It includes the following topics:

About Setting Up Android Mobile Devices for Keyword Automation Testing

To run keyword automation testing on android mobile devices, the system requirements on Microsoft Windows are:

- Microsoft® Windows® 10, 8, 7, 10 or Vista (32 or 64-bit)
- 2 GB of RAM minimum, 4 GB of RAM recommended
- At least 1 GB of RAM for Android SDK, emulator system images, and caches
- 1280 x 800 minimum screen resolution
- Java Development Kit (JDK) 8
- (Optional) For accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality.

The following software is required to run keyword automation testing on a Microsoft Windows platform:

- Android Software Development Kit. For more information, see Installing Android Software Development Kit on Microsoft Windows 7 Machine
- Appium. For more information, see Installing Appium on Microsoft Windows.

The following is required to run keyword automation testing on the Mobile Application Framework (MAF):

SiebelMobile.apk

The tasks involved in setting up Android mobile devices for keyword automation testing are:

- *Installing Android Software Development Kit on Microsoft Windows 7/10 Machine*
- *Installing Appium on Microsoft Windows*
- *Setting the ANDROID HOME Variable*
- *Setting the Path Variables*
- *Verifying Android Installation and Configuration*

Installing Android Software Development Kit on Microsoft Windows 7/10 Machine

You can use the Android SDK (Software Development Kit) to create applications using the Android platform. The installer checks your machine to see if required tools like the Java SE Development Kit (JDK) are available and installs it if required. The installer saves the Android SDK Tools in a specified location outside the Android Studio directories.

Before installing the Software Development Kit (SDK) on a Windows 7 /10 machine, Java JDK 8 must be installed as a prerequisite.

To install the Android SDK

1. Double click installer_r24.3.4-windows.exe to install the SDK.

   > **Note:** Write down the name and location of the SDK saved on your system. You may have to refer to the SDK directory later if using SDK tools from the command line.

   After the installation is complete, the Android SDK Manager starts.
2. Click Tools and then select Options in the Android SDK Manager.

   The Android SDK Manager – Settings screen appears.
3. On the Android SDK Manager – Settings screen, enter the HTTP Proxy Server and HTTP Proxy Port details as required to bypass any firewall.
4. Click Packages and then select the following packages to install in the Android SDK Manager:
   a. Install Android 5.0.1 (API 21)
   b. Install Android 6.0 (API 23)
   c. Install Extras.
5. Click Install Packages.

To accept the license agreement for each package, double-click each package name, and then click Install. A loading progress message appears on Android SDK Manager window. Do not exit the Android SDK Manager until loading has finished, otherwise the loading process will be cancelled.

Installing Appium on Microsoft Windows

Before installing Appium on Microsoft Windows, make sure that the .NET Framework 4.5 redistributable libraries are available.

To install Appium on Windows

1. Download the Appium for Windows zip archive file and extract the files into a folder (for example, Appium) on your C: Drive.
2. Double click appium-installer.exe.
3. Install the Appium tool on your C: Drive.
4. Set ANDROID_HOME as your Android SDK path and add the tools and platform-tools folders to your PATH variable.

Setting the ANDROID HOME Variable

You must set the ANDROID_HOME and the path environment variables after installing the different packages.

To set the ANDROID HOME variable

1. Open the Environment Variable.
    a. Right-click My Computer, select Properties, and then select Advanced system settings.
    b. Go to the Advanced tab, and click Environment Variables.
    c. Click New under User Variables for <USERNAME>, and on the New User Variable dialog box that appears:
        i. Enter the following Variable name:ANDROID_HOME
        ii. Enter the following Variable value: C:\SDK (SDK folder path)

        Note: The SDK folder path might vary depending on the SDK folder location

    d. Click OK to close the New User Variable dialog box.
2. Click OK to close the Environment Variables window.

Setting the Path Variables

The following procedure shows you how to set the path variables for Android SDK. You must set the path variables to run the scripts on the Android device and emulator.

To set the path variables for Android SDK

1. Navigate to and open the SDK folder (for example, C:\SDK).

The tools and platform-tools folders are located in the SDK folder.

2. Make a note of the path to both these folders, as follows:

```
C:\SDK\tools

C:\SDK\platform-tools\
```

3. Open Environment Variables.
   a. Right-click My Computer, select properties, and then select Advanced system settings.
   b. Go to the Advanced tab, and click Environment Variables.
4. Under System Variables, select the Path variable, and then click Edit.
5. On the Edit System Variables dialog box that appears, edit the value for the system variable as required.
   - For example, append the full path to the \tools folder to the end of the line as follows: C:\SDK\tools.
   - For example, append the full path to the \platform-tools folder to the end of the line as follows: C:\SDK \platform-tools.
6. Click OK to close all dialog boxes.

Verifying Android Installation and Configuration

The following procedure shows you how to verify if the Android is installed and configured correctly.

To verify Android installation and configuration

- At the Command Prompt, enter the following command and then press return:

  ```
  Android
  ```

  The SDK manager starts.

Testing Automation on a Android Device

To check the automation testing on a real device, you must connect the device (For example, Samsung Galaxy Tab) to the Windows 7 /10 machine with a USB cable.

Before running automation testing on a real device, make sure that WiFi and VPN connections are up and running.

To run automation testing on a real device

1. Start your Android device.
2. Tap Settings, and then General About Device.
3. Tap the Build number seven times to enable Developer option.
4. Return to the Settings menu and select Developer option.
5. Tap the Developer options to turn on USB Debugging from the menu on the next screen.

Automation Testing on an Emulator

An emulator, such as an Android Virtual Device (AVD), is software or hardware that enables a computer system (for example, a Windows operating system) to behave like another computer system (for example, an Android platform). It provides a virtual environment of another system.

To run automation testing on an emulator on Microsoft Windows

1. Start the AVD Manager as follows:

    a. Navigate to and open the SDK folder. For example: C:\SDK
    b. Double-click AVD Manager.exe to start the AVD Manager.

    The AVD Manager is used to create virtual Android devices.
2. Using AVD Manager, create an android virtual device (emulator) as follows:

    a. Go to the Device Definitions tab, click Create Device, click the Android Virtual Device, and then click Create.

    The Create new Android Virtual Device window appears
    b. Enter values for the fields shown in the following table.

    > **Note:** The values shown in the following table are example values for creating a virtual device using Galaxy Tab S 10.5. Values typically vary depending on the device you want to emulate. When creating a virtual device, use the following specifications:
    >
    > - Screen Size : 10.5 inches
    > - Resolution : 2560 x 1600

Field	Description	Sample Value
AVD Name	Name of the Android virtual device.	AVD_for_Samsung_GalaxyTab_S
Device	Name of the Android device.	Samsung_Galaxy_Tab_S
Target	Name of the target Android device.	Android 5.0.1 - API [Level2]
CPU/AB1	The application binary interface.	ARM (armeabi-v7a)

    c. Select the device and then click Start.
    d. Click Launch.

    The Emulator Starts.

3. Update the XML file by setting the following parameters:
    - APPLICATION_TYPE= Mobile_Chrome - Automation on Chrome Browser
    - APPLICATION_TYPE= Mobile_Native Browser - Automation on AVD Native Browser.

      You must start the emulator manually before triggering a run since emulators take more time to start.
    - APPLICATION_TYPE=Mobile_SM_Android - Automation on Siebel MAF Application

Deploying the Siebelmobile.apk

The following procedure explains how to deploy the Siebelmobile.apk.

To deploy the Siebelmobile.apk

1. From the JDeveloper Menu, select Application, Application Properties, then Android2.
2. Double click Android2 (MAF for Android) option.

   The MAF for Android Deployment Profile Properties dialog box is displayed. Ensure that the Build Mode is set to Debug and Application Name is SiebelMobile.
3. Click OK.
4. Navigate to Application, select Deploy Android2.

   The Deploy Android2 dialog box is displayed.
5. Select Deploy application to package from the available list of deployment options.
6. Check the Deploy message displayed in the lower pane of the Deploy Android2 dialog box.

   The message "`Deploy the mobile application to an Android deployment package`" is displayed in the lower pane.
7. Click Finish.

   The mobile application is deployed successfully in the Android deployment package.

15 REST API Reference

REST API Reference

This chapter describes the REST APIs available for testing related objects and actions. It includes the following topics:

- *Create a Test Execution Record*
- *Rerun a Test Execution Record*
- *Create Test Passes for a Test Execution Record*
- *Querying for a Test Execution Record*
- *REST API for Data Sets*
- *REST API for Test Script*
- *REST API for Test Set*
- *REST API for Master Suite*

Create a Test Execution Record

You create a Test Execution record and request automated test runs by sending an HTTP POST to the URI.

The sequence involves a POST for TestExecution first, followed by a POST for ScheduleRun. The first POST request creates a Test Execution record with the appropriate server credentials and the second POST request prepares the attachments necessary for execution, including the user credentials, and sets Status to Requested.

The following example shows a request to create a Test Execution record, using Test Execution Profile, and return the Run Id.

1. Create Test Execution Record
    - URL:

      ```
      https://ServerName:Port/siebel/v1.0/service/Automation Rest
      Service/TestExecution
      ```

    - Method: POST
    - Content-Type: application/json
    - Authorization: Basic

    Request Body

    ```
    {
    "body":
    {
    "Profile Name": "My_Profile",
    "Application Version":"1-BUILD",
    "Master Suite Name":"CORE_UIF1",
    "Description":"Acceptance test Scenarios",
    ```

```
"Run Reference":"Client_Pool_A",
"Client OS":"Windows",
"ServerCredentials":
[{
"Application Alias":"Siebel Universal Agent",
"Application Type":"Desktop_Chrome",
"URL":"https://ServerName:Port/siebel/app/callcenter/enu",
"Server OS Type":"Windows"
},
{
"Application Alias":"Siebel Web Tools",
"Application Type":"Desktop_Chrome",
"URL":"https://ServerName:Port/siebel/app/callcenter/enu",
"Server OS Type":"Windows"
},{
"Application Alias":"CORE_UIF",
"Application Type":"Desktop_Chrome",
"URL":"https://ServerName:Port/siebel/app/callcenter/enu",
"Server OS Type":"Windows"
}]}}
```

Response:

```
{
 "Id": "88-3EQD7V",
 "Server Credentials": "NEW_REST;Siebel Universal Agent",
 "Users List": "SADMIN"
}
```

Note:

- Ensure there is a record My_Profile in the Test Execution Profile Applet, and if it has valid values for `Default Application URL` and `Default REST URI` parameters, URL can be omitted in the Request body.
- In the example, if the Profile parameter is "" (blank) or if not included in REST API call, then hard coded values for System Parameters are used.
- Test Execution Profile marked as Default is not set automatically.

2. Schedule the Run

   - URL:

     https://ServerName:Port/siebel/<Version>/service/Automation **Rest Service/ScheduleRun**

   - Method: POST
   - Content-Type: application/json
   - Authorization: Basic

   Request Body

```
{
 "body": {
 "Test Run Id": "88-3EQD7V",
```

```
 "Override Validation": "Y",
 "Users List": "SADMIN:ldap"
 }
}
```

Response

```
{
 "Status": "Run Request is Successfully created. 88-3EQD7V"
}
```

Backward Compatibility:

The Test Execution Profile is introduced in 25.8. For compatibility with prior versions, following section retains the examples from earlier versions.

The following shows a request to create a Test Execution record without a Test Execution Profile and return the Run Id.

```
URL: https://ServerName:Port/siebel/v1.0/service/Automation Rest Service/TestExecution
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request Body:
{
"body":
{
"Application Version":"1-BUILD",
"Master Suite Name":"CORE_UIF",
"Description":"Acceptance test Scenarios",
"Run Reference":"Client_Pool_A",
"Client OS":"Windows",
"ServerCredentials":
[
{
"Application Alias":"Siebel Universal Agent",
"Application Type":"Desktop_Chrome",
"URL":"https://ServerName:Port/siebel/app/callcenter/enu",
"Server OS Type":"Windows"
},
{
"Application Alias":"Siebel Financial Services",
"Application Type":"Desktop_Chrome",
"URL":"https://ServerName:Port/siebel/app/fins/enu",
"Server OS Type":"Windows"
}
]
}
}
Response to a successful request:
{
"Id": "88-1WCI9B",
"Server Credentials": "Siebel Universal Agent;Siebel Financial Services",
"Users List": "SADMIN;DBROWN"
}
```

Note: If application aliases are not available on firing the request, the response will provide the missing aliases.

The following shows a request to create a Schedule Run for a given Run Id, created using a Test Execution request.

```
URL: https://ServerName:Port/siebel/<Version>/service/Automation Rest Service/ScheduleRun
HTTP Method: POST
Content type: application/json
```

```
Authorization: Basic

Request body:
{
"body":
{
"Test Run Id":"88-1WCI9B",
"Override Validation": "Y",
"Users List":"SADMIN:pwd;DBROWN:pwd"
}
}
Response to a successful request:
{
"Status": "Run Request is successfully created 88-1WCI9B"
}
}
```

The following shows a sample request containing all parameters for a Test Execution.

```
Request Body:
{
"body":
{
"Application Version":"23086",
"Master Suite Name":"COREUIF",
"Description":"Test Run For monthly build",
"Run Reference":"Client_Pool_A",
"Client OS":"Windows",
"Schedule Run At":"09/26/2020 05:32:07",
"Perl Path":"c:\\perl\\bin\\perl.exe",
"Siebel Tools Machine":"ServerName",
"Siebel Tools Path":"c:\\23082",
"Siebel Tools User Name":"user1",
"Siebel Tools DSN":"serverdsn",
"Siebel Tools DSN User":"userid",
"EAI Machine Name":"eaiserverName",
"EAI Port Number":"16690",
"EAI User Name":"sadmin",
"BIP Outbound WS":"http://serverName:port/xmlpserver/services/publicreportservice_v11",
"BIP Server Name":"serverName",
"BIP Server Port":"9500",
"BIP XMLP Server":"serverName",
"BIP XMLP Server Port":"9502",
"BIP User Name":"userName",
"Client Machine":"<MAC IP address>",
"MAC-Username":"userId",
"Mobile Port":"9090",
"ServerCredentials":
[
{
"Application Alias":"Siebel Universal Agent",
"Application Type":"Desktop_Chrome",
"Server OS Type":"Windows",
"URL":"https://ServerName:Port/siebel/app/callcenter/enu",
"Server Home Path":"c:\\23082",
"Siebel Server Machine":"serverName",
"Siebel Server User" :"user1",
"Gateway Machine":"serverName",
"Siebel Gateway Port":"9390",
"AI Server Port":"16690"
},
{
"Application Alias":"Siebel Financial Services",
"Application Type":"Desktop_Chrome",
"Server OS Type":"Windows",
"URL":"https://ServerName:Port/siebel/app/fins/enu",
```

```
"Server Home Path":"c:\\23082",
"Siebel Server Machine":"serverName",
"Siebel Server User" :"user1",
"Gateway Machine":"serverName",
"Siebel Gateway Port":"9390",
"AI Server Port":"16690"
}
]
}
}
```

The following shows a sample request to containing all parameters for a Schedule Run.

```
Request Body:
{
 "body":
 {

 "Test Run Id":"88-1WCI9B",
 "Override Validation": "Y",
 "MAC Machine Pwd": "pwd1",
 "Siebel Tools Machine Pwd":"toolspwd",
 "Siebel Tools DSN Pwd": "toolsdsnpwd",
 "BIP Server Pwd": "bipserverpwd",
 "EAI Server Pwd": "EAIpwd",
 "Server Credentials": "Siebel Universal Agent:pwd;Siebel Financial Services:pwd2",
 "Users List":"SADMIN:pwd;DBROWN:pwd",
 "Parameters":
 [
 {
 "Name":"SSOUser",
 "Value":"usernamevalue"
 },
 {
 "Name":"SSOPwd",
 "Value":"pwd"
 }

 ]
 }
}
```

Rerun a Test Execution Record

The following shows a request to submit a Rerun for an already completed Test Execution record. Preconditions are that Status is set to Completed and that the Reports zip and other attachments are available from prior run(s).

```
URL: https://ServerName:Port/siebel/<Version>/service/Automation Rest Service/Rerun
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request body:
{
"body":
{
"Id":"88-1WCI9B"
}
}
Response to a successful request:
{
```

```
"Status": "Run Request is successfully created 88-1WCI9B"
}
```

Create Test Passes for a Test Execution Record

From 25.8 onwards, you don't need to make an explicit REST API call to create test passes, as it is automatically done along with Reports upload. However, if you still to create them, then using the REST API for 'CreateTestPasses', will delete existing Test Passes and Test Results, if any, and create the new entries.

The following shows a request to Create Test Passes once a run is completed. The latest XML file (indicated by the highest suffix number in the file name) in the Attachments applet will automatically be selected as input to create test pass entries. Preconditions are that Test Sets are associated with Test Plans and that Test Pass has the Completed LOV.

```
URL: https://ServerName:Port/siebel/<Version>/service/Automation Rest Service/CreateTestPasses
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request body:
{
"body":
{
"Id":"88-1WCI9B"
}
}
Response to a successful request:
{
"Status": "Test Pass Entries Created Successfully. Query in Test Pass View Starting with 88-1WCI9B"
}
```

- Test passes will be automatically generated upon completion of execution and hence Create Test Passes button will be disabled.
- The "Create Test Passes" button is enabled when the "Import Test Step Results" profile parameter is set to "N".

Button Behavior

When you click the "Create Test Passes" button:

1. **Initial Click**: A confirmation dialog is displayed with the message: "Test Pass and the child entries are already created for this Test Execution. Do you want to re-create Test Passes? Earlier ones will be deleted permanently."
2. **Confirmation**: If you confirm, the Test Passes will be re-created, and the earlier ones will be deleted permanently.
3. **Test Passes and Test Step Results Already Created**: The button is disabled when both Test Passes and Test Step Results are created.
4. **Test Passes Deleted**: If Test Passes are deleted and you click the "Create Test Passes" button again, the confirmation dialog will not appear, and Test Passes will be created. However, Test Step Results will not be uploaded.

Querying for a Test Execution Record

The following shows a request to query the details of a Test Execution record, and can also be used to query the Status of a Test Execution record.

```
URL: https://ServerName:Port/siebel/<Version>/data/Keyword Automation/Automation Exec Config/88-1WCI9B
```

```
HTTP Method: GET
Content type: application/json
Authorization: Basic

Response to a successful request:
{
"Id": "88-1WCI9B",
"Status": "Completed",
"Master Suite Name": "CORE_UIF1"
}
```

REST API for Data Sets

The following shows a request to create a data set record.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Data Set/Automation Data Set
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request body:
{
"Name": "Test_DataSet",
"Status": "Active",
"Description": "DataSet"
}
Response to a successful request:
{
"Id": "88-1WCIA5",
"Name": "Test_DataSet",
"Status": "Active",
"Description": "DataSet"
}
```

The following shows a request to insert a new field into a data set.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Data Set/Automation Data Set/88-1WCIA5/
Automation Data Set Field
HTTP Method: PUT
Content type: application/json
Authorization: Basic

Request body:
{
"Id": "123",
"Name": "Name",
"Applet Name": "Account List Applet",
"Description": "List Applet Field"
}
Response to a successful request:
{
 "Id": "88-1WCIAF"
}
```

The following shows a request to insert a new value for a field in a data set.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Data Set/Automation Data Set/88-1WCIA5/
Automation Data Set Field/88-1WCIAF/Automation Data Set Values
HTTP Method: PUT
Content type: application/json
Authorization: Basic
```

```
Request body:
{
"Id":"123",
"Sequence":"1",
"Data Set Field Value": "Value1"
}
Response to a successful request:
{
"Id": "88-1WCIAP"
}
```

Note: Before deleting any data set using a REST request, make sure to disassociate any test scripts first.

REST API for Test Script

The following shows a request to create a a test script record.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Test Script/Automation Test Script
HTTP Method: POST
Content type: application/json
Authorization: Basic
Request body:
{
"Name": "Test_Script1",
"Status": "Active",
"Description": "Script1",
"Test Env": "Test",
"DataSet Name": "Test_DataSet1"
}
Response to a successful request:
{
"Id": "88-1WCIB5"
}
```

The following shows a request to query a test step record.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Test Script/Automation Test Script/88-1WCIB5/
Automation Test Step Child
HTTP Method: GET
Content type: application/json
Authorization: Basic

Response to a successful request:
{
"Id": "88-1WCIB8"
}
```

Note: Only the Get method is supported for test steps.

REST API for Test Set

The following shows a request to create a test set record.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Test Set/Automation Test Set
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request body:
{
"Name": "Test_Set",
"Status": "Active",
"Description": "New Set",
"Test Plan Name": "UIFControls_Scenarios"
}
Response to a successful request:
{
"Id": "88-1WCIBT"
}
```

The following shows a request to insert a test script record into a test set.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Test Set/Automation Test Set/88-1WCIBT/
Automation Test Script Child
HTTP Method: PUT
Content type: application/json
Authorization: Basic

Request body:
{
"Id": "123",
"Sequence": "1",
"Name": "Test_Script1"
}
Response to a successful request:
{
"Id": "88-1WCIBT"
}
```

REST API for Master Suite

The following shows a request to create a Master Suite record, returning Row id.

```
URL: https://ServerName:Port/siebel/<Version>/data/Automation Master Suite/Automation Master Suite
HTTP Method: POST
Content type: application/json
Authorization: Basic

Request body:
{
"Name": "Test_Mastersuite",
"Status": "Active",
"Description": "New Msuite",
"Test Env": "Test",
"Release": "IP2017",
"Team Name": "Falcons"
}
Response to a successful request:
{
"Id": "88-1WCICN"
}
```

The following shows a request to insert a test set record into a Master Suite.

```
https://ServerName:Port/siebel/<Version>/data/Automation Master Suite/Automation Master Suite/88-1WCICN/
Automation Test Set Child
HTTP Method: PUT
Content type: application/json
Authorization: Basic

Request body:
{
"Id":"123",
"Sequence": "1",
"Name": "Test_Set",
"Skip": "N"
}
Response to a successful request:
{
"Id": "88-1WCICX"
}
```

16 Keywords Reference

Keywords Reference

This chapter defines the keywords that are available for Siebel Open UI keyword automation testing and describes how to use each keyword. It includes the following topics:

- *Keywords Description*
- *Keywords Supporting Tools and Server Configuration*
- *Unsupported Keywords for Siebel Open UI Keyword Automation*

Note: Before using any keywords, you must import test scripts to the database (there is an Import button on the recording panel) – you can also manually add test scripts by navigating to Site Map, Release Screen, then the Test Scripts view. Once a test script is imported, you can add test steps to the script. Recording captures interaction steps only and any other actions (such as Compare, GetValue, Verify, and so on) must be manually added to the test script.

Keywords Description

This topic describes each keyword that is supported for Siebel Open UI keyword automation testing. The following table defines each keyword, outlines whether it applies to Desktop or Mobile (or both), and includes a link to more detailed information.

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
Attachment Manager	Performs actions on the Attachment Download Manager page in Siebel Mobile Application Framework (MAF).	No	Yes
ClickButton	Clicks on a button control.	Yes	Yes
ClickLink	Drills down on a link in a list applet, drills down on recently viewed links on the homepage, and shows more or less objects.	Yes	Yes
ClickOnChart	Drills down on a requested series or category, provided by the user, on a Chart applet.	Yes	No
ClickSyncButton	Clicks the Sync button to navigate from offline to online and online to offline based on the user provided options. It also verifies the state of the application after the specified navigation.	No	Yes
ClickTopNotification	Clicks the first unread message in the notification list.	Yes	No

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
ColumnsDisplayed	Adds or removes columns in the list applet column (which is an option available in the applet menu).	Yes	Yes
CompareValue	Compares a variable value with the expected value.	Yes	Yes
CreateRecord	Creates a new record in a list or form applet.	Yes	Yes
CustomExtension	Runs a custom extension JAR file.	Yes	Yes
DoubleClick	Double clicks on an element in a user specified applet.	Yes	No
DragAndDrop	Selects a record and moves it to a particular field.	Yes	Yes
Draw	Captures a signature.	Yes	Yes
FileDownload	Downloads (and exports) a file.	Yes	No
FileUpload	Attaches or uploads (or imports) a file.	Yes	Yes
GetAboutRecord	Reads the values (Row Id, Created by, Date, and so on) from About Record (selected from the Siebel applet menu) and stores them in a user variable.	Yes	Yes
GetChartType	Obtains the type of chart in an applet and stores the value in a user variable.	Yes	No
GetConfigParam	Reads the value from the **`unitconfig.xml/ batchconfig.xml`** file and stores the value in a user variable.	Yes	Yes
GetRecordCount	Obtains the total number of records and stores the value in a user variable.	Yes	Yes
GetState	Obtains the state of a specified object and stores the value in a user variable.	Yes	Yes
GetValue	Obtains the value of a specified object and stores the value in a user variable.	Yes	Yes
GetValueFromMenuPopup	Reads a value from an application level pop-up menu.	Yes	No
GoToSettings	Views and changes the default settings of a user profile.	Yes	Yes

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
GoToThreadbarView	Navigates to a view that is available in the Threadbar link.	Yes	Yes
GoToView	Navigates to a specified view using the Tab view, Tree view, or Site Map links.	Yes	Yes
HierarchicalList	Expands or collapses a record in a hierarchical list applet, and shows the child items.	Yes	No
InboundWebServiceCall	Reads the XML request from a .xml file, posts the request to the server, and saves the XML response from the server.	Yes	No
InputValue	Enters a value into a specified field.	Yes	Yes
InvokeAppletMenuItem	Invokes a menu item from an applet-level menu in a list or form.	Yes	Yes
InvokeMenuBarItem	Invokes a menu item from the application menu bar.	Yes	Yes
InvokeObject	Invokes an object in a specified field in a list or form applet.	Yes	Yes
InvokeREST	Invokes a REST API call from within a test script to execute the API, verify and/or use the response.	No	No
Launch	Starts the browser and logs in to an application with the provided username.	Yes	Yes
LockColumn	Locks or unlocks a selected column.	Yes	Yes
LogOut	Logs out from an application.	Yes	Yes
MafSettings	Performs the required action in the MAF Settings page in MAF applications.	No	Yes
MultiSelectRecordsInListA	Selects one or more rows in a list applet.	Yes	Yes
QueryRecord	Queries an existing record from a list or form applet.	Yes	Yes
RemoveFromMvg	Removes a record from the list in a multi-value group (MVG).	Yes	Yes
SelectCheckBox	Selects or clears a check box depending on the provided value (True or False).	Yes	Yes

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
SelectFromMvg	Select the specified record after querying the available records in an MVG.	Yes	Yes
SelectFromPickApplet	Queries and selects the first record from a drop-down list applet.	Yes	Yes
SelectPDQValue	Selects a value from the Predefined Dropdown Query (PDQ).	Yes	Yes
SelectPickListValue	Selects a value from a drop-down list in a list or form applet.	Yes	Yes
SelectRadioButton	Selects a radio button.	Yes	No
SelectRecordInListApplet	Selects a record from a list applet.	Yes	Yes
SelectToggleValue	Selects a value from a toggle control in a list applet.	Yes	No
SelectVisibilityFilterValue	Selects a value from the Visibility Filter drop-down list.	Yes	No
SendKeys	Trigger a keyboard event.	Yes	Yes
SetDateTime	Calls the DateTime or Date pop-up calendars to specify the date and time.	Yes	Yes
SortColumn	Sorts a selected column.	Yes	Yes
SwitchTab	Switches between the browser tabs.	Yes	Yes
TreeExplorer	Expands or collapses an explorer tree, and selects items from or shows items under an explorer tree.	Yes	No
VerifyColumnLockStatus	Verifies the lock status of a column.	Yes	Yes
VerifyColumnSortOrder	Verifies the order of records in a selected column.	Yes	Yes
VerifyError	Verifies the error message for a string value.	Yes	Yes
VerifyFileLoad	Performs image validation.	Yes	Yes
VerifyFocus	Verifies the focus present on an applet, view, field, or row depending on the value provided (True or False).	Yes	Yes

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
VerifyInPicklist	Counts the number of items in a drop-down list, auto selects using substring, and verifies the values in the drop-down list.	Yes	Yes
VerifyObject	Verifies the presence of an object or the UI name for an object.	Yes	Yes
VerifyRecordCount	Verifies the row count in a list applet.	Yes	Yes
VerifyState	Verifies the state of a specified field.	Yes	Yes
VerifyTopNotification	Verifies whether the first read or unread message in the notification list appears or not.	Yes	Yes
VerifyValue	Verifies a field value by comparing it with a user variable.	Yes	Yes
Wait	Allows the application to stay idle for the user specified time.	Yes	Yes
StartLoop	Marks the start of a loop, and initializes the optional parameter to 1. Increments the counter each time the execution comes back to this step	Yes	Yes
EndLoop	Marks the end of Loop, checks the condition expression and the max iteration. If condition is evaluating to true or if max iteration is met, exits the loop.	Yes	Yes

Using Variables in Test Scripts

In Siebel Test Automation, you can use variables to store transient or dynamic values. For instance, at run time, you may read a value from application into a variable using *GetValue* keyword, and later query a record using the variable. Same variable can be referenced across Test Scripts used in a Master Suite.

Any name prefixed with @ symbol is treated as a variable. For example: `@VarName`. It should be one word without any spaces. Variable names are case-sensitive.

AttachmentManager

You use the AttachmentManager keyword to perform actions in the Attachment Download Manager page in MAF applications.

Note: The AttachmentManager keyword works only on MAF iOS and MAF Android devices.

Signature

The AttachmentManager keyword supports the following signature:

```
AttachmentManager(AppletRN, Name of the Entity|Name of the File Name, ...;DOWNLOAD/
DOWNLOADALL/REMOVE/CLOSE/TOP)
```

AttachmentManger signature supports the following actions:

```
DOWNLOAD
DOWNLOADALL
TOP
UPLOAD
UPLOADALL
UPLOADTOP
REMOVE
REMOVEALL
=filename with extension
=Error - File Not Found
=100%
=Error - Network Error
=Sync Data First
```

Desktop Examples

The AttachmentManager keyword does not apply to desktop applications.

Mobile Examples

The following table describes how to use the AttachmentManager keyword to perform actions in the Attachment Download Manager page in MAF applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
Download Attachment Applet	Download Attachment Applet	AdminSalesTool|pills1.jpg;DOWNLOAD;IPH3	To download user specified files.
Download Attachment Applet	Download Attachment Applet	AdminSalesTool|Zonall_Launch.zip,AdminSalesTool|ZonallPatientProfile.zip;DOWNLOAD;IPH3	To download user specified files.
Download Attachment Applet	Download Attachment Applet	AdminSalesTool|pills1.jpg;REMOVE;IPH3	To remove the selected files.
Download Attachment Applet	Download Attachment Applet	AdminSalesTool|Zonall_Launch.zip;TOP;IPH3	To move the file to the first (TOP) position.

Target Object	Inputs	Closing Action	Comments
Download Attachment Applet	Download Attachment Applet	NULL;DOWNLOADALL;IPH3	Selects all the files and performs the action when the Download button is clicked.
Download Attachment Applet	Download Attachment Applet	NULL;REMOVEALL;IPH3	Selects all the files and performs the action when the Remove button is clicked.
Download Attachment Applet	Download Attachment Applet	AccountAttachment|pills1.jpg;=Error - File Not Found;IPH3	To verify Status of the user specified in Download Attachment - the equals sign ('=') signifies verification.
Download Attachment Applet	Download Attachment Applet	NULL;=pills1.jpg;IPH3	To verify if the user specified file (pills1.jpg) is available in the Download Attachment - the equals sign ('=') signifies verification.
Download Attachment Applet	Upload Attachment Applet	AccountAttachment|Test.jpeg;UPLOAD;IPH3	To upload user specified files.
Download Attachment Applet	Upload Attachment Applet	AccountAttachment|Test.jpeg;=Sync Data First;IPH3	To verify Status of the user specified in Upload Attachment - the equals sign ('=') signifies verification.
Download Attachment Applet	Upload Attachment Applet	AccountAttachment|Test.jpeg;UPLOADTOP;IPH3	To move the file to the first (TOP) position.
Download Attachment Applet	Upload Attachment Applet	NULL;UPLOADALL;IPH3	Selects all the files and performs the action when the Upload button is clicked.
Download Attachment Applet	Upload Attachment Applet	NULL;=Test.jpeg;IPH3	To verify if the user specified file (Test.jpeg) is available in the Upload Attachment - the equals sign ('=') signifies verification.
Download Attachment Applet	Upload Attachment Applet	AccountAttachment|Test.jpeg,AccountAttachment|pills.jpeg;UPLOAD;IPH3	To upload user specified files.

Target Object	Inputs	Closing Action	Comments
Download Attachment Applet	Attachment Applet	NULL;Close;IPH3	To close the Attachment Manager.

ClickButton

You use the ClickButton keyword to click on a button control present in any list or form applet or in any multi-value group or drop-down applet, and to click Close (the X icon) to close a pop-up window.

Signature

The ClickButton keyword supports the following signature:

```
ClickButton(AppletRN|ButtonRN,OK/CANCEL/NULL)
```

Note the following about the ClickButton keyword signature:

- If the action is to be performed on Tile applets, then the SelectRecordInListApplet keyword should be used before the Click button.
- You must provide OK and Cancel options in case a Delete confirmation dialog box is expected.
- You must provide NULL for other buttons, even if a pop-up window or dialog box is expected. Other keywords will carry out any subsequent action on the pop-up window or dialog box.

Desktop Examples

The following table describes how to use the ClickButton keyword to click on button controls in desktop applications.

Target Object	Inputs	Closing Action	Comments
Synergy Toolbar	N/A	NULL	Clicks the Delete Record button in a list applet and handles the button in the confirmation pop up.
NULL|SiebTabViews	N/A	NULL	In SUI theme, invokes the L2 level links that appear when a button is clicked.
SIS Account List Applet|NewQuery	N/A	NULL	Clicks the Delete Record button in a list applet and handles the button in the confirmation pop up.
SIS Account List Applet|DeleteRecord	N/A	OK	Clicks the Delete Record button in a list applet.
SIS Account Entry Applet|GotoNextSet	N/A	NULL	Clicks the Go to Next Set button in a form applet.
Product Pick Applet(Eligibility)|PickRecord	N/A	NULL	Clicks the drop-down list in a pop-up window.
NULL|PickRecord	N/A	NULL	Clicks a button in a pop-up window.

Target Object	Inputs	Closing Action	Comments
Close	N/A	OK	Clicks Close or X button in a pop-up window and then clicks OK.
Close	N/A	Cancel	Clicks Close or X button in a pop-up window and then clicks Cancel.

Mobile Examples

The following table describes how to use the ClickButton keyword to click on button controls in desktop mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Contact List Applet - Mobile|DeleteRecord	N/A	OK	Clicks the Delete Record button in a list applet.
SHCE Sales Contact List Applet - Mobile|Create	N/A	NULL	Clicks the Create button in a list applet.
Close	N/A	OK	Clicks Close or X button in a pop-up window and then clicks OK.
NULL|NewQuery	N/A	NULL	Clicks a button in a pop-up window.

ClickLink

You use the ClickLink keyword to drill down on links in a list applet, to drill down on recently viewed links on the homepage, and to show more or show less objects.

Signature

The ClickLink keyword supports the following signatures:

```
ClickLink(AppletRN|FieldRN/ClassName/RowId|[RowNum],Value/Variable/NULL/ShowMore/
Show Less/Expand/Collapse)
```

Note: The row number is optional in this signature. If row number is not provided, then the first row will be used by default.

```
ClickLink(AppletRN|@Var ,Value/Variable)
```

Note: @var support in FieldRN is available only for Mobile application.

```
ClickLink(AppetRN|TimeslotRN ,NULL)
```

Note: This signature clicks on the given Timeslot in the Calendar applet.

```
ClickLink(AppetRN|@FirstName+@LastName+TimeslotRN,NULL)
```

Note: This signature provides a support for clicking on dynamic Timeslot in the Calendar applet.

Desktop Examples

The following table describes how to use the ClickLink keyword to drill down on links in desktop applications.

Target Object	Inputs	Closing Action	Comments
ClickLinkAccount Contact List Applet|Last Name	<Ramakrishna>	N.A	Drills down on the Last Name value. For example, drills down on the last name value Ramakrishna.
ClickLinkAppletRN| ClassName	NULL	N.A	Clicks on the link based on the Class Name.
NULL|SiteMap	NULL	N.A	Clicks on the toolbar item (Site Map).
Account Contact List Applet|Last Name|2	Pinas	N.A	Drills down on the Last Name of the second record with the value provided in the inputs.
SIS Account List Applet| ToggleListRowCount	NULL	N.A	Clicks the Show More link in the list applet.
SIS Account List Applet| ToggleListRowCount	Show More	N.A	Clicks the Show More link in the List applet.
SIS Account List Applet| ToggleListRowCount	Show Less	N.A	Clicks the Show Less link in the List applet.
LS Pharma Inbox Applet| ButtonMaximizeApplet	Expand	N.A	Expands the list in the Home page.
LS Pharma Inbox Applet| ButtonMinimizeApplet	Collapse	N.A	Collapses the list in the Home page.
Contact Home Public and Private View Link List Applet|Name	My Contacts	N.A	Clicks the link of the recently viewed contacts screen (My Contacts).
NULL|Save Query As Applet.SaveAs	NULL	N.A	Clicks the button link in the pop-up window.
Account Home Public and Private View Link List Applet|2	All Accounts	N.A	Click the frequently viewed links.

Target Object	Inputs	Closing Action	Comments

Mobile Examples

The following table describes how to use the ClickLink keyword to drill down on links in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Contact List Applet - Mobile|Last Name|1	<last name>	N/A	Drills down on the name value in the first row.
SHCE Sales Contact List Applet - Mobile|Last Name	<last name>	N/A	Drills down on the name value.
LS Home Page Calendar Applet-Mobile|@Var	NULL	N/A	Drills down on a call in the calendar applet. You can obtain the row Id of the record using GetAboutRecord keyword by navigating to the view.
LS Home Page Calendar Applet-Mobile|slot-0-090000MikeAdlerDr.	NULL	N/A	Clicks the specified timeslot in the Calendar applet.
LS Home Page Calendar Applet-Mobile|@AccountCall +slotCol-0-10:00	NULL	N/A	Supports clicking on the dynamic Timeslot in the Calendar applet (Account call).
LS Home Page Calendar Applet-Mobile|@FirstName +@LastName+@title +slotCol-0-10:00	NULL	N/A	Supports clicking on the dynamic Timeslot in the Calendar Applet (Contact call).
LS Home Page Calendar Applet-Mobile|@Accountname+@Date	NULL	N/A	Supports clicking on the dynamic activity in the Calendar applet.

ClickOnChart

You use the ClickOnChart keyword to drill down on a required series or category, provided by the user, on a Chart.

Signature

The ClickOnChart keyword supports the following signature:

```
ClickonChart(AppletRN, Series;Category)
```

Desktop Examples

The following table describes how to use the ClickOnChart keyword in desktop applications.

Target Object	Inputs	Closing Action	Comments
Oppty Chart Applet - Campaign Analysis	1; 0	N/A	Drills down on the chart based on the series and category specified by the user.

Mobile Examples

The ClickOnChart keyword does not apply to mobile applications.

ClickSyncButton

Depending on user provided options, you use the ClickSyncButton keyword to click on the Sync button to switch from offline to online mode or from online to offline mode. The keyword also verifies the state of the application after the specified navigation.

Signature

The ClickSyncButton keyword supports the following signature:

```
ClickSyncButton(RnofSyncButton,RnofOfflineOptions;RnofStateoftheApplication)
```

Desktop Examples

The ClickSyncButton keyword does not apply to desktop applications.

Mobile Examples

The following table describes how to use the ClickSyncButton keyword to switch between online and offline modes in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
GoOffline	NULL;Offline	N/A	Clicks the Sync button and moves to offline mode.
GoOffline	uploadGoOnline;Online	N/A	Clicks the Sync button and selects the option uploadGoOnline in offline mode.
GoOffline	syncStayOffline;Online	N/A	Clicks the Sync button and selects the option syncStayOffline in offline mode.

Target Object	Inputs	Closing Action	Comments
GoOffline	uploadStayOffline;Online	N/A	Clicks the Sync button and selects the option uploadStayOffline in offline mode.

ClickTopNotification

You use the ClickTopNotification keyword to click and drill down on the first unread message in the notifications list. You also use the keyword to click the Mark All as Read option in the notifications list, and then close the notifications list.

Signature

The ClickTopNotification keyword supports the following signature:

```
ClickTopNotification(MessagebroadcastRN,Expectedmessage,Close/KeepOpen)
```

Note: The user must click on Mark All as Read option before using the click operation on any message.

ClickTopNotication checks for notification messages for up to ten iterations (with an interval of one minute for each of the iterations).

Desktop Examples

The following table describes how to use the ClickTopNotification keyword to drill down on the first unread message in the notifications list.

Target Object	Inputs	Closing Action	Comments
MsgBrdCstIcon	Account_10142015_041155918	Close	Clicks the first unread message in the notifications list, and closes the control.
MsgBrdCstIcon	Mark All As Read	Close	Clicks the Mark all as Read option in the notifications list, and closes the control.
MsgBrdCstIcon	Mark All As Read	KeepOpen	Clicks the Mark all as Read option in the notifications list, and keeps the control open.
MsgBrdCstIcon	NULL	Close	Closes the notification control.
MsgBrdCstIcon	Account_10142015_041155918	KeepOpen	Clicks the first unread message in the notifications list and keeps the control open.

Mobile Examples

The ClickTopNotification keyword does not apply to mobile applications.

ColumnsDisplayed

You use the ColumnsDisplayed keyword to specify the columns to appear in a list applet, and in what order. You use the keyword to move columns from the Available to the Selected list (or from the Selected to the Available list), and to move columns up and down so that the column order changes as required. The following actions are supported: Save, Reset, and Cancel.

Signature

The ColumnsDisplayed keyword supports the following signature:

```
ColumnsDisplayed(AppletRN|ColumnsDisplayedRN,Select:columnName1|columnName2|..;
DeSelect:columnName1|columnName2|..;
Order:columnName|Up/Down/Top/Bottom,RN (Save/Cancel/Reset Defaults))
```

where:

- The value in columnName must be used for desktop applications.
- The display text in columnName must be used for mobile applications.

Desktop Examples

The following table describes how to use the ColumnsDisplayed keyword to move columns in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet| Columns Displayed (SWE)	Select:Account Team;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Adds the Account Team column to the list applet.
SIS Account List Applet| Columns Displayed (SWE)	NULL;DeSelect:Account Team;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Hides the Account Team column to the list applet.
SIS Account List Applet| Columns Displayed (SWE)	Select:ALL;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Adds all columns to the list applet.
SIS Account List Applet| Columns Displayed (SWE)	NULL;DeSelect:ALL;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Hides all columns to the list applet.
SIS Account List Applet| Columns Displayed (SWE)	NULL;NULL;Order:Account Team|UP	Columns Displayed Popup Applet (SWE).ButtonSave	Orders the columns by moving one position (up) in the list applet.
SIS Account List Applet| Columns Displayed (SWE)	NULL;NULL;Order:Account Team|DOWN	Columns Displayed Popup Applet (SWE).ButtonSave	Orders the columns by moving one position (down) in the list applet.
SIS Account List Applet| Columns Displayed (SWE)	NULL;NULL;Order:Account Team|TOP	Columns Displayed Popup Applet (SWE).ButtonSave	Moves the column to the first position in the list applet.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;Order:Account Team|BOTTOM	Columns Displayed Popup Applet (SWE).ButtonSave	Moves the column to the last position in the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonCancel	Opens the ColumnDisplayed applet, and clicks Cancel.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonReset	Opens the ColumnDisplayed applet and clicks Reset.
SIS Account List Applet|Columns Displayed (SWE)	Select:Account Team;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonReset	Move the column to the last position in the list applet and clicks Reset.

Mobile Examples

The following table describes how to use the ColumnsDisplayed keyword to move columns in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Columns Displayed (SWE)	Select:Account Team;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Adds the Account Team column to the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;DeSelect:Account Team;NULL	Columns Displayed Popup Applet (SWE).ButtonSave	Hides the Account Team column to the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;Order:Account Team|UP	Columns Displayed Popup Applet (SWE).ButtonSave	Orders the columns by moving one position (up) in the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;Order:Account Team|DOWN	Columns Displayed Popup Applet (SWE).ButtonSave	Orders the columns by moving one position (down) in the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;Order:Account Team|TOP	Columns Displayed Popup Applet (SWE).ButtonSave	Moves the columns to the first position in the list applet.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;Order:Account Team|BOTTOM	Columns Displayed Popup Applet (SWE).ButtonSave	Moves the columns to the last position in the list applet.

CompareValue

You use the CompareValue keyword to compare a variable value with the expected value. The expected value can be a variable or value.

Signature

The CompareValue keyword supports the following signature:

```
CompareValue(@Variable|Operator|value (or) @Variable)
```

Operator can be one of the following characters or values:

- = (equals)
- > (greater than)
- < (less than)
- <= (less than or equal to)
- >= (greater than or equal to)
- `contains`
- `startswith`
- `endswith`

Desktop Examples

The following table describes how to use the CompareValue keyword to compare a variable value with the expected value for desktop applications.

Target Object	Inputs	Closing Action	Comments
N/A	@var1|>=|3.56	N/A	Verifies a variable value by comparing it with the expected value 3.56.
N/A	@Var1|=|@Var2	N/A	Verifies a variable value by comparing it with the expected value.
N/A	@Var|startswith|nc	N/A	Verifies a variable value by comparing it with the expected value that starts with nc.
N/A	@Var|endswith|nc	N/A	Verifies a variable value by comparing it with the expected value that ends with nc.
N/A	@Var|=|{[123]}	N/A	Verifies a variable value by comparing it with the expected value.
N/A	@Var|=|"text123;	N/A	Verifies a variable value by comparing it with the expected value text123.

Mobile Examples

The following table describes how to use the CompareValue keyword to compare a variable value with the expected value for mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
N/A	@var1|>=|3.56	N/A	Verifies a variable value by comparing it with the expected value 3.56.
N/A	@Var1|=|@Var2	N/A	Verifies a variable value by comparing it with the expected value.
N/A	@Var|startswith|nc	N/A	Verifies a variable value by comparing it with the expected value that starts with nc.
N/A	@Var|endswith|nc	N/A	Verify a variable value by comparing it with the expected value that ends with nc.
N/A	@Var|=|{[123]}	N/A	Verifies a variable value by comparing it with the expected value.
N/A	@Var|=|"text123;	N/A	"Verifies a variable value by comparing it with the expected value text123.

CreateRecord

You use the CreateRecord keyword to create a new record by entering values into one or more fields in a list or form applet. The keyword fails in list applets if the sequence Id is automatically generated when a user tries to create a new record.

Signature

The CreateRecord keyword supports the following signature:

```
CreateRecord(AppletRN|Button(RN)|[RowNum],FieldRN(1..N)|Value(1...N) OR
Variable,RN of Save record in AppletMenu)
```

Note the following about the CreateRecord keyword signature:

- The row number is optional.
- The keyword supports:
  - Unique values by adding the '$' symbol at the end of the text.
  - Date format like Today,Today+1,Today-1.
  - Variables like @var1, @var2 and numbers.

Desktop Examples

The following table describes how to use the CreateRecord keyword to create a new record in list and form applets in desktop applications.

Target Object	Inputs	Closing Action	Comments
Contact List Applet|NewRecord	Last Name|last,First Name|first	WriteRecord(SWE)	Creates a record in the list applet.
SIS Account List Applet|NewRecord	Name|D$,Account Status|Red Customer	WriteRecord(SWE)	Creates a record in the list applet.
Contact Form Applet|NewRecord	LastName|@var1,FirstName|first	WriteRecord(SWE)	Creates a record in the form applet.
Contact List Applet|NewRecord|3	Last Name|@Variable	WriteRecord(SWE)	Creates a record in the third row in the list applet.

Mobile Examples

The following table describes how to use the CreateRecord keyword to create a new record in list and form applets in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Opportunity Form Applet - Mobile|New	Name|John$,Currency|USD,CloseDate|11|12|12	WriteRecord	Creates a record in the form applet.
CG Account List Applet - Mobile|New	Name|abc	WriteRecord	Creates a record in the list applet.
CG Account List Applet - Mobile|New|4	Name|abc	WriteRecord	Creates a record in the fourth row in the list applet.

CustomExtension

You use the CustomExtension keyword to run a custom extension JAR file by entering the requisite ClassName and inputs to it, and providing the output variables for the return values from the program. To abort the test script, use the Abort-on-Fail end action. The default end action is Continue-on-Fail.

Signature

The CustomExtension keyword supports the following signature:

```
CustomExtension(ClassName,Input Values/Output Variables,Abort-on-Fail/Continue-on-Fail)
```

Note the following about the CustomExtension signature inputs and closing actions.

- **Target Object.** Provide a fully qualified Custom Class Name.

- **Inputs.** The following inputs are required:
  - NULL or a comma separated list of input values; and
  - NULL or a comma separated list of output variable names.
- **Closing Action.** The closing action for the Custom Class Name (or the target object) can be:
  - Abort-on-Fail (that is, abort test script execution).
  - Continue-on-Fail (that is, continue with the next step).

    Continue-on-Fail is the default.

Note: Make sure to backup your extensions folder during a DISA upgrade, install, or uninstall process. For more information on how to build and deploy a custom extension JAR file, see *Extending Keyword Automation Capabilities*.

Desktop and Mobile Examples

The following table describes how to use the CustomExtension keyword in desktop and mobile applications.

Target Object	Inputs	Closing Action	Comments
com.siebel.customobject	value1;@outvalue1;IPH3	Abort-on-Fail	ClassName with no input/output variables
com.siebel.login	value1;@outvar1;IPH3	Abort-on-Fail	ClassName with one value
com.siebel.objclick	value1,value2;@outvar1, @outvar2;IPH3	Continue-on-Fail	ClassName with one or more values
com.siebel.linkclick	@var1,@var2;NULL;IPH3	Continue-on-Fail	ClassName with variables

Handling SSO using CustomExtension:

- Similar to PortalApplication, **SSO** reserved word is supported in Launch Keyword. If SSO is used launch Keyword will only launch the application but will not perform login.

  SSO User Id and Passwords can be provided in Parameters applet of Test Execution. Automatic variables with same names as Parameters and initialized with respective Values are created by framework.
- Implement program to perform SSO login, and invoke via CustomExtension keyword. Pass the SSO credentials to program using variables automatically created from Parameters applet.

Test Step Sequence	Description	Action	Target Object	Inputs
1	Launch without login	Launch	N/A	Siebel Universal Agent;SSO;Y Test Step form applet:

Test Step Sequence	Description	Action	Target Object	Inputs
				Component Alias: **Siebel Universal Age** User Name: **SSO** Clear Browser: **Y**
2	Login steps	CustomExtension	Fully qualified Java class name Example: `com.mycompany.automatio`	@SSOUser,@SSOPwd;@out1;IPH3 Test Step form applet: Class : **Siebel Universal Agent**

DoubleClick

You use the DoubleClick keyword to double click on an element in an applet.

Signature

The DoubleClick keyword supports the following signature:

```
DoubleClick(AppletRN|ItemRN)
```

Desktop Examples

The following table describes how to use the DoubleClick keyword in desktop applications.

Target Object	Inputs	Closing Action	Comments
TNT Function Bookings Gantt Applet|@var1+-event +@FunType+@FuncDate+@FunStart+@FunInvStatus	N/A	N/A	Double clicks on the element.

Mobile Examples

The DoubleClick keyword does not apply to mobile applications.

DragAndDrop

You use the DragAndDrop keyword to select a record in an applet and move it to a particular field.

Note the following about the DragAndDrop keyword signature:

- The records must be visible in the screen.
- In the mobile application, the keyword works only in landscape mode and both source and destination applets must be active.

Signature

The DragAndDrop keyword supports the following signatures:

```
DragAndDrop (SourceAppletRN|NULL|Rowno, DestinationAppletRN|FieldRN)
DragAndDrop (SourceAppletRN|FieldRN, DestinationAppletRN|FieldRN)
```

Desktop Examples

The following table describes how to use the DragAndDrop keyword to select a record from an applet and move it to a specific field in desktop applications.

Target Object	Inputs	Closing Action	Comments
FINCORP Deal Account Pick Applet|NULL|3	Opportunity List Applet|Account	N/A	Selects a record from an applet and moves it to a specific field.
Opportunity List Applet|Account	Opportunity List Applet|Primary Revenue Win Probability	N/A	Swaps the columns in the applet.
NT Function Bookings Gantt Applet|@var1-event+@FunType +@FuncDate+@FunStart + @FunInvStatus	TNT Function Bookings Gantt Applet|@var2+UR	N/A	Selects a record from an applet and moves it to a specific field in a Gantt Chart.
WebControlPalette|WebControl-field	PreviewArea;placeholder-112	N/A	Selects a record from a field in the Webcontrol palette and moves it to placeholder-112 in the preview area of the Web Tools Editor.
WT Repository Applet Edit Web Layout Palette|NULL|2	PreviewArea;placeholder-112;IPH3	N/A	None.

Mobile Examples

The following table describes how to use the DragAndDrop keyword to select a record from an applet and move it to a specific field in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
FINCORP Deal Account Pick Applet|NULL|3	Opportunity List Applet|Account	N/A	Selects a record from an applet and moves it to a specific field.

Target Object	Inputs	Closing Action	Comments
LS Home Page Contact List Applet - Mobile |NULL|3	LS Home Page Calendar Applet-Mobile |slotCol-0-08:00	N/A	Selects a record from the call applet and moves it to the calendar applet (according to the specified rn).

Draw

You use the Draw keyword to capture the signature.

Signature

The Draw keyword supports the following signature:

```
Draw(FieldRN)
```

Desktop and Mobile Examples

The following table describes how to use the Draw keyword to capture a signature in desktop and mobile applications.

Target Object	Inputs	Closing Action	Comments
FieldRN	N/A	N/A	Captures a signature.

FileDownload

You use the FileDownload keyword to download (and export) a file.

Signature

The ServerConfig keyword supports the following signature:

```
FileDownload(FileName,ButtonRN/NULL,Save/Cancel)
```

The file downloads to the following location: `C:\\temp\\Download_File`.

Note: The Exefiles folder must be copied from the TestHarness before execution.

Desktop Examples

The following table describes how to use the FileDownload keyword to download a file in desktop applications.

Target Object	Inputs	Closing Action	Comments
abc.txt	NULL	Save	Downloads the file.
export.csv	SWE Export Applet.btnNext	Cancel	Cancels the download pop-up window.

Mobile Examples

The FileDownload keyword does not apply to mobile applications.

FileUpload

You use the FileUpload keyword to attach and upload (import) a file.

Signature

The FileUpload keyword supports the following signature:

```
FileUpload(AppletRN|FieldRN(or)ButtonRN, FileName , ButtonRN/NULL)
```

Note the following about the FileUpload keyword signature:

- To upload a file, the file must be placed in the FileUpload folder at the TestHarness location.
- The Exefiles folder must be copied from the TestHarness before execution.

Desktop Examples

The following table describes how to use the FileUpload keyword to upload a file to the FileUpload folder in desktop applications.

Target Object	Inputs	Closing Action	Comments
SWE Import Applet|file	InputFileSales.txt	NULL	Uploads the Sales.txt file to the FileUpload folder.
UCM List Import Jobs Form Applet - Job Details|FileName	output.csv	NULL	Uploads the output.csv file to the FileUpload folder.

Mobile Examples

```
FileUpload Attachment List Applet - Mobile|New File IMG_002.PNG;IPH2;IPH3 NULL
```

GetAboutRecord

You use the GetAboutRecord keyword to obtain parameter values from the About Record pop-up window, invoked from the Applet Menu, and store the values in a user variable.

Signature

The GetAboutRecord keyword supports the following signature:

```
GetAboutRecord(AppletRN|RN of AboutRecord,RN of Label(1..N)|Variable(1..N))
```

Desktop Examples

The following table describes how to use the GetAboutRecord keyword to obtain and store information from the About Record pop-up window in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|About Record (SWE)	Row:|@var1	N/A	Stores the row field value from the AboutRecord pop-up window in a variable.
SIS Account Entry Applet|About Record (SWE)	Row:|@var1,Conflict:|@var2, Created_On:|@var3,Created By:| @var4	N/A	Stores the Row, Conflict, Created_On, and CreatedBy field values from the About Record pop-up window in a variable.

Mobile Examples

The following table describes how to use the GetAboutRecord keyword to obtain and store information from the About Record pop-up window in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|About Record (SWE)	RowId|@var1,UpdatedOn:|@var2, CreatedOn:|@var3	N/A	Stores the Row, UpdatedOn, Created_on field values from the About Record pop-up window in a variable.

GetChartType

You use the GetChartType keyword to obtain the type of chart in an applet and store the value in a user variable.

Signature

The GetChartType keyword supports the following signature:

```
GetChartType (AppletRN, @type)
```

Desktop Examples

The following table describes how to use the GetChartType keyword in desktop applications to obtain and store information from the Chart type.

Target Object	Inputs	Closing Action	Comments
Oppty Chart Applet - Campaign Pipeline Analysis	@Var;IPH2;IPH3	N/A	Stores the Chart type value in a variable.

Mobile Examples

The GetChartType keyword does not apply to mobile applications.

GetConfigParam

You use the GetConfigParam keyword to read the values in the config.xml file. The keyword retrieves one value at a time, given the correct parameter name (tagname).

Signature

The GetConfigParam keyword supports the following signature:

```
Tag name till value/Null;?Required Tagname;Variable
```

Note the following about the GetConfigParam keyword signature:

- All three inputs are mandatory to enter.
- If the attribute value to be obtained from the config.xml file is available under any parent tag, then provide the application name in the Inputs column.
- If the attribute value to be obtained from the config.xml file is not available under any parent tag (directly), then provide NULL in the Inputs column.

GetRecordCount

You use the GetRecordCount keyword to obtain the total number of records and store the value in a user variable.

Signature

The GetRecordCount keyword supports the following signature:

```
GetRecordCount(Applet RN|RN of Recordcount Menu item/NULL,Variable)
```

Note: Use NULL for the applets without menu.

Desktop Examples

The following table describes how to use the GetRecordCount keyword to obtain the total number of records and store the value in a user variable in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Record Count (SWE)	@Var1	N/A	Stores the total record count in a user variable.
SIS Account List Applet|NULL	@Var1	N/A	Uses NULL for the applets without menu.

Mobile Examples

The following table describes how to use the GetRecordCount keyword to obtain the total number of records and store the value in a user variable in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile| Record Count (SWE)	@recordcount	N/A	Stores the total record count in a user defined variable.
SHCE Sales Account List Applet - Mobile|	@Var1	N/A	Uses NULL for the applets without a menu.

GetState

You use the GetState keyword to obtain the state of a specified object and store the state in a variable. The state of an object can be Read-only, Enabled, Disabled, Editable, and so on.

Signature

The ServerConfig keyword supports the following signatures:

```
GetState(AppletRN|FieldRN|[RowNum],@Variable)
GetState(AppletRN|MenuButtonRN|MenuItemRN,@Variable)
GetState(ApplicationLevelMenuRN|ApplicationLevelMenuItemRN,@Variable)
```

Desktop Examples

The following table describes how to use the GetState keyword to obtain the state of a specified object and store the state in a variable in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Product List Admin Applet|Release	@variable1	N/A	Stores the Release state in a variable.

Target Object	Inputs	Closing Action	Comments
SIS Product List Admin Applet|XA Class Name|3	@variable2	N/A	Stores XA Class Name state for the third record in a variable.
SIS Product List Admin Applet| SiebAppletMenu|Delete Record (SWE)	@variable3	N/A	Stores the Delete record (SWE) state in a variable.
Menu-File|File - Send Fax	@variable4	N/A	Stores the File - Send Fax state in a variable.

Mobile Examples

The following table describes how to use the GetState keyword to obtain the state of a specified object and store the state in a variable in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile| QuerySrchSpec	@var	N/A	Gets the state of a variable and stores the state in @var.
SHCE Sales Account List Applet - Mobile| SiebAppletMenu|Record Count (SWE)	@var	N/A	Gets the state of a menu item and stores the state in @var.

GetValue

You use the GetValue keyword to obtain the value from a specified object and store the value in a variable.

Signature

The GetValue keyword supports the following signature:

```
GetValue(AppletRN|FieldRN/ClassName/ThreadbarID|[RowNum/Tileindex],@Variable)
```

Note: If the action is to be performed on tile applets, then the Tile index or row number must start from one.

Desktop Examples

The following table describes how to use the GetValue keyword to obtain the value from the specified object and store the value in a variable in desktop applications.

Target Object	Inputs	Closing Action	Comments
AppletRN|ClassName|[RowNumber]	@Var	N/A	Gets a value based on Class Name.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Account Status|2	@Var	N/A	Gets a value from the Account Status field in the second row and stores the value in a variable.
Opportunity Form Applet - Child|SalesRep2	@Var	N/A	Gets a value from the SalesRep2 field in the form applet and stores the value in a variable.
Opportunity Form Applet - Child|Description2	@Var	N/A	Gets the value from the Description2 field in the form applet and stores the value in a variable.
NULL|Save Query As Applet._SweQueryName	@Var	N/A	Gets a value from the pop-up input field.
SIS Account List Applet|Name	@Var[3]	N/A	Gets a value of first three records in the Accounts list applet.

Mobile Examples

The following table describes how to use the GetValue keyword to obtain the value from a specified object and store the value in a variable in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Name|4	@accountname	N/A	Gets a value from the Name field in the fourth row and stores the value in the account name variable.
SHCE Sales Account List Applet - Mobile|Name	@accountname	N/A	Gets a value from the Name field in the first row and stores the value in the account name variable.

GetValueFromMenuPopup

You use the GetValueFromMenuPopup keyword to read values from application level pop-up menus.

Signature

The GetValueFromMenuPopup keyword supports the following signature:

```
GetValueFromMenuPopup(MenuRN|MenuItemRN,RN_Label(1..N)|Variable(1..N))
```

Desktop Examples

The following table describes how to use the GetValueFromMenuPopup keyword to read values from application level pop-up menus in desktop applications.

Target Object	Inputs	Closing Action	Comments
Help|Help - Technical Support	Application Version|@var1	N/A	Gets the value in the Application Version field when Help - Technical Support is selected from the application-level menu.
Help|Help - About Record	Created By|@var1	N/A	Gets the value in the CreatedBy field when Help, then Technical Support is selected from the application-level menu.

Mobile Examples

The GetValueFromMenuPopup keyword does not apply to mobile applications.

GoToSettings

You use the GoToSettings keyword to view and change the default settings of a user profile.

Signature

The GoToSettings keyword supports the following signature:

```
Gotosettings()
```

Desktop and Mobile Examples

The following table describes how to use the GoToSettings keyword to view and change the default settings of a user profile in desktop and mobile applications.

Target Object	Inputs	Closing Action	Comments
N/A	N/A	N/A	Navigates to the Settings (user profile) screen.

GoToThreadbarView

You use the GoToThreadbarView keyword to move to a view in the threadbar.

Signature

The GoToThreadbarView keyword supports the following signature:

```
GoToThreadbarView(Id of Threadbar Link)
```

Desktop Examples

The following table describes how to use the GoToThreadbarView keyword to move to a view in the threadbar in desktop applications.

Target Object	Inputs	Closing Action	Comments
Views_tb_0	N/A	N/A	Clicks on the specified threadbar link.
Views_tb_1	N/A	N/A	Clicks on the specified threadbar link.

Mobile Examples

The GoToThreadbarView keyword does not apply to mobile applications.

GoToView

You use the GoToView keyword to move to a specified view from the Tab view, Tree view or Site Map view.

Signature (Desktop)

The GoToView keyword supports the following signature on desktop devices:

```
GoToView(ScreenRN|ViewRN|Level)
```

Note the following:

- To go to Sitemap: ScreenRN|View RN|NULL
- To go to ScreenTab: ScreenRN|View RN|L1
- To go to TabView: NULL|ViewRN|L2/L3/L4

Note the following about the GoToView keyword signature:

- View levels can be L1, L2, L3, and L4.
    - L1 means First Level View Bar , Hamburger Menu in Aurora Theme or Top level Tabs as in Tabbed views.
    - L2 refers to the Second Level view bar.
    - L3 refers to the Thrid Level view bar .
    - L4 refers to the Fourth Level view bar .
- When navigating to any view without going via Sitemap , ScreenRN should be NULL and should mention the Level of the destination View (L1/L2/L3/L4)
- Before going to the next level, the user must be in the immediate previous level. For example, if the user has to go to level 3, the user must be in level 2.

Signature (Mobile)

The GoToView keyword supports the following signature on mobile machines:

```
GotoView(ViewRN)
```

Desktop Examples

The following table describes how to use the GoToView keyword to move to a specified view in desktop applications.

Target Object	Inputs	Closing Action	Comments
NULL|Accounts Screen|L1	N/A	N/A	Navigates to the accounts screen, first level view bar.
NULL|Account List View|L2	N/A	N/A	Navigates to the accounts screen, second level view bar.
NULL|TNT SHM Opportunity Agenda View|L3	N/A	N/A	Navigates to the accounts screen, third level view bar.
NULL|Quote Item XA View|L4	N/A	N/A	Navigates to the accounts screen, fourth level view bar.
Sitemap.Asset Management Screen|Sitemap.Asset Mgmt - Assets View|NULL	N/A	N/A	Clicks on the Site Map links.

Mobile Examples

The following table describes how to use the GoToView keyword to move to a specified view in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
Pharma Contact List View - Mobile	N/A	N/A	Navigates to the Contact List view.

HierarchicalList

You use the HierarchicalList keyword to expand and collapse the Hierarchical List applet, including both parent and child records based on the row number provided. You also use the keyword to obtain the total number of child records in the Hierarchical List applet.

Signature

The HierarchicalList keyword supports the following signature:

```
HierarchicalList(AppletRN|Rownum,Expand/Collapse/GetChildItemsCount|Variable)
```

Note the following about the HierarchicalList keyword signature:

- Expand is for expanding the record.
- Collapse is for collapsing the record.
- GetChildItemsCount|@var is for expanding and taking a child count of the record irrespective of whether the record has a child or not.

Desktop Examples

The following table describes how to use the HierarchicalList keyword to expand, collapse and get the child count in desktop applications.

Target Object	Inputs	Closing Action	Comments
Order Entry - Line Item List Applet (Sales)|1	Expand	N/A	Expands the first record.
Order Entry - Line Item List Applet (Sales)|8	Collapse	N/A	Collapses the eighth record.
Order Entry - Line Item List Applet (Sales)|2	GetChildItemsCount|Variable	N/A	Expands the second record and reads the child count in the record.

Mobile Examples

The HierarchicalList keyword does not apply to mobile applications.

InboundWebServiceCall

You use the InboundWebServiceCall keyword to read an XML request from a .xml file, post the request to the server, and save the XML response from the server. The keyword also verifies the expected TagName and value in the XML response.

Signature

The InboundWebServiceCall keyword supports the following signature:

```
Inboundwebservicecall(XMLFIle;Tagname|@VAR1/Value,Tagname2|@var2/Value/ NULL;Tagname|Value/
@var/@STOREvar1,Tagname|value/@var/@STOREvar2)
```

Note the following about the InboundWebServiceCall keyword signature:

- You must provide the full Tagname in the signature.
- The variable name must begin with @STORE to store the response value of a tagname into the variable.

For EAI Webservice calls over HTTPS, Application CA certificate must be imported into Java default truststore (cacerts). Sample command to complete the step:

```
<JAVA_HOME>\bin\keytool -import -file
<PATH_TO_CERTIFICATE_FILE> -alias "<Alias>" -keystore
```

```
"<JAVA_HOME>\lib\security\cacerts"
```

Replace JAVA_HOME with JRE install location (for example: `c:\DISA\jre`) and PATH_TO_CERTIFICATE_FILE should be replaced with CA certificate file location. You may provide a name for the certificate by replacing <Alias>.

XML Structure

To pass the dynamic variable into the XMLFIle, use the following XML structure:

```
<tagname>$var</tagname>
<tagname>="$var"</tagname>
```

For unit run, update EAI section in unitconfig.xml as follows:

```
<EAI>
 <EAI-SERVERNAME>Component Alias</EAI-SERVERNAME>
 <EAI-USERNAME>Username</EAI-USERNAME>
 <EAI-PASSWORD>Password</EAI-PASSWORD>
</EAI>
```

Desktop Examples

The following table describes how to use the InboundWebServiceCall keyword to read, post, and save an XML request and then verify the response in desktop applications.

Target Object	Inputs	Closing Action	Comments
N/A	Prod1.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|@STOREVAR In the Test Step Applet: RequestXMLFile = Prod1.xml TagName to Send=swip:WorkspaceName Variable=@var TagName to Send=swip:WorkspaceReuseFlag Variable=@var2 TagName to Verify=ActiveFlag Value/Variable/@Storevar=@STOREVAR	N/A	Stores the tagname value in the response (ActiveFlag) in the @STOREVAR variable.
N/A	Prod1.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|Y In the Test Step Applet: RequestXMLFile = Prod1.xml TagName to Send=swip:WorkspaceName	N/A	Passes the values to a SOAP request dynamically.

Target Object	Inputs	Closing Action	Comments
	Variable=@var TagName to Send=swip:WorkspaceReuseFlag Variable=@var2 TagName to Verify=ActiveFlag Value/Variable/@Storevar=Y		
N/A	Currency.xml;NULL;ConversionRateResu 0.0161 In the Test Step Applet: RequestXMLFile = Currency.xml TagName to Send=NULL Variable= TagName to Verify=ConversionRateResult Value/Variable/@Storevar=0.0161	N/A	Sends the XML request and verifies the response.
N/A	Currency.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|@var2 In the Test Step Applet: RequestXMLFile = Currency.xml TagName to Send=swip:WorkspaceName Variable=@var TagName to Send=swip:WorkspaceReuseFlag Variable=@var2 TagName to Verify=ActiveFlag Value/Variable/@Storevar=@var2	N/A	Passes the values to a SOAP request dynamically.
N/A	new.xml;NULL;OrderItem|ProductId|88-46TS9:ActionCode|@STOREsdf In the Test Step Applet: RequestXMLFile = new.xml TagName to Send=NULL	N/A	Stores ActionCode from Order item tags in the @STOREsdf variable with the unique productID 88-46TS9.

Target Object	Inputs	Closing Action	Comments
	Variable= TagName to Verify=OrderItem|ProductId|88-46TS9:ActionCode Value/Variable/@Storevar=@STOREsdf		
N/A	InactiveWS.xml;con:FirstName|#FirstName,con:LastName|#LastName;faultstring|@STOREmessageIn the Test Step Applet:RequestXMLFile = InactiveWS.xmlTagName to Send=con:FirstNameVariable/Value=#FirstName(FirstName is the data set column Name) TagName to Send=con:LastNameVariable/Value=#LastName (LastName is the data set column Name) TagName to Verify=faultstringValue/Variable/@Storevar=@STOREmessage	N/A	Illustrates referencing Data Set Values

Mobile Examples

The InboundWebServiceCall keyword does not apply to mobile applications.

InputValue

You use the InputValue keyword to enter a value into a field. The keyword supports unique values by adding the dollar ($) symbol to the end of the text. For example, if the input is john$, then a random unique value appends like "john548".

The InputValue keyword supports the following special characters and operators:

- * (asterisk). For example: like AAX*.
- = (equals). For example: date=Today+1.
- + (plus). For example: Today+1.
- - (minus). For example: Today-1.
- LIKE, like. For example: date format like, LIKE AAX*.
- OR. For example: @var1 OR @var2.
- Numerals zero to nine (0 - 9).

Signature

The InputValue keyword supports the following signature:

```
InputValue(AppletRN|FieldRN|[Active_Record/Tileindex/RowNum],Value OR Variable)
```

Note the following about the HierarchicalList keyword signature:

- RowNumber is optional. If RowNum is not specified, then RowNum defaults to the first row.
- Active_Record obtains the row number of the active record during execution.
- If the action is to be performed on tile applets, then the tile index or row number must start from one.
- !+! is the delimiter used for concatenating dynamic variable in the middle of input data. For more information, see the examples in the following table.

Desktop Examples

The following table describes how to use the InputValue keyword to enter values into fields in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account list Applet|Name	john$	N/A	Creates a unique value when $ is appended to the end of the value.
SIS Account list Applet|Name	Test!+!@storedVar!+!Kwd	N/A	Appends the dynamic variable value(storedVar) in between static text by using the delimiter !+!.
SIS Account list Applet|Name|Active_ Record	john$	N/A	Enters the value in to the field for the highlighted row in the list applet.
Opportunity Form Applet - Child| CloseDate	today+10	N/A	Enters the today+10 value into the CloseDate Field.
SIS Account List Applet|Main Phone Number	LIKE 650	N/A	Enters the phone number LIKE 650*.
NULL|Save Query As Applet._ SweQueryName	Test1234	N/A	Enters the value in to the pop-up input field.
SIS Account list Applet|Name	NULL	N/A	Enters the empty value into the input field (Name).

Mobile Examples

The following table describes how to use the InputValue keyword to enter values into fields in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Account Entry Applet - Mobile|Name	john$	N/A	Creates a unique value when $ is appended to the end of the value.
SHCE Sales Account List Applet - Mobile| Location|2	Paris	N/A	Enters the value into the second row of the list applet. For example, Paris.

InvokeAppletMenuItem

You use the InvokeAppletMenuItem keyword to call a menu item from an applet-level menu in a list or form.

Signature

The InvokeAppletMenuItem keyword supports the following signature:

```
InvokeAppletMenuItem(AppletRN|RN of Menu Item,(RN of Closing Action/NULL))
```

Note the following about the InvokeAppletMenuItem keyword signature:

- The Confirmation dialog box does not close after you delete a record.
- The VerifyError keyword must be used after the deletion of any record.

Desktop Examples

The following table describes how to use the InvokeAppletMenuItem keyword to call a menu item from an applet-level menu in a list or form in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|New Record (SWE)	N/A	NULL	Invokes the New Record applet menu item with no action in the pop-up window that appears.
SIS Account List Applet|Delete Record (SWE)	N/A	NULL	Invokes the Delete Record applet menu item with no action in the pop-up window that appears.
SIS Account List Applet|Record Count (SWE)	N/A	CloseApplet	Invokes the Record Count applet menu item where users must click on OK in the pop-up window that appears.
SIS Account List Applet|About Record (SWE)	N/A	CloseButton	Invokes the About Record applet menu item where users must click on OK in the pop-up window that appears.

Mobile Examples

The following table describes how to use the InvokeAppletMenuItem keyword to call a menu item from an applet-level menu in a list or form in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Record Count (SWE)	N/A	CloseApplet	Invokes the Record Count applet menu item where users must click OK in the pop-up window that appears.

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Get Bookmark URL (SWE)	N/A	CloseApplet	Invokes the Get Bookmark URL applet menu item where users must click OK in the pop-up window that appears.

InvokeMenuBarItem

You use the InvokeMenuBarItem keyword to call a menu item from the application-level menu.

Signature

The InvokeMenuBarItem keyword supports the following signature:

```
InvokeMenuBarItem(MenuRN|MenuItemRN)
```

Desktop Examples

The following table describes how to use the InvokeMenuBarItem keyword to call a menu item from the application-level menu in desktop applications.

Target Object	Inputs	Closing Action	Comments
Menu-Help|Help - Technical Support	IPH1;IPH2 ;IPH3	OK/CANCEL/NULL	Selects Technical Support from the (application-level) Help menu.

Mobile Examples

The InvokeMenuBarItem keyword does not apply to mobile applications.

InvokeObject

You use the InvokeObject keyword to call UI objects such as PickApplet, MVG, Calculator, Currency Image, and so on.

Signature

The InvokeObject keyword supports the following signature:

```
InvokeObject(AppletRN|FieldRN|[RowNum]/[Active_Record])
```

Note: The Active_Record obtains the row number of the active record during the execution.

Desktop Examples

The following table describes how to use the InvokeObject keyword to call UI objects in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Product List Admin Applet|Product Line	N/A	N/A	Invokes the list applet field object.
SIS Product List Admin Applet|Product Line|3	N/A	N/A	Invokes the list applet field object in the third row.
SIS Product List Admin Applet|Product Line|Active_Record	N/A	N/A	Invokes the list applet field object from the current active row.
Opportunity Form Applet - Child|Account2	N/A	N/A	Invokes the form applet field object.
Opportunity Form Applet - Child|Revenue2	N/A	N/A	Invokes a Form applet field object.
NULL|Revenue2	N/A	N/A	Invokes the object in a pop-up window.

Mobile Examples

The following table describes how to use the InvokeObject keyword to call UI objects in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Opportunity List Applet ReadOnly - Mobile|Primary Revenue Amount|1	N/A	N/A	Invokes the Currency pop-up window.
SHCE Sales Opportunity List Applet ReadOnly - Mobile|Account	N/A	N/A	Invokes the drop-down list applet.
NULL|Currency Code	N/A	N/A	Invokes the Currency code dialog box.

InvokeREST

You use the InvokeREST keyword to test and issue REST API calls from within a test script. Using the InvokeREST keyword (for example, to execute the API, verify and/or use the response) reduces the complexity and effort required in automating REST APIs and integrations for automated testing. Note the following:

- For REST API calls over HTTPS, Application CA certificate must be imported into Java's default truststore (cacerts). Sample command to complete the step:

  ```
  <JAVA_HOME>\bin\keytool -import -file <PATH_TO_CERTIFICATE_FILE> -alias "<Alias>" -keystore "<JAVA_HOME>\lib\security\cacerts"
  ```

  Replace `JAVA_HOME` with JRE install location and `PATH_TO_CERTIFICATE_FILE` should be replaced with CA certificate file location(for example: C:\DISA\jre). You may provide a name for the certificate by replacing `<Alias>`.

  For information related to certificate import/truststore, refer to section, *Communications Encryption* in Siebel Security Guide.
- InovkeREST Test Step can be added before a Launch Test Step. The REST URI alias in InvokeREST appears in Test Execution Server Credentials applet as Application Alias.
- Provide REST Server details in the Test Execution record against the REST URI Alias in the "Server Credentials" applet `url: https://servername.com:port number/...`, and specify the password in the Users applet against the correct user id.

  Ensure the Application Type is set to `REST_URI` from drop-down. If the value `REST_URI` is missing, add or activate in **Administration - Data → List of Values** View.
- For a unit run (Play button on Test Script applet), update unitconfig.xml in DISA folders with following attributes and appropriate values within OPTIONAL_PARAM node.

  Example:

  ```
  <APPLICATIONS>
   <APPLICATION>
   <APPLICATION-ALIAS>REST URI Alias</APPLICATION-ALIAS>
   <APPLICATIONTYPE-BROWSER>REST_URI</APPLICATIONTYPE-BROWSER>
   <URL>https://hostname:portname/siebel/v1.0</URL>
   <SERVER-OS-TYPE>Windows</SERVER-OS-TYPE>
   </APPLICATION>
  </APPLICATIONS>
  <USERS>
   <USER>
   <USERID>User_Name</USERID>
   <PASSWORD>Password</PASSWORD>
   </USER>
  </USERS>
  ```

  For multiple user IDs usage in InvokeREST keyword, USER node within USERS can be repeated.

Signature

The InvokeREST keyword supports the following signature:

```
InvokeREST(HTTP Method Name:URL_Alias, UserId|REST_Resource, Input JSON file name | values to be replaced in request body (comma separated); NULL, REST Response code to verify / One variable name or comma separated list of variables.)
```

The following table describes how to use the InvokeREST keyword to test and issue REST API calls from within a test script.

Target Object	Inputs	Closing Action	Comments
GET / POST / PUT / DELETE; <REST URI Alias, userid> | REST Resource	Input JSON file name | values to be replaced in request body (comma separated); NULL	REST Response code to verify / One variable name or comma separated list of variables. **Note:** Variable names must match the field names in the REST response. If the field name contains space, variables name must be enclosed within quotes, for example: @"variable with space"	Issues the REST API call and uses the Response to verify the response code and/or read values into variables.

Pre-defined and Auto-generated Variables for using or validating EAI/REST Response:

Assuming JSON response structure for REST API, Keyword framework provides few pre-defined variables and generates new variables based on the response. It is important to note that the predefined and auto-generated variables (if repeating) are reinitialized or overwritten with subsequent InvokeREST call. Hence, ensure to use the values prior to subsequent InvokeREST call.

1. `@StatusCode`: Typically used in closing action to validate REST Response Code. For example, `StatusCode:200` in closing action will ensure that framework validates the response code to be 200.
2. `@Errormsg`: If provided during closing action, keyword framework can capture the error message for the cases when REST API returns so.
3. `@resp`: The complete response from InvokeREST is captured with this predefined variable.
4. Fetching property values from JSON into Variables: Variables if provided in Closing Action, are matched with the property names of JSON response and their values are updated accordingly. For example, if `@Name` is provided in Closing Action, keyword framework will look-up the top-level JSON response for **Name** property and copy the property value to the variable @Name.

Example 1

InvokeREST keyword Example 1 is shown in the following table.

InvokeREST Keyword	Example
Target Object	`GET;NEW_REST,SADMIN|data/Account/Account/88-347S25/Account Attachment?inlineattachment=True` Method to Invoke: `GET;NEW_REST,SADMIN` REST Resource: `|data/Account/Account/88-347S25/Account Attachment?inlineattachment=True`
Inputs	`NULL;NULL;IPH3` Request Body: NULL Work space name/Any Search Spec: NULL

InvokeREST Keyword	Example
Choice of Closing Action	`@StatusCode:200,@AccntFileSize,@"Accnt Attachment Id"` End Action: `@StatusCode:200,@AccntFileSize,@"Accnt Attachment Id"`

In this example, using the `InvokeREST` keyword, you may make an API request `"data/Account/Account/88-347S25/Account Attachment?inlineattachement=True"` and capture the property AccntFileSize and Acct Attachment Id from the response in variable as `@AccntFileSize` and `@"Accnt Attachment Id"`.

The API request method is **GET** and the REST URL alias is provided using NEW_REST (any name of your choice, but without spaces). In the Test Execution record, **Server Credentials** applet will be populated with NEW_REST and you may update the REST URL appropriately. Please note that the complete REST URL will be formed by concatenating REST URL and REST Resource provided with InvokeREST keyword.

Example 1 Output Response:


```
{
 "AccntFileSize":"89",
 "AccntFileName":"Test5555",
 "Comment":"",
 "Account Id":"88-347S25",
 "Id":"88-347S2Z",
 "AccntFileDate":"09/21/2022
01:54:10",
 "AccntFileDockStatFlg":"E",
 "AccntFileSrcType":"FILE",
 "AccntFileAutoUpdFlg":"Y",
 "AccntFileDockReqFlg":"N",
 "AccntFileExt":"txt",
 "AccntFileDeferFlg":"R",
 "AccntFileSrcPath":"",
 "Accnt Attachment
Id":"QXR0YWNobWVudCBmb3IgZGVtbw==",
 "Link":[
 {
 "rel":"self",
 "href":"https://asdfjkl:16690/siebe
l/v1.0/data/Account/Account/88-347S25/Ac
count Attachment/88-347S2Z",

 "name":"Account Attachment"
 },
 {
 "rel":"canonical",
 "href":"https://asdfjkl:16690/siebe
l/v1.0/data/Account/Account/88-347S25/Ac
count Attachment/88-347S2Z",

 "name":"Account Attachment"
 },
 {

 "rel":"parent",
 "href":"https://asdfjkl:16690/siebe
l/v1.0/data/Account/Account/88-347S25",

 "name":"Account"
 }
 ]
```

```
}
```

Example 2

InvokeREST keyword Example 2 is shown in the following table.

InvokeREST Keyword	Example
Target Object	`PUT;NEW_REST,SADMIN|data/Account/Account/88-347S25/Account Attachment/` Method to Invoke: `PUT;NEW_REST,SADMIN` REST Resource: `data/Account/Account/88-347S25/Account Attachment/`
Inputs	`Attachment.json;NULL;IPH3` Request Body: `Attachment.json` Work space name/Any Search Spec: `NULL`
Choice of Closing Action	`@StatusCode:200,@"items:Account Attachment:Id"` End Action: `@StatusCode:200,@"items:Account Attachment:Id"` **Note:** @"items:Account Attachment:Id" indicates the hierarchy in JSON response i.e., Id is child of **Account Attachment** which in turn is child property of **items**.

Content of `Attachment.json`:

```
{
 "AccntFileName":"Test$,
 "AccntFileExt":"txt",
 "Accnt Attachment Id":"QXR0YWNobWVudCBmb3IgZGVtbw"
}
```

Ensure to place input JSON file (i.e. Attachment,json in this case) in the resources/invokerest folder (ensure same folder structure in your Resources.zip attachment to Master Suite).

Example 2 Output Response:

```
{
 "items":{
 "Id":"88-347S25",
 "Account Attachment":{
 "Id":"88-347S2Z",
 "Accnt Attachment Id":"https://asdfjkl:16690/siebel/v1.0/data/Account/Account/88-347S25/Account
 Attachment/88-347S2Z?fields=Accnt Attachment Id",
 "Link":{
 "rel":"self",
 "href":"https://asdfjkl:16690/siebel/v1.0/data/Account/Account/88-347S25/Account Attachment/88-347S2Z",
 "name":"Account Attachment"
 }
 },
 "Link":{
 "rel":"self",
 "href":"https://asdfjkl:16690/siebel/v1.0/data/Account/Account/88-347S25",
 "name":"Account"
```

```
 }
 }
}
```

Example 3

InvokeREST keyword Example 3 is shown in the following table.

InvokeREST Keyword	Example
Target Object	`POST;NEW_REST,SADMIN|data/Account/Account` Method to Invoke: `POST;NEW_REST,SADMIN` REST Resource: `data/Account/Account`
Inputs	`data_POST.json|InvokeREST_Data_$, @var1; NULL;IPH3` Request Body: `data_POST.json\|InvokeREST_Data_$,@var1` Work space name/Any Search Spec: `NULL`
Choice of Closing Action	`@items:Id,@items:Link:name` End Action: `@items:Id,@items:Link:name`

Content of `data_POST.json`:

```
{
 "Name": "InvokeREST_Data_$",
 "Primary Organization": "Millennium Institutional Finance Services IF ENU",
 "Location": "HQ-Distribution",
 "Description": "@var1",
 "Primary Organization Id": "1-1DG"
}
```

It is important to note that variable support is present with JSON file as well and keyword framework will make attempt to replace content as required. For example, `$` with `InvokeREST_Data_` will be replaced with `timestamp`. Similarly, `@var1` will be replaced appropriately, provided the value is defined.

> **Note:** If the variable name contains space, they should be within quotes appropriately as per JSON format. For example, if the variable name is @"Modified Description", it should be present in JSON file as "Description" : "@ \"Modified Description\""

Example 3 Output Response:

```
{
 "items":{
 "Name":"InvokeREST_Data_10212022_032608182",
 "Id":"88-347S25",
 "Location":"HQ-Distribution",
 "Primary Organization Id":"1-1DG",
 "Primary Organization":"Millennium Institutional Finance Services IF ENU",
 "Description":"AccountData",
 "Link":{
 "rel":"self",
 "href":"https://asdfjkl:16690/siebel/v1.0/data/Account/Account/88-347S25",
```

```
"name":"Account"
}
}
}
```

Example 4

InvokeREST keyword Example 4 is shown in the following table.

InvokeREST Keyword	Example
Target Object	`DELETE;NEW_REST,SADMIN|data/Account/Account/88-347S25/` Method to Invoke: `DELETE;NEW_REST,SADMIN` REST Resource: `data/Account/Account/88-347S25/`
Inputs	`NULL;NULL;IPH3` Request Body: `NULL` Work space name/Any Search Spec: `NULL`
Choice of Closing Action	`@StatusCode:200` End Action: `@StatusCode:200`

Example 4 Output Response:

```
{}
```

Example 5

Using the API /data/Quote/Quote, insert a record in Quotes using the method POST and fetch following Name, Primary Organization, Quote Number and Revision in variables from the response.

InvokeREST Keyword	Example
Target Object	`POST;REST,SADMIN|data/Quote/Quote` Method to Invoke: `POST;REST,SADMIN` REST Resource: `data/Quote/Quote`
Inputs	`Name|QuoteName$,Primary Organization\|ABC,Quote Number\|1234,Revision\|1;NULL;IPH3` Request Body: `Name\|QuoteName$,Primary Organization|ABC,Quote Number|1234,Revision|1` Work space name/Any Search Spec: `NULL`
Choice of Closing Action	`@Name,@"Primary Organization",@"Quote Number",@Revision` End Action: `@Name,@"Primary Organization",@"Quote Number",@Revision`

InvokeREST Keyword	Example

Launch

You use the Launch keyword to start and log in to applications by providing the username.

Signature

The Launch keyword supports the following signature:

```
Launch(component_alias;username;Y/N
```

Note the following about the Launch keyword signature:

- Component_alias must come from the predefined component names for a product.
- Component_alias and username are mandatory parameters.
- [clearBrowser] is an optional parameter, which can be specified if there is a specific requirement to clear cookies at application startup.
- By default, cookies are not cleared at application startup.

Desktop Examples

The following table describes how to use the Launch keyword to start and log in to an application by providing the username in desktop applications.

Target Object	Inputs	Closing Action	Comments
N/A	CORE_UIF;SADMIN;Y	N/A	Starts the browser, clears the browser cookies and logs in with the username SADMIN.
N/A	CORE_UIF;SADMIN;N	N/A	Starts the browser and logs in with the username SADMIN.
N/A	CORE_UIF;PortalApplication (or SSO);Y	N/A	Launches the application URL mapped to Component Alias, but does not interact with the login page. In subsequent Test Step, provide the user credentials through CustomExtension keyword for SSO, or InputValue keyword for Siebel Portal Applications.
N/A	PHARMAM;LaunchApp;Y	N/A	Starts the application.

Mobile Examples

The following table describes how to use the Launch keyword to start and log in to an application by providing the username in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
N/A	PHARMAM;SPORTER	N/A	Starts the application.

LockColumn

You use the LockColumn keyword to lock or unlock a selected column.

Signature

The LockColumn keyword supports the following signature:

```
LockColumn(AppletRN | FieldRN,Lock/Unlock)
```

Note: Lock/Unlock input is case insensitive

Desktop Examples

The following table describes how to use the LockColumn keyword to lock or unlock a selected column in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Name	LOCK	N/A	Locks the selected column.
SIS Account List Applet|Name	UNLOCK	N/A	Unlocks the selected column.

Mobile Examples

The following table describes how to use the LockColumn keyword to lock or unlock a selected column in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
Contact List Applet - Mobile|First Name	LOCK	N/A	Locks the selected column.
Contact List Applet - Mobile|First Name	UNLOCK	N/A	Unlocks the selected column.

LogOut

You use the LogOut keyword to log out from applications.

Signature

The LogOut keyword supports the following signature:

```
LogOut
```

Desktop and Mobile Examples

The following table describes how to use the LogOut keyword to log out from applications in desktop and mobile applications.

Target Object	Inputs	Closing Action	Comments
N/A	N/A	N/A	Logs out from the application.

MafSettings

You use the MafSettings keyword to perform the required action in the MAF Settings page in MAF applications (on iOS and Android devices).

Signature

The signature to start the Siebel app is as follows:

```
Launch(component_alias;LaunchApp;[clearBrowser])
```

The signature for the MafSettings keyword is as follows:

```
MafSettings(Action:Id; Value/NULL; OK/CANCEL/NULL)
```

Note the following about the MafSettings keyword signature:

- Action can be one of the following: Input, Click, Verify, verifyerrorMsg, Flip, or Springboard.
- Value can be a User input value or NULL.
- ClosingAction can be OK, CANCEL, or NULL.

Desktop Examples

The MafSettings keyword does not apply to desktop applications.

Mobile Examples

The following table describes how to use the MafSettings keyword to perform the required action in the MAF Settings page in MAF applications on iOS and Android devices.

Target Object	Inputs	Closing Action	Comments
Click:setting-fragment2:host__inputElement	NULL	NULL	Clicks host.

Target Object	Inputs	Closing Action	Comments
Input:setting-fragment2:host__inputElement	Servername	NULL	Inputs the host value.
verifyerrorMsg:	Alert Invalid Host Server Address	OK	Verifies the alert message.
verify:setting-fragment2:host::lbl	Host;True	NULL	Verifies the host.
flip:sbs-dock__switch	Y	NULL	Flips Enable Show in the dock.
Springboard:Refresh Button	NULL	NULL	Clicks the Springboard toolbar.

MultiSelectRecordsInListApplet

You use the MultiSelectRecordsInListApplet keyword to select multiple records in a list applet.

Signature

The MultiSelectRecordsInListApplet keyword supports the following signature:

```
MultiSelectRecordsinListApplet(AppletRN,RowNum(1..N)
```

Desktop Examples

The following table describes how to use the MultiSelectRecordsInListApplet keyword to select multiple records in list applets in desktop applications.

Target Object	Inputs	Closing Action	Comments
AppletRN	1,3,5	N/A	Selects first, third, and fifth record from the list applet.

Mobile Examples

The following table describes how to use the MultiSelectRecordsInListApplet keyword to select multiple records in list applets in mobile applications on (mobile devices).

Target Object	Inputs	Closing Action	Comments
CG Account List Applet - Mobile	1,3,5	N/A	Selects the first, third, and fifth record from the list applet.

QueryRecord

You use the QueryRecord keyword to query a record in a list or form applet.

Signature

The QueryRecord keyword supports the following signature:

```
QueryRecord(Applet RN|ButtonRN,FieldRN(1..N)|Value(1...N) OR Variable)
```

Note: To query for an existing record, use one or more fields or run an empty query.

Desktop Examples

The following table describes how to use the QueryRecord keyword to query a record in list or form applets in desktop applications.

Target Object	Inputs	Closing Action	Comments
Contact List Applet|NewQuery	Last Name|oracle$,First Name\|company$,Work Phone #|6506123456	N/A	Queries the record with the specified search criteria.
Contact List Applet|NewQuery	LastName| David,FirstName|Albert	N/A	Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|AAAX*	N/A	Queries the record with the specified search criteria.
Contact List Applet|NewQuery	Work Phone #|650678-0987	N/A	Queries the record with the specified search criteria.
Opportunity Form Applet - Child|NewQuery	Opportunity Currency2|USD	N/A	Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|AAAX*	N/A	Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|NULL	N/A	Searches for an empty query (that is for records where First Name is NULL).
Contact List Applet|NewQuery	M/M|Mr.	N/A	Queries the record with the specified search criteria.

Mobile Examples

The following table describes how to use the QueryRecord keyword to query a record in list or form applets in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Opportunity Form Applet - Mobile|NewQuery	Name|John$,Currency|USD,CloseDate|11|12|12	N/A	Queries the record with the specified search criteria.
CG Account List Applet - Mobile|NewQuery	Name|abc	N/A	Queries the record with the specified search criteria.

RemoveFromMvg

You use the RemoveFromMvg keyword to remove a specified record or all records from an MVG list. The "Query:" will be appended to the results after executing the keyword.

Signature

The RemoveFromMvg keyword supports the following signature:

```
RemoveFromMvg(AppletRN|FieldRN|[RowNum]/
[Active_Record],Query:FieldRN(1…N)(1..N)|Value(1…N:);Remove/RemoveAll;OK/
CANCEL,RN(OK)
```

Note the following about the RemoveFromMvg keyword signature:

- Active_Record is the row number of the active record during execution.
- The OK and Cancel options are optional in the input column.

Desktop Examples

The following table describes how to use the RemoveFromMvg keyword to remove a specific or all records from an MVG list in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Sales Rep	Active Last Name|Silver,Active First Name|Victor;Remove	RN of OK	Removes an account team from an account in a list applet.
SIS Account List Applet|Sales Rep|Active_Record	Active Last Name|Silver,Active First Name|Victor;Remove	RN of OK	Removes an account team from an account in a list applet. Performs the action on the field of the active row.
SIS Account Entry Applet|SalesRep	Active Last Name|Dupont,Active First Name|Dupont;Remove	RN of OK	Removes an account team from an account in a form applet.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Sales Rep	NULL;RemoveAll	RN of OK	Removes all accounts in a list applet.
SIS Account List Applet|Sales Rep	NULL;RemoveAll;OK	RN of OK	Removes all accounts in a list applet. Users must click OK on confirmation.

Mobile Examples

The following table describes how to use the RemoveFromMvg keyword to remove a specified or all records from an MVG list in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Primary Account Address Name	Name|abc;RemoveAll	Idcancel	Removes all accounts in a list applet.
SHCE Sales Account List Applet - Mobile|Primary Account Address Name	Name|abc;Remove	Idcancel	Removes an account in a list applet.

SelectCheckBox

You use the SelectCheckBox keyword to select or clear a check box.

Signature

The SelectCheckBox keyword supports the following signature:

```
SelectCheckbox(AppletRN|FieldRN|[RowNum],TRUE(or)FALSE(or)@variable)
```

Desktop Examples

The following table describes how to use the SelectCheckBox keyword to select or clear a check box in desktop applications.

Target Object	Inputs	Closing Action	Comments
Opportunity List Zpplet|Committed	TRUE	N/A	Selects the check box.
SIS Account List Applet|Fund Eligible Flag|3	FALSE	N/A	Clears the check box in the third row of the list applet.
Opportunity List Applet|Committed	@Variable	N/A	Selects or clears the check box depending on the variable value (True or False).
MultiAdd Product On Order|Select All|Select All	True/False	N/A	Selects the check box in the column header.

Target Object	Inputs	Closing Action	Comments

Mobile Examples

The following table describes how to use the SelectCheckBox keyword to select or clear a check box in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Address List Applet - Mobile|SSA Primary Field	TRUE	N/A	Selects the check box for the specified object based on the provided user value.
SHCE Address List Applet - Mobile|SSA Primary Field|3	FALSE	N/A	Clears the check box for the specified object based on the provided user value.

SelectFromMvg

You use the SelectFromMvg keyword to select a specified record after querying the available records in an MVG.

Signature

The SelectFromMvg keyword supports the following signature:

```
SelectFromMvg(AppletRN|FieldRN|[RowNum]/
[Active_Record],FieldRN(1…N)|Value(1…Nb),RN of OK)
```

Note: Active_Record obtains the row number of the active record during execution.

Desktop Examples

The following table describes how to use the SelectFromMvg keyword to select a specified record after querying the available records in an MVG in desktop applications. The Query will be appended to the results after executing the keyword.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Sales Rep	Query:Last Name|Yang	Idok	Selects a Sales Rep after querying the available records in an MVG.
SIS Account List Applet|Sales Rep|Active_Record	Query:Last Name|Yang	Idok	Selects a Sales Rep after querying the available records in an MVG of the active record.

Mobile Examples

The following table describes how to use the SelectFromMvg keyword to select a specified record after querying the available records in an MVG in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Primary Account Address Name	Query:Name|abc	Idcancel	Selects a name after querying the available records in an MVG.

SelectFromPickApplet

You use the SelectFromPickApplet keyword to query and select the first record from a drop-down list applet.

Signature

The SelectFromPickApplet keyword supports the following signature:

```
SelectFromPickApplet(AppletRN|FieldRN|[RowNum]/
[Active_Record],Query:FieldRN(1…N)|Value(1…Nt),RN(OK/CANCEL))
```

Note the following about the SelectFromPickApplet keyword signature:

- RowNum is an optional value. If RowNum is not specified, then RowNum defaults to the first row.
- Active_Record obtains the row number of the active record during execution.

Desktop Examples

The following table describes how to use the SelectFromPickApplet keyword to query and select the first record from a drop-down list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
Opportunity Form Applet - Child|Account2	Query:City|Berkeley	Idcancel	Queries and cancels the first value in a drop-down list applet form.
Account Profile Applet|Price List	Query:Name |ACR764 Price List,Start Date|7/16/2007 05:00:00	PMPopupQueryPick	Queries and selects the first value in a drop-down list applet.
Contact List Applet|Account|1	Query:Name|Hibbing	PopupQueryPick	Queries and selects the first value in a drop-down list applet form.
Contact List Applet|Account|Active_Record	Query:Name|Hibbing	PopupQueryPick	Queries and selects the first value in a drop-down list applet (in active record).
NULL|Currency Code	Query:Currency Code|USD	PopupQueryPick	Queries and selects the first value in a drop-down list applet form.

Mobile Examples

The following table describes how to use the SelectFromPickApplet keyword to query and select the first record from a drop-down list applet in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Parent Account Name	Query:Name|abc	PopupQueryPick	Queries and selects the first value in a drop-down list applet form.
SHCE Sales Contact List Applet - Mobile|NewRecord	Query:Last Name|abc	AddRecord	Queries and selects the first value in a drop-down list applet form (having clicked the plus (+) icon in the list applet).

SelectPDQValue

You use the SelectPDQValue keyword to select a value from the Predefined Query (PDQ) list in the application.

Signature

The SelectPDQValue keyword supports the following signature:

```
SelectPDQValue(ItemRN OR Variable)
```

Desktop Examples

The following table describes how to use the SelectPDQValue keyword to select a value from the PDQ list in desktop applications.

Target Object	Inputs	Closing Action	Comments
Account	N/A	N/A	Selects a PDQ value named <abc>.
@Variable	N/A	N/A	Selects a PDQ value stored in a variable.

Mobile Examples

The following table describes how to use the SelectPDQValue keyword to select a value from the PDQ list in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
Account	N/A	N/A	Selects a PDQ value named <abc>.
@Variable	N/A	N/A	Selects a PDQ value stored in a variable.

Target Object	Inputs	Closing Action	Comments

SelectPickListValue

You use the SelectPicklistValue keyword to select a value from the drop-down list in a form or list applet.

Signature

The SelectPicklistValue keyword supports the following signature:

```
SelectPicklistValue(AppletRN|FieldRN|[RowNum]/[Active_Record],Value(or)Variable)
```

Note: Active_Record obtains the row number of the active record during execution.

Desktop Examples

The following table describes how to use the SelectPicklistValue keyword to select a value from the drop-down list in a form or list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Account Status|2	Active	N/A	Sets a value in a drop-down list of a particular field in the second row of the list applet.
SIS Account List Applet|Account Status|Active_Record	Active	N/A	Sets a value in a drop-down list of a particular field in the active row of the list applet.
SIS Account List Applet|Account Status	@Variable	N/A	Sets a value (by using a variable) in a drop-down list of a particular field in a list applet.
Opportunity Home Search Virtual Form Applet|SalesStage	@Variable	N/A	Sets a value (by using a variable) in a drop-down list of a particular field in a form applet.

Mobile Examples

The following table describes how to use the SelectPicklistValue keyword to select a value from a drop-down list in a form or list applet in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Account Entry Applet - Mobile|Type	Customer	N/A	Selects a value from a drop-down list or LOV (List of Values) in a form or list applet.

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Type	@var	N/A	Sets a value in a drop-down list of a particular field in the second row of a list applet.
SHCE Sales Account List Applet - Mobile|Type|1	@var	N/A	Sets a value (by using a variable) in a drop-down list of a particular field in a form applet.

SelectRadioButton

You use the SelectRadioButton keyword to select a radio button.

Signature

The SelectRadioButton keyword supports the following signatures:

```
SelectRadioButton(AppletRN|FieldRN ,Value OR Variable)
SelectRadioButton(NULL|FieldRN ,Value)
SelectRadioButton(AppletRN|FieldRN ,Variable)
```

Note: Value must not be the UI name. Use the DOM attribute as the name Value.

Desktop Examples

The following table describes how to use the SelectRadioButton keyword to select a radio button in desktop applications.

Target Object	Inputs	Closing Action	Comments
Sort Order Popup Applet (SWE)| rdbDesc1	Ascending	N/A	Selects a specified radio button.
NULL|SWE Export Applet.rdbRowsToExport	All Rows In Current Query	N/A	Selects a specified radio button.
Sort Order Popup Applet (SWE)| rdbDesc1	@Variable	N/A	Selects a variable value radio button.

Mobile Examples

The SelectRadioButton keyword does not apply to mobile applications.

SelectRecordInListApplet

You use the SelectRecordInListApplet to select a particular record in a list applet.

Signature

The SelectRecordInListApplet keyword supports the following signature:

```
SelectRecordinListApplet(AppletRN|FieldRN|[RowNum/Tileindex],Value/Variable/NULL)
```

Note: If the action is to be performed on tile applets, then the tile index and row number must start from one.

Desktop Examples

The following table describes how to use the SelectRecordInListApplet keyword to select a particular record in a list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Name	Metropolitan Investments	N/A	Selects the specified field value record in the list applet.
AppletSIS Account List Applet|Name|5	NULL	N/A	Selects a record (the fifth record) based on the specified row index (5).
AppletSIS Account List Applet|Name|5	NULL	N/A	Selects the fifth record in the list applet.
AppletQuote List Applet|Quote Number	@Variable	N/A	Selects the specified field value from a variable record in the list applet.
AppletSIS Account List Applet|Name|4	Abc	N/A	Selects the specified row (or fourth record) in the list applet.

Mobile Examples

The following table describes how to use the SelectRecordInListApplet keyword to select a particular record in a list applet in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
AppletRN|FieldRN|[RowNumber]	Value	N/A	Selects a record in the list applet, according to the specified input value.
AppletSHCE Sales Account List Applet - Mobile|Name|1	FinanceOne Corporation	N/A	Selects the specified value in the specified row number.

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Name	FinanceOne Corporation	N/A	Matches the value in the visible records.
AppletSHCE Sales Account List Applet - Mobile|Name|1	@accountName	N/A	Matches the field value in a variable record.

SelectToggleValue

You use the SelectToggleValue keyword to select a value from a toggle control in a list applet.

Signature

The SelectToggleValue keyword supports the following signature:

```
SelectToggleValue(AppletRN|ToggleRN,Value)
```

Desktop Examples

The following table describes how to use the SelectToggleValue keyword to select a value from a toggle control in a list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|SiebToggle	Accounts	N/A	Selects the specified value from the toggle control in a list applet.
Pharma Calendar Activity List Applet|SiebToggle	1	N/A	Selects the specified value from the toggle control in a list applet. **Note:** Input "1" represents the "value" attribute in DOM

Mobile Examples

The SelectToggleValue keyword does not apply to mobile applications.

SelectVisibilityFilterValue

You use the SelectVisibilityFilterValue keyword to select a value from the Visibility Filter drop-down list in an applet.

Signature

The SelectVisibilityFilterValue keyword supports the following signature:

```
SelectVisibilityFilterValue(AppletRN|ItemRN)
```

Desktop Examples

The following table describes how to use the SelectVisibilityFilterValue keyword to select a value from the Visibility Filter drop-down list in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|All Account List View	N/A	N/A	Selects the All Account list view from the Visibility Filter drop-down list.
Contact List Applet|Manager's Contact List View	N/A	N/A	Selects the Manager's Contact list view from the Visibility Filter drop-down list.

Mobile Examples

The SelectVisibilityFilterValue keyword does not apply to mobile applications.

SendKeys

You use the SendKeys keyword to trigger keyboard events.

Signature

The SendKeys keyword supports the following signature:

```
SendKeys(Keystroke)
```

Note the following about the SendKeys keyword signature:

- You must use SendKeys only if the test case requires an action to be performed using the keyboard.
- You must open SendKeys in Inetpub.
- You must enable screenshot for test steps when using SendKeys for debugging.
- Before performing a SendKeys action, the focus must be on the object.
- Avoid using the keyboard (including SendKeys) during unit testing of the test cases.
- During batch execution, the computer window in which execution occurs must be closed, to avoid sending false key input.
- Special characters keys are supported, including combinations of the following:
  - Lowercase a to z.
  - Uppercase A to Z.
  - Numerals 1 to 9.

- The following keyboard keys are supported:

  ADD; ALT; ARROW_UP; ARROW_DOWN; ARROW_LEFT; ARROW_RIGHT; BACK_SPACE; CANCEL; CLEAR; COMMAND; CONTROL; DELETE; DECIMAL; DIVIDE; DOWN; END; ENTER; EQUALS; ESCAPE; F1; F2; F3; F4; F5; F6; F7; F8; F9; F10; F11; F12; HELP; HOME; INSERT; MULTIPLY; NUMPAD0; NUMPAD1; NUMPAD2; NUMPAD3; NUMPAD4; NUMPAD5; NUMPAD6; NUMPAD7; NUMPAD8; NUMPAD9; PAGE_UP; PAGE_DOWN; SHIFT LEFT; SHIFT RIGHT; SPACE; SUBTRACT; TAB; UP; OPENING_BRACE; CLOSING_BRACE.

Desktop Examples

The following table describes how to use the SendKeys keyword to trigger keyboard events in desktop applications.

Target Object	Inputs	Closing Action	Comments
CTRL+ALT+N	N/A	N/A	Selects all records.
CTRL+S	N/A	N/A	Saves.
CTRL+TAB	N/A	N/A	Tabs out.

SetDateTime

You use the SetDateTime keyword to call the DateTime or Date pop-up calendar specify the date and time.

Signature

The SetDateTime keyword supports the following signature:

```
SetDateTime(AppletRN|FieldRN|[RowNum]/[Active_Record],today(+/-))
```

Note the following about the SendDateTime keyword signature:

- Active_Record obtains the row number of the active record during execution.
- If the time is not specified, then the default time is 00:00:00. Value can be a DATETIME, a DATE, or a Variable.

Desktop Examples

The following table describes how to use the SetDateTime keyword to call the DateTime or Date pop-up calendar to provide the date and time in desktop applications.

Target Object	Inputs	Closing Action	Comments
Opportunity Form Applet - Child|CloseDate2	DD:MM:YYYY;01:01:2015	N/A	Chooses a date in the Calendar pop-up window.
Activity List Applet With Navigation|Planned	Today	N/A	Sets the current date and time.

Target Object	Inputs	Closing Action	Comments
Activity List Applet With Navigation|Planned|Active_Record	Today	N/A	Sets the current date and time in the field of the active record.
Activity List Applet With Navigation|Planned	DD:MM:YYYY:HH:MM:SS;05:08:2014 : 10:17:20	N/A	Sets the specified date and time.
Activity List Applet With Navigation|Planned	DD:MM:YYYY:HH:MM:SS;05:08:2014 : 10:17:20	N/A	Sets the specified date and time.
Opportunity Form Applet - Child|CloseDate2	Today+365	N/A	Sets the specified date plus (+)365 days.
Opportunity Form Applet - Child|CloseDate2	Today-365	N/A	Sets the specified date minus (-)365 days.
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR	N/A	Sets a specified date in the current year.
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR-1	N/A	Sets the specified date in the previous year.
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR+2	N/A	Sets the specified date in the current year plus (+)2.

Mobile Examples

The following table describes how to use the SetDateTime keyword to call the DateTime or Date pop-up calendar to provide the date and time in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Contact Opportunity List Applet - Mobile|Primary Revenue Close Date	Today	N/A	Sets the current date and time.
SHCE Sales Opportunity Quote List Applet ReadOnly - Mobile|Start Date	Today+9	N/A	Sets the current date and time.
SHCE Quote Entry Applet - Mobile|StartDate	DD:MM:YYYY;11:05:2014	N/A	Sets the specified date in the target object.
SHCE Sales Order Entry List Applet - Mobile|Order Date	DD:MM:YYYY:HH:MM:SS; 11:5:2014:12:12:30	N/A	Sets the specified date and time in the target object.

Target Object	Inputs	Closing Action	Comments
SHCE Sales Order Entry List Applet - Mobile|Order Date	DD:MM:YYYY:HH:MM:SS; 1:8:2014:12:30:20	N/A	Sets the specified date in the target object.

SortColumn

You use the SortColumn keyword to sort a selected column.

Signature

The SortColumn keyword supports the following signature:

```
SortColumn(AppletRN | FieldRN,ASC/DESC)
```

Note: The SortColumn input is case insensitive.

Desktop Examples

The following table describes how to use the SortColumn keyword to sort a selected column in desktop applications.

Target Object	Inputs	Closing Action	Comments
Opportunity List Applet|Primary Revenue Amount	ASC	N/A	Sorts the columns in ascending order.
SIS Account List Applet|Type	DESC	N/A	Sorts the columns in descending order.

Mobile Examples

The following table describes how to use the SortColumn keyword to sort a selected column in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Account Team List Applet - Mobile| Active Last Name	ASC	N/A	Sorts the columns in ascending order.
SHCE Sales Contact List Applet - Mobile| First Name	DESC	N/A	Sorts the columns in descending order.

StartLoop

Insert a Test Step in your Test Script at a point where the looping needs to start. Optionally, you can provide a variable name as parameter. This variable would be initialized to 1 on first instance of StartLoop, and then incremented for each iteration. You may use this variable for the iteration number.

Refer to section *StartLoop and EndLoop Logical Looping Constructs* for usage details.

EndLoop

Insert a Test Step in your Test Script at a point where the looping needs to end. Optionally, you can provide a condition expression or a max iteration number, or both, as parameters.

Refer to section *StartLoop and EndLoop Logical Looping Constructs* for usage details.

SwitchTab

You use the SwitchTab keyword to switch between browser tabs.

Signature

The SwitchTab keyword supports the following signature:

```
SwitchTab(id/WindowName,URL/NULL)
```

Desktop Examples

The following table describes how to use the SwitchTab keyword in desktop applications to switch between browser tabs.

Target Object	Inputs	Closing Action	Comments
N/A	Account;@URL;IPH3	N/A	Switches to the new tab with the specified URL.
N/A	1;NULL;IPH3	N/A	Focuses and navigates to the first tab.
N/A	Account;NULL;IPH3	N/A	Focuses and navigates to the Account tab.

Mobile Examples

The following table describes how to use the Wait keyword in mobile applications.

Target Object	Inputs	Closing Action	Comments
N/A	Account;@URL;IPH3	N/A	Switches to the new tab with the specified URL.
N/A	1;NULL;IPH3	N/A	Focuses and navigates to the first tab.
N/A	Account;NULL;IPH3	N/A	Focuses and navigates to the Account tab.

TreeExplorer

You use the TreeExplorer keyword to perform operations in the explorer tree applet; operations such as to expand and collapse the tree, select items in the tree, and show the child items.

Signature

The TreeExplorer keyword supports the following signature:

```
TreeExplorer(TreeAppletRN|Tree_id,Expand/Collapse/SelectTreeItem/
GetTreeChildItemsCount|@var/IsNodeExists[;@Var;True/False;GetTreeChildItemsCount|@var])
```

Desktop Examples

The following table describes how to use the TreeExplorer keyword to expand, collapse, select items from, and show the child items under an explorer tree in desktop applications.

Target Object	Inputs	Closing Action	Comments
Account Tree Applet|1.9	Expand	N/A	Expands the tree structure.
Account Tree Applet|1.9	Collapse	N/A	Collapses the tree structure.
Account Tree Applet|1.9	IsNodeExists	N/A	Verifies the existence of a node.
Account Tree Applet|1.9	IsNodeExists;@Var;True/False	N/A	Verifies the existence of a node in the case of a negative scenario.
Account Tree Applet|1.9	SelectTreeItem	N/A	Selects the specified tree item.
Account Tree Applet|1.2	GetTreeChildItemsCount|@var	N/A	Obtains the number of child items and saves the value in a variable.

Mobile Examples

The TreeExplorer keyword does not apply to mobile applications.

VerifyColumnLockStatus

You use the VerifyColumnLockStatus keyword to verify the lock status of a column.

Signature

The VerifyColumnLockStatus keyword supports the following signature:

```
VerifyColumnLockStatus(AppletRN | FieldRN,Lock/Unlock
```

Note: The VerifyColumnLockStatus input is case insensitive.

Desktop Examples

The following table describes how to use the VerifyColumnLockStatus keyword to verify the lock status of a column in desktop applications.

Target Object	Inputs	Closing Action	Comments
LockStatusSIS Account List Applet|Name	LOCK	N/A	Verifies the lock status of the specified column.
LockStatusSIS Account List Applet|Name	UNLOCK	N/A	Verifies the unlock status of the specified column.

Mobile Examples

The following table describes how to use the VerifyColumnLockStatus keyword to verify the lock status of a column in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
LockStatusContact List Applet - Mobile|First Name	LOCK	N/A	Verifies the lock status of the specified column.
LockStatusContact List Applet - Mobile|First Name	UNLOCK	N/A	Verifies the unlock status of the specified column.

VerifyColumnSortOrder

You use the VerifyColumnSortOrder keyword to verify the order of records in a selected column.

Signature

The VerifyColumnSortOrder keyword supports the following signature:

```
VerifyColumnSortOrder(AppletRN | FieldRN,ASC/DESC)
```

Desktop Examples

The following table describes how to use the VerifyColumnSortOrder keyword to verify the order of records in a selected column in desktop applications.

Target Object	Inputs	Closing Action	Comments
Opportunity List Applet|Primary Revenue Amount	ASC	N/A	Verifies that records are sorted in ascending order.
SIS Account List Applet|Type	DESC	N/A	Verifies that records are sorted in descending order.

Mobile Examples

The following table describes how to use the VerifyColumnSortOrder keyword to verify the order of records in a selected column in mobile applications.

Target Object	Inputs	Closing Action	Comments
SHCE Account Team List Applet - Mobile|Active Last Name	ASC	N/A	Verifies that records are sorted in ascending order.
SHCE Sales Contact List Applet - Mobile|First Name	DESC	N/A	Verifies that records are sorted in descending order.

VerifyError

You use the VerifyError keyword to verify the error messages that appear in applications i.e., Siebel error pop ups'.

Signature

The VerifyError keyword supports the following signature:

```
VerifyError(ExpectedMessageSubstring1|ExpectedMessageSubstring2|…..ExpectedMessage
SubstringN,OK/CANCEL)
```

Note: Clicking on OK in the pop-up window is an implicit action.

Desktop Examples

The following table describes how to use the VerifyError keyword to verify the error messages that appear in applications in desktop applications.

Target Object	Inputs	Closing Action	Comments
N/A	'Duration' is a required field|Please enter a value for the field.|SBL-DAT-00498	OK	Verifies whether the error message, which appears in the application, contains the specified string value or not. (The Pipe separator is used a separator to verify multiple strings.)
N/A	'Duration' is a required field. Please enter a value for the field.(SBL-DAT-00498)	OK	To verify if the complete error message is displayed correctly.

Mobile Examples

Target Object	Inputs	Closing Action	Comments
N/A	'Duration' is a required field|Please enter a value for the field.|SBL-DAT-00498.	OK	Verifies whether the error message, which appears in the application, contains the specified string value or not.

VerifyFileLoad

You use the VerifyFileLoad keyword to perform image validation, such as checking that the image has downloaded completely (100% download) and the image filename.

Signature

The VerifyFileLoad keyword supports the following signature:

```
VerifyFileLoad(AppletRN|ImageFileName|TilePosition,Y/N [Full Download])
```

Note: The TilePosition index must start at one.

Desktop Examples

The following table describes how to use the VerifyFileLoad keyword to perform image validation in desktop applications.

Target Object	Inputs	Closing Action	Comments
eDetailer Messaging Plan Items Preview List Applet - Mobile|ThumbnailFN3_01132017.jpg|1	Y	N/A	Verifies that the image has downloaded completely.

Target Object	Inputs	Closing Action	Comments
eDetailer Messaging Plan Items Preview List Applet - Mobile|ThumbnailFN3_01132017.jpg	Y	N/A	Verifies that the image has downloaded completely.

Mobile Examples

Verifyfileload eDetailer Messaging Plan Items Preview List Applet - Mobile|ThumbnailFN3_01132017.jpg|1 Y

VerifyFocus

You use the VerifyFocus keyword to verify the focus location in an application. Focus can be on one of the following in an application: list, form, view, field, or rows in an applet.

Signature

The VerifyFocus keyword supports the following signature:

```
VerifyFocus(AppletRN(or)ViewRN|FieldRN,TRUE/FALSE)
```

Note the following about the VerifyFocus keyword signature:

- The True and False input covers the following scenarios:
- Use False to check that focus is not on a particular applet, field, or row.
- Use True to check that focus is on a particular applet, field, row, application-level menu, or application-level menu item.

Desktop Examples

The following table describes how to use the VerifyFocus keyword to verify the focus location in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|NULL	TRUE	N/A	Verifies that focus is on the specified list applet.
SIS Account List Applet|NULL	FALSE	N/A	Verifies that focus is not on the specified list applet.
SIS Account List Applet|Row Status|2	TRUE	N/A	Verifies that focus is on the specified field in the list applet.
SIS Account Entry Applet|Name	TRUE	N/A	Verifies that focus is on the specified field in the form applet.
SIS Account Entry Applet|Name	FALSE	N/A	Verifies that focus is not on the specified field in the form applet.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|2	TRUE	N/A	Verifies that focus is on the specified row number in the list applet.
SIS Account List Applet|2	FALSE	N/A	Verifies that focus is not on the specified row number in the list applet.
SIS Account List Applet|2,3	TRUE	N/A	Verifies that focus is on the specified row numbers in the list applet.
Search Admin|NULL	TRUE	N/A	Verifies that focus is on the specified view.
Menu-File|Null	TRUE	N/A	Verifies that focus is on the application-level menu.
Menu-File|Null	FALSE	N/A	Verifies that focus on the application-level menu.
Menu-File|File - Custom Print	TRUE	N/A	Verifies that focus is on the application-level menu item.
Menu-File|File - Custom Print	FALSE	N/A	Verifies that focus is on the application-level menu item.

Mobile Examples

The following table describes how to use the VerifyFocus keyword to verify the focus location in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Account Entry Applet - Mobile|Name	TRUE	N/A	Verifies that focus is on the specified field in the form applet.
SHCE Sales Account List Applet - Mobile|NULL	TRUE	N/A	Verifies that focus is on the specified list applet.
SHCE Sales Account List Applet - Mobile|Name|1	TRUE	N/A	Verifies that focus is on the specified field in the list applet.
SHCE Sales Account List Applet - Mobile|1	TRUE		Verifies that focus is on the specified row number in the list applet.

VerifyInPicklist

You use the VerifyInPicklist keyword to count the number of items in a drop-down list, auto select using the substring, and verify whether the values exist or not in the drop-down list without selecting the values.

Signature

The VerifyInPicklist keyword supports the following signature:

```
VerifyInPicklist(AppletRN|FieldRN|[RowNum],Count:operator,value(or) Variable;True/False
[AutoSelect]: [Exists]: value or variable; True/false)
```

Desktop Examples

The following table describes how to use the VerifyInPicklist keyword to verify the values in a drop-down list in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Type|1	Count;=;118|True	N/A	Verifies that all values (118) are present in the Type drop-down list.
SIS Account List Applet|Type|1	AutoSelect;contains;ac|TRUE	N/A	Verifies that there are values that contain the keyword ac.
SIS Account List Applet|Type|1	Exists;=;Customer123|True	N/A	Verifies whether a value exists or not.
Opportunity Form Applet - Child| SalesStage2	AutoSelect;starts;App|TRUE	N/A	Verifies that there are values that start with App.
SIS Account List Applet|Type|1	Exists;=;@var|True	N/A	Verifies whether a variable value exists or not.

Mobile Examples

Verifies the color of the picklist value.

```
Cfg Cx Runtime Instance Frame (JS HI)|RF1060
_Accessory_DOMAINSELECT
color:RED;=;RF1060_Packof10Zip Disks|TRUE
Packof10Zip Disks|TRUE
```

Verifies that the items in the Accessories pick list are showing up in the specified color (which is RED in this example).

VerifyObject

You use the VerifyObject keyword to verify the presence of an object or UI name in applications.

Signature

The VerifyObject keyword supports the following signature:

```
VerifyObject(AppletRN|FieldRN/MenuItemRN,UN name/data-caption of MenuItem/Inner
Text or Title for field items/NULL; TRUE/FALSE)
```

Note the following in the VerifyObject keyword signature:

- Use NULL as the input to verify the existence of an object
- Use the object label (for example, the name of a button) as the input to verify the object name.
- Use the menu item label as the input to verify the menu item name.
- Use the field label as the input to verify the field name or title.
- True or False input is mandatory.
- If the True parameter is set and if the expected object does not match the actual object in the UI, then the test step fails.
- If the True parameter is set and if the expected object matches the actual object in the UI, then the test step passes.
- If the False parameter is set and if the expected object does not match the actual object in the UI, then the test step passes.
- If the False parameter is set and if the expected object matches the actual object in the UI, then test the step fails.

Note: True/False is case insensitive.

Desktop Examples

The following table describes how to use the VerifyObject keyword to verify the presence of an object or UI name in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|NULL	NULL;TRUE	N/A	Verifies the existence of an applet.
NULL|SUI_OPEN_TOOLBAR	NULL;TRUE	N/A	Verifies that the toolbar is open.
NULL|SUI_CLOSED_TOOLBAR	NULL;TRUE	N/A	Verifies that the toolbar is open.
SIS Account List Applet|SiebVisList	All Accounts Across Organizations;TRUE	N/A	Verifies that the item exists in the Visibility list in the list applet.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|DeleteRecord	NULL;TRUE	N/A	Verifies that the Delete button exists in the list applet.
SIS Account List Applet|DeleteRecord	Delete;TRUE	N/A	Verifies that the Delete button UI name exists in the list applet.
SIS Account Entry Applet|DeleteRecord	NULL;TRUE	N/A	Verifies that the Delete button exists in the form applet.
SIS Account List Applet|NewQuery	NULL;TRUE	N/A	Checks that the NewQuery button exists in the list applet.
SIS Account List Applet|Type	Account Type;TRUE	N/A	Checks that the Column Account Type exists in the list applet.
SIS Account List Applet|QueryComboBox	NULL;TRUE	N/A	Verifies that the QueryComboBox drop-down list exists in the list applet.
SIS Account List Applet|About Record (SWE)	About Record [Ctrl+Alt +K];TRUE	N/A	Verifies that the menu item About Record (SWE) exists in the list applet.
SIS Account List Applet|Account Type Code|1	Customer;TRUE	N/A	Checks the title of the Account Type Code field with the row number in the list applet.
Menu-File|File - Create Bookmark	Create Bookmark...;TRUE	N/A	Verifies that the application menu item Create Bookmark exists in the file menu.
NULL|SWE Export Applet.rdbRowsToExport	All Rows In Current Query;TRUE	N/A	Verifies that the Radio button exists in the pop-up.
NULL|Account List View|L2	Accounts List;TRUE	N/A	Verifies that the application links exist.
SIS Account List Applet|DeleteRecord	Delete;FALSE	N/A	Verifies the absence of the Delete button UI name in the list applet (negative scenario).

Mobile Examples

The following table describes how to use the VerifyObject keyword to verify the presence of an object or UI name in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
AppletRN|FieldRN	FieldUN;TRUE/FALSE	N/A	Verifies the UI name of the field.

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|NewQuery	Query;TRUE	N/A	Verifies that the Query button exists in the list applet.
SHCE Sales Account List Applet - Mobile|QuerySrchSpec	Accounts;TRUE	N/A	Verifies that items exist in the QuerySearch text box in the list applet.
SHCE Sales Account List Applet - Mobile|DeleteRecord	Delete;TRUE	N/A	Verifies that the Delete button UI name exists in the list applet.
SHCE Sales Account List Applet - Mobile|QueryComboBox	NULL;TRUE	N/A	Verifies that the QueryComboBox drop-down list exists in the list applet.
SHCE Sales Account List Applet - Mobile|Type|1	Customer;TRUE	N/A	Checks the title of the Account Type Code field with the row number in the list applet.
NULL|Account List View	Accounts List;TRUE	N/A	Verifies that the application links exist.

VerifyRecordCount

You use the VerifyRecordCount keyword to verify the number of records (the record count) in a list applet.

Signature

The VerifyRecordCount keyword supports the following signature:

```
VerifyRecordCount(AppletRN|Rn of record count from the list/Form applet menu/
NULL,Operator,Value(or)@Variable)
```

Note the following about the VerifyRecordCount keyword signature:

- The following operators are supported: greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=), equals (<>).
- If using NULL, then the record count is verified in a pop-up window or applet that does not have an applet menu.

Desktop Examples

The following table describes how to use the VerifyRecordCount keyword to verify the number of records in a list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Record Count (SWE)	>=,6	N/A	Verifies if the count is greater than or equal to 6.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Record Count (SWE)	<=,6	N/A	Verifies if record count is less than or equal to 6.
SIS Account List Applet|Record Count (SWE)	<>,6	N/A	Verifies if the record count with less than or greater than 6.
SIS Account List Applet|Record Count (SWE)	< ,6	N/A	Verifies the record count is less than 6.
SIS Account List Applet|Record Count (SWE)	> ,6	N/A	Verifies if the record count is greater than 6.
SIS Account List Applet|Record Count (SWE)	> =,@var	N/A	Verifies if the record count is greater than or equal to any variable.
SIS Account List Applet|NULL	> =,2	N/A	Verifies if the record count is greater than or equal to 2.

Mobile Examples

The following table describes how to use the VerifyRecordCount keyword to verify the number of records in a list applet in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Record Count (SWE)	>,6	N/A	Verifies if the record count is greater than or equal to 6.
SHCE Sales Account List Applet - Mobile|Record Count (SWE)	<,@var	N/A	Verifies if the record count is less than any variable.
SHCE Sales Account List Applet - Mobile|NULL	<,2	N/A	Verifies if the record count is less than 2.

VerifyState

You use the VerifyState keyword to verify the state of an object.

Signature

The VerifyState keyword supports the following signatures:

```
VerifyState(AppletRN|FieldRN|[RowNum], TRUE/FALSE)
VerifyState(AppletRN|MenuButtonRN|MenuItemRN, TRUE)
```

```
VerifyState(ApplicationLevelMenuRN|ApplicationLevelMenuItemRN, TRUE)
```

Note the following about the VerifyState keyword signature:

- If an object is enabled, editable, or drillable, then verification is True.
- If an object is editable and drillable, then verification is True.
- If an object is read-only and drillable, then verification is False
- If an object is read-only or disabled, then verification is False.
- You can verify the state of most objects, including the following: button, text, check box, drop-down list, menu item, application-level menu items.

Desktop Examples

The following table describes how to use the VerifyState keyword to verify the state of an object in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Product Form Applet - ISS Admin|NewRecord	TRUE	N/A	Verifies whether the button state is enabled or disabled.
SIS Product List Admin Applet|Billable Flag|3	FALSE	N/A	Verifies whether the check box field is enabled or disabled.
SIS Product Form Applet - ISS Admin|SiebAppletMenu|New Record (SWE)	TRUE	N/A	Verifies the state of menu item in the form applet.
Menu-Query|Query - QueryAssist	FALSE	N/A	Verifies the state of application level menu items.
NULL|Tree	TRUE	N/A	Verifies the state of the application.
NULL|Tab	TRUE	N/A	Verifies the state of the application.
NULL|Side Menu	TRUE	N/A	Verifies the state of the application.

Mobile Examples

The following table describes how to use the VerifyState keyword to verify the state of an object in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|QuerySrchSpec	TRUE	N/A	Verifies the state of the variable and stores the value in @var.
SHCE Sales Account List Applet - Mobile|SiebAppletMenu|Record Count (SWE)	TRUE	N/A	Verifies the state of the menu item.

Target Object	Inputs	Closing Action	Comments

VerifyTopNotification

You use the VerifyTopNotification keyword to verify whether or not a notification message appears in the application. The keyword also verifies the color of the notification.

Signature

The VerifyTopNotification keyword supports the following signature:

```
VerifyTopNotification((MessagebroadcastRN,Expectedmessage|[ExpectedHeader];
Color|Expectedd Color/NULL|Match/NoMatch,Close/KeepOpen))
```

Note: You must click the Marl All as Read option before using the click operation on any notification message. ClickTopNotification checks for the message in the notifications list for up to ten iterations (with an interval of one minute between iterations).

Desktop Examples

The following table describes how to use the VerifyTopNotification keyword to verify whether or not a notification message appears in desktop applications.

Target Object	Inputs	Closing Action	Comments
MsgBrdCstIcon	Account_10142015_041 155918;NULL;Match	Close	Verifies whether or not the first unread message appears in the notifications list, and closes the control.
MsgBrdCstIcon	Account_10142015_041 155918;NULL;NoMatch	Close	Verifies whether or not the first unread message appears in the notifications list, and closes the control.
MsgBrdCstIcon	Account_10142015_041 155918;Color|Red;Match	Close	Verifies whether the color of the first unread message in the notifications list matches the input value (Color|Red in this case), and closes the control.
MsgBrdCstIcon	Account_10142015_041 155918;Color|Red;Match	KeepOpen	Verifies whether the color of the first unread message in the notifications list matches the input value (Color|Red in this case), and keeps the control open.
MsgBrdCstIcon	SVP Action|populate Actions;NULL;Match	Close	Verifies whether the first unread message in the notifications list matches the input value, and closes the control.

Mobile Examples

The following table describes how to use the VerifyTopNotification keyword to verify whether or not a notification message appears in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
MsgBrdCstIcon	Data synchronization notification|Data is ready for download;NULL;Match	Close	Verifies whether or not the first unread message appears in the notifications list, before going offline and closing the control.

VerifyValue

You use the VerifyValue keyword to verify a field value by comparing the field value with a user variable (input value).

Signature

The VerifyValue keyword supports the following signature:

```
VerifyValue(AppletRN|FieldRN|[RowNum]/[Active_Record],Operator,Value(or)@Variable)
```

Note the following about the VerifyValue keyword signature:

- If performing the action on tile applets, then the tile index and row number must start with one.
- The row number is optional. If RowNum is not specified, then RowNum defaults to the first row.
- The following operators are supported: greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=), not equals (<>), `contains`, `startswith`, `endswith`, `LIKE`, and so on.

Desktop Examples

The following table describes how to use the VerifyValue keyword to verify a field value by comparing the field value with a user variable (input value) in desktop applications.

Target Object	Inputs	Closing Action	Comments
Contact List Applet|StartDate	=,Today	N/A	Verifies the field value by comparing it with the input value.
Opportunity List Applet| Committed	=,Y	N/A	Verifies the check box field value by comparing it with the input value.
Contact List Applet|M/M|2	=,Mr.	N/A	Verifies the field value by comparing it with the input value.
SIS Account List Applet|Name	=,@Var1	N/A	Verifies the field value by comparing it with the input value.
SIS Account List Applet|Name|2	=,@Var1	N/A	Verifies the field value by comparing it with the input value.
Contact List Applet|Last Name	startswith,Ab	N/A	Verifies the field value by comparing it with the input value, starts with Ab.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Name	<>,A*	N/A	Verifies the field value by comparing it with the input value, not equal to A.
Opportunity List Applet|Name	=,LIKE Q*	N/A	Verifies the field value by comparing it with the input value.
Opportunity List Applet|Name	<>,"Q*	N/A	"Verifies the field value by comparing it with the input value.

Mobile Examples

The following table describes how to use the VerifyValue keyword to verify a field value by comparing the field value with a user variable (input value) in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
SHCE Sales Account List Applet - Mobile|Name|4	=,AG Edwards & Sons, Inc	N/A	Verifies the field value by comparing it with the input value.
SHCE Sales Account List Applet - Mobile|Name	=,@accountname	N/A	Verifies the field value by comparing it with the input value.
SHCE Sales Account List Applet - Mobile|Name	startswith,Ab	N/A	Verifies the field value by comparing it with the input value, starts with Ab.
SHCE Sales Account List Applet - Mobile|Name	=,A	N/A	Verifies the field value by comparing it with the input value, not equal to A.

Wait

You use the Wait keyword to allow the application to remain idle for the user specified time.

Signature

The Wait keyword supports the following signature:

```
wait(seconds)
```

Desktop Examples

The following table describes how to use the Wait keyword in desktop applications.

Target Object	Inputs	Closing Action	Comments
N/A	10	N/A	Application remains idle for 10 seconds.

Mobile Examples

The following table describes how to use the Wait keyword in mobile applications.

Target Object	Inputs	Closing Action	Comments
N/A	10	N/A	Application remains idle for 10 seconds.

Keywords Supporting Tools and Server Configuration

The following keywords support Siebel Tools and Siebel Server configuration for mobile application:

- *InvokePerl*
- *ToolsConfig*
- *ServerConfig*

The following prerequisites are requried to use these keywords:

- Strawberry Perl must be installed and setup.
- Strawberry Perl must be installed on client machines.

InvokePerl

You use the InvokePerl keyword to execute a Perl file.

- Signature supported:

  ```
  InvokePerl (Select Machine|scriptname.pl:@VAR/String[String1,String2....N]/NULL|Output Variable)
  ```

  - `Select Machine` : SiebelServer machine or TestHarness machine. Siebel Server reads the server credentials from the xml file and executes the Perl file on server machine.
  - `TestHarness Machine` : Executes the perl file locally.
  - `Scriptname.pl` : Perl script to be executed.
  - `@VAR/String[String1,String2....N` : Variable support or strings to be passed to the perl file.
  - `@Output Variable` : Variable support to get the value from the perl file.

  > **Note:** You should declare a PerlExecStatus variable in the Perl file. Separate all string inputs by commas to InvokePerl resolves to ARGV[1] in your perl program. Comma separated inputs in your program are parsed, if multiple strings are expected.

- Include PerlExecStatus at the end of the Perl File (as shown in the following example).

```
print"PerlExecStatus = $PerlExecStatus";
```

- To save a value into a variable, use the following syntax in the perl file:

```
print"\@invokePerl1=$ARGV[1].";
print"\@invokePerl2=$ARGV[0].";
print"perlexecstatus = $PerlExecStatus";
```

- Prerequisites:

  Folder Structure: All user defined perl files and cmd.txt for execution should be placed in the `Resources/invokePerl` folder in TestHarness.

  The exe files folder should be placed in the `framework\exe` in TestHarness.

- Update the following tags in config.xml:

```
SIEBEL-SERVER-NAME=macihne-name ----SERVER_MACHINE will be passed as $ARGV [0] to the perl file
SIEBEL-SERVER-MACHINE-LOGIN-ID=userid ----SERVER_LOGIN will be passed as $ARGV [1] to the perl file
SIEBEL-SERVER-MACHINE-PASSWORD=pwd ----SERVER_LOGIN will be passed as $ARGV [2] to the perl file
SIEBEL-SERVER-OS-TYPE=WINDOWS ----SERVER_OS_TYPE will be passed as $ARGV [3] to the perl file
SERVER-DB-TYPE=MSSQL ----SERVER_DB_TYPE will be passed as $ARGV [4] to the perl file
SERVER_DB_NAME=dbname ----SERVER_DB_NAME will be passed as $ARGV [5] to the perl file
DBSERVER-AND-PORT=dbserver,port ----DBSERVER_AND_PORT will be passed as $ARGV [6] to the perl file
DB-TABLE-OWNER=XYZ ----DB_TABLE_OWNER will be passed as $ARGV [7] to the perl file
DB-SERVER-LOGIN=SADMIN|MSSQL ----DBSERVER_Login will be passed as $ARGV [8]&& $ARGV [9] to the perl file
TEST-MACHINE-NAME=machinename ----CLIENT_MACHINE will be passed as $ARGV [10] to the perl file
----------- ----TestHarness Path $ARGV[11]
SIEBEL-GATEWAY-PORT=servername:port ----ENTSERVER_AND_PORT will be passed as $ARGV[12]
SERVER-INSTALLATION-PATH ----C:\\23044\\ses\\siebsrvr\\BIN ----SERVER INSTALLATION PATH will be passed
 as $ARGV[13]
------------- ----Exefiles folder location will be passed as $ARGV[14]
Variable passed from script ----Will be passed as $ARGV[15]
[PERL-PATH]
Ex: PERL-PATH= \\\\servername\\install\\PERL_UTILS\\Perl1\\bin\\perl.exe
```

Examples in the following table show how to use the InvokePerl keyword.

Keyword	Target Object	Inputs	Closing Action
InvokePerl	N/A	SiebelServer Machine;test.pl:NULL;NULL	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:value1,value2;NULL	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:value1;NULL	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:value1,@invokePerl;NULL	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:@invokePerl;NULL	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:@invokePerl, @invokePerl;NULL	N/A

Keyword	Target Object	Inputs	Closing Action
InvokePerl	N/A	SiebelServer Machine;test.pl:@invokePerl, @invokePerl;@invokePerl1	N/A
InvokePerl	N/A	SiebelServer Machine;test.pl:@invokePerl, @invokePerl;@invokePerl1,@invokePerl2	N/A
InvokePerl	N/A	TestHarness Machine;test.pl:NULL;NULL	N/A
InvokePerl	N/A	TestHarness Machine;test.pl:value1,value2;NULL	N/A
InvokePerl	N/A	TestHarness Machine;test.pl:value1,@invokePerl;NULL	N/A
InvokePerl	N/A	TestHarness Machine;test.pl:@invokePerl, @invokePerl;@invokePerl1	N/A
InvokePerl	N/A	TestHarness Machine;test.pl:@invokePerl, @invokePerl;@invokePerl1,@invokePerl2	N/A

ToolsConfig

You use the ToolsConfig keyword to apply Siebel customizations by importing pre-exported sif files using Web Tools. To avoid using the ToolsConfig keyword altogether, use the "Archive - Import from archive" menu option in Web Tools to add the steps to import the pre-exported sif files.

- Signature supported:

  ```
  ToolsConfig(Sif1,Sif2|Merge/Overwrite|branch$,workspace$)
  ```

  Ensure Siebel Tools/Web Tools is installed and working with the correct DSN and tools.cfg file.
- Sif files must be copied to the `Resources\toolsconfig` folder.
- `Resources\` folder must be zipped and attached to the Master Suite attachment applet. In case Resources.zip is already attached, ensure to include new files/folder and re-attach.
- STE run will copy the Resources.zip to the client machine and unzip the file during execution.
- ToolsConfig is a composite keyword which covers the following functionalities:
  - Create an Integration branch using Web Tools.
  - Create a Workspace under the newly created branch using Web Tools.
  - Import the SIF files through CLI.
  - Create a Checkpoint, Submit, and Deliver using Web Tools.
- After completing an operation using the toolsconfig keyword, you can execute change_branch.txt with server restart using the serverconfig keyword to change the application branch. The command in change_branch.txt is shown in the following example.

Example - for call center application:

```
change param WorkspaceBranchName=int_branch for comp SCCObjMgr_enu
```

Update the following tags in unitconfig.xml:

```
PERL-PATH
SIEBEL-TOOLS-MACHINE
WINDOWS-LOGIN-USERID
WINDOWS-LOGIN-PASSWORD
SIEBEL-TOOLS-PATH
SIEBEL-TOOLS-DSN
SIEBEL-TOOLS-USERNAME
SIEBEL-TOOLS-PASSWORD
[PERL-PATH]
Ex: PERL-PATH= \\slcnas607\\karta\\ATF_QTP\\Perl\\bin\\perl.exe
```

For batch run, update these parameters in the "Test Execution" view.

- Toolsconfig Revert option: Once you complete the Testing in the Integration branch, to change the application from Integration branch to MAIN branch, you can execute a command in the server manager using the serverconfig keyword.

  Example - for Siebel Call Center application :

  ```
  "change param WorkspaceBranchName=MAIN for comp SCCObjMgr_enu"#
  ```

  change_branch.txt file:

  ```
  change_branch.txt file will have the command to change the application from MAIN branch to Integration
   branch.
  ```

  Note: Here the txt file name should be change_branch. This is mandatory.

Examples in the following table show how to use the ToolsConfig keyword.

Keyword	Target Object	Inputs	Closing Action	Comments
ToolsConfig	N/A	RTC.sif|Merge|branch$, workspace$	N/A	Importing and compiling single sif file.
ToolsConfig	N/A	RTC.sif ,RTC1.sif,RTC2.sif| Overwrite|branch$,workspace $	N/A	Importing and compiling multiple sif file.

ServerConfig

You use the ServerConfig keyword to configure and start0 the Siebel server.

- Signature supported:

```
ServerConfig(Y)

ServerConfig(commandsFile.txt)
```

> **Note:** If the Launch keyword is used after ServerConfig, then use clear browser in the launch.

`SERVER` : Machine where Siebel server is running. Machine details are available in the SERVER-LOGIN-CREDENTIAL tag, which is in config.xml.

`commandsFile.txt` : Text file containing the list of commands to be executed in server manager.

- Prerequisite:

  Folder Structure: All perl files related to the serverconfig keyword should be available in the `framework\perl` folder in Test Harness

- ServerConfig is a composite keyword which includes the following:
  - Launch server manager
  - Command to be executed
  - Stop ses/swsm
  - Stop sieb server
  - Start ses/swsm
  - Start sieb server
- Update the following tags in config.xml

```
[PERL-PATH]
Ex: PERL-PATH= \\slcnas607\\karta\\ATF_QTP\\Perl\\bin\\perl.exe
SERVER-OS-TYPE
SERVER-LOGIN-CREDENTIAL
AI-SERVER-PORT
SERVER-HOME-PATH
GATEWAYSERVER-TSLPORT
```

Examples in the following table show how to use the ServerConfig keyword.

Keyword	Target Object	Inputs	Closing Action	Comments
ServerConfig	N/A	Y;NULL;IPH3	N/A	Bounce the SERVER.
ServerConfig	N/A	N;cmd.txt;IPH3	N/A	Connect to server manager ,execute the commands in cmd.txt file.

Keyword	Target Object	Inputs	Closing Action	Comments
ServerConfig	N/A	Y;cmd.txt;IPH3	N/A	Connect to server manager ,execute the commands in cmd.txt file and bounce the SERVER.

Unsupported Keywords for Siebel Open UI Keyword Automation

The following keywords are not supported for Siebel Open UI keyword automation testing.

- Calendar(Change slot of Call/Activity)
- Dispatch Board
- Intelligent Advisor
- Find Result Pane
- Search Result Pane
- Charts
- Promotion Designer
- Gantt (Diary)
- Task Pane items
- Gantt Chart - Promotion List

Note the following:

- A script recorded in any language (for example, DEU) can be played back in the same language (DEU in this case).
- A script recorded in ENU cannot be played back in any other NON ENU language (like DEU, JPN, and so on).

You can make a note of the different scenarios and the corresponding values to be entered in the resource.xml file. The different scenarios and values listed in the following table are applicable only for NON ENU languages.

S.No.	Scenario where Resource file needs to be updated	Value needs to be updated in Resource File
1	Column Display (Hide, Show, MovingUP, MovingDown	Selected Field Value should be entered Resource File
2	Verify the Error/Warning message	Error/Warning message in NONENU Language
3	Toggle Value	Selected Togglevalues should be entered in NONENU languages
4	Verify Short Date	Short Date in corresponding NONENU language format
5	Verify Long Date	Long Date in corresponding NONENU language format

S.No.	Scenario where Resource file needs to be updated	Value needs to be updated in Resource File
6	Verify Date Time	Date Time in corresponding NONENU language format
7	Verify Value - Numeric	Numeric value should be entered in corresponding NONENU language format
8	Verify Value - String	Translated string value
9	Verify Value - DateTime/Date	Date Time in corresponding NONENU language format should be entered
10	Any string value present in Combo box	Data LIC value should be entered
11	Visibility DropDown value	Translated value of visibilty drop-down should be entered
12	Verifies Button Existence/Absence	Translated name of the button should be entered
13	Verifies Column name Existence/Absence	Translated name of the column should be entered
14	Verifies Menu Item Existence/Absence	Translated name of the MenuItem should be entered
15	Verifies Existence/Absence of Label of any field	Translated name of the Label should be entered
16	Verifies the Presence/Absence of RadioButton on the Popup	Translated value of radio button label should be entered
17	Compare strings values	Translated string value should be entered
18	Compare numeric values	Numeric value should be entered in corresponding NONENU language format
19	Compare date/DateTime values	Date/DateTime value should be entered in corresponding NONENU language format
20	Picklist applet -if any combo box value need to be selected	Data LIC value should be entered
21	MVG applet -if any combo box value need to be selected	Data LIC value should be entered
22	Application level Menu Item verification	Translated name of the MenuItem should be entered
23	Selection of Radio button in export and import	Labels translated value should be entered
24	Selection of Radio button in Advance Sort	LIC values of drop-down should be entered

S.No.	Scenario where Resource file needs to be updated	Value needs to be updated in Resource File
25	In simple query - query with any combo box value	Data LIC value should be entered for query combo box
26	Verify the Application level Menu Enable/Disable	Translated value of Application level menu should be entered
27	Verify the Applet level Menu Enable/Disable	Translated value of Applet level menu should be entered
28	Verify Label in any form applet	Translated value of Label should be entered in File
29	Verify applet level record count (1 - 30 of 30+)	Translated Record count value should entered in file
30	Applet level search option	Data LIC value for combo box value should be entered

17 Database Test Scripts

Database Test Scripts

This chapter describes various sample database test scripts and the actions performed by some sample database test scripts. It includes the following topics:

- *Sample Database Test Scripts*
- *Actions Performed by Sample Database Test Scripts*

Sample Database Test Scripts

The following table describes the list of scripts available in the sample database.

Master Suite	Test Set	Sequence	Test Scripts	Description
COM_SampleDBDemo _ Desktop	COM_Sample DB_ ABOFlows	1	COM_SampleDB_ NewOrderFlow	Ron Weasley is a residential customer looking for a good deal on a new family wireless plan. Ron calls the wireless carrier for a quotation. The contact center agent looks up Ron Weasley's account, browses the product catalog, selects the best family wireless package and offers three new phones for Ron's family at varying price points.
COM_SampleDBDemo _ Desktop	COM_Sample DB_ ABOFlows	2	COM_SampleDB_ UpdateOrder	David Smith is an existing residential customer who has a family wireless plan. David calls the wireless carrier to add additional lines to his family package. Contact Center agent looks up David Smith's account and adds 5 new lines to David's account. The CRM system notifies the agent that David's current package is limited to 4 lines. The agent adds 3 new lines as one line is already associated to David, selects the options for text messages, data plan and reviews the order details with David.
COM_SampleDBDemo _ Desktop	COM_Sample DB_ ABOFlows	3	COM_SampleDB_ DebundlePromotion	David is an existing residential customer who has a family wireless plan. David calls the wireless carrier to remove his spouse from the family package and make it an individual plan.
COM_SampleDBDemo _ Desktop	COM_Sample DB_ ABOFlows	4	COM_SampleDB_ Upgrade PromotionFlow	Suzie Harera is a residential customer who has a basic quad play bundle - Internet, TV, Home Phone and Wireless Phone. Suzie calls

Master Suite	Test Set	Sequence	Test Scripts	Description
				the contact center to upgrade to a premium quad play bundle.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ BulkOrder	1	COM_SampleDB_ BulkOrder_New OrderFlow	Axiom Financial Group is a business customer that purchases 50 wireless service plans with pre-assigned numbers from the Communication Service Provider's Account Manager. The Account Manager connects to the CSP's commerce application, creates an account list, and places a bulk order to onboard 50 new users. Alternatively, Axiom Financial Group requests an optional feature "WLAN Subscription" to be added for 20 of its loyal users. Competitive pressure requires the communication service provider to upgrade all Promo 60 plan customers to Promo 120 plan.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ BulkOrder	2	COM_SampleDB_ BulkOrder_Modify Flow	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ BulkOrder	3	COM_SampleDB_ BulkOrder_Upgrade OrderFlow	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ ProductOffer	1	COM_SampleDB_ ProductOffer_Data Setup	Communication Service Provider creates a targeted mass market offer across varying channels (Web, Retail store, SMS). Customer accepts the offer(s) using a single click: offer to add free worldwide calls to their current wireless package at a 50% discount during the holiday season, upgrade their SMS package to higher package or auto-renew their contract.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ ProductOffer	2	COM_SampleDB_ ProductOffer_ ModifyAdd	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ ProductOffer	3	COM_SampleDB_ ProductOffer_ ModifyReplace	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ ProductOffer	4	COM_SampleDB_ ProductOffer_ ModifyUpgrade	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ PG_ Flows	1	COM_SampleDB_ PG_ Data_Setup_1	Product administrator verifies the community offer (promotion group) defined in the system using Siebel administrator (SADMIN) login. PG Scenario 3 disconnects a Silver Participant membership and Nation 550 Minutes bundle promotion for PGA2 child

Master Suite	Test Set	Sequence	Test Scripts	Description
				account and it also disconnects the entire promotion group from PGA1 account.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ PG_ Flows	2	COM_SampleDB_ PG_ Data_Setup_2	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ PG_ Flows	3	COM_SampleDB_ PG_ Scenario_1	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ PG_ Flows	4	COM_SampleDB_ PG_ Scenario_2	Same description as in previous row.
COM_SampleDBDemo _ Desktop	COM_SampleDB_ PG_ Flows	5	COM_SampleDB_ PG_ Scenario_3	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Desktop	1	COM_SampleDB_ Telco_ DataSetup Flow	CitiExpress is a business customer. Tim is a B2B sales rep for a leading Communication Service Provider (CSP). CitiExpress is Tim's customer with a wireless contract for 1000 employees. CitiExpress wants to change the barring options for 4 of their employees' phones and change the SIM number for an employee who had recently lost his phone. Tim connects to CSP's commerce application, searches for his customer, and place an order to change the barring options and change the SIM card number.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	1	COM_SampleDB_ Telco_ ChangeSIM _Messages	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	2	COM_SampleDB_ Telco_ SelectAll _Remove	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	3	COM_SampleDB_ Telco_ChangeSIM_ SubmitOrder	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	4	COM_SampleDB_ Telco_ AddRemove_ Flow1	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	5	COM_SampleDB_ Telco_ AddRemove_ Flow2	Same description as in previous row.
COM_SampleDB Demo_Telco	COM_SampleDB_ Telco_ Mobile	6	COM_SampleDB_ Telco_ DataCleanUp	Same description as in previous row.
COM_SampleDBDemo _Mobile	COM_SampleDB _ PartnerCommerce	1	COM_SampleDB_ PartnerCommerce_ NewOrder	Samuel Crenshaw is a partner user and places an order on the wireless carrier's partner portal. Samuel uses his iPad to login to the partner portal, browses the partner

Master Suite	Test Set	Sequence	Test Scripts	Description
				catalog, compares product attributes and prices, adds the best product to his cart, selects and configures options and places an order on behalf of his customer.
COM_SampleDBDemo _Mobile	COM_SampleDB _ PartnerCommerce	2	COM_SampleDB_ PartnerCommerce_ Compare	Same description as in previous row.
COM_SampleDBDemo _Mobile	COM_SampleDB_ eSales	1	COM_SampleDB_ eSales_ ERIGBY Login	Kate Allison is shopping for a high speed Internet, TV and long distance phone service for her home. Kate uses her iPad to navigate to the communication service provider's web store, browses the product catalog and adds the standard triple play package to her shopping cart. Kate completes the new user registration form, reviews the shipping and pricing details and submits an order. Eleanor Rigby is a business user and logs in to the communication service provider's web store to track the orders placed on behalf of her company. Additionally, Eleanor registers and administers other users of her company.
COM_SampleDBDemo _Mobile	COM_SampleDB_ eSales	2	COM_SampleDB_ eSales_ Flow1	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	1	Workflow Activation	Eric Hess is an existing customer, who calls up to the customer care to raise a service request, that could be a problem related to products or services purchased from the company. Tim Malone the call center agent receives the call and creates an SR for the issue raised by the customer. Tim enters the required details to the SR including description, asset details, contact details, etc.
Demo Flow for Service Request	Service Request Test Set	2	Create SR	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	3	Associate Product	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	4	Update SR	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	5	Check Entitlements	Tim checks customer's entitlements for the service. Once done, he looks for the possible solution for the issue raised by the customer and adds the solution to the SR. He decides that a field engineer needs to visit the customer site, so he creates an activity and assign it to an appropriate field engineer to fix the problem.
Demo Flow for Service Request	Service Request Test Set	6	Associate Asset	Same description as in previous row.

Master Suite	Test Set	Sequence	Test Scripts	Description
Demo Flow for Service Request	Service Request Test Set	7	Associate Entitlements	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	8	Associate Solution	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	9	Add Activity Plan	Same description as in previous row.
Demo Flow for Service Request	Service Request Test Set	10	Delete Data	Delete all the newly created data from the environment.

Actions Performed by Sample Database Test Scripts

The following table describes the actions performed by test scripts in the sample database.

Sl #	Script Name	Action to be Performed	Test Step #	Current Data	Proposed Change
1	Script - COM_SampleDB_ Telco_ChangeSIM_ SubmitOrder	N/A	34	Row Number is NULL	Row Number: Active_Record
2	Script - COM_SampleDB_ Telco_DataSetupFlow	Update	1	Component Alias: Siebel Power Communications	Component Alias: Siebel Power Communications_Setup
2	Script - COM_SampleDB_ Telco_DataSetupFlow	Insert @ Test Step #26	27	N/A	Click New, enter the following values, and then save the record: Keyword: HierarchicalList Applet RN: Order Entry - Line Item List Applet (Sales) Row #: 2 Action: Expand Variable: NULL **Note:** Select Renumber from the Menu.
2	Script - COM_SampleDB_ Telco_DataSetupFlow	Insert @ Test Step #108	109	N/A	Click New, enter the following values, and then save the record: Keyword: HierarchicalList

Sl #	Script Name	Action to be Performed	Test Step #	Current Data	Proposed Change
					Applet RN: Order Entry - Line Item List Applet (Sales) Row #: 2 Action: Expand Variable: NULL **Note:** Select Renumber from the Menu.
2	Script - COM_SampleDB_ Telco_DataSetupFlow	Insert @ Test Step #133	134	N/A	Click New, enter the following values, and then save the record: Keyword: HierarchicalList Applet RN: Order Entry - Line Item List Applet (Sales) Row #: 2 Action: Expand Variable: NULL **Note:** Select Renumber from the Menu.
3	Script - COM_SampleDB_ Telco_ChangeSIM_ Messages	Update	1	User Name: ANTHYAGA	User Name: SADMIN
3	Script - COM_SampleDB_ Telco_ChangeSIM_ Messages	Update	22	ExpectedMessage Substring: Unable to complete SIM Swap. There is a stolen or admin bar for the selected service. Remove the stolen or admin bar to continue with your request. (SBL-ORD-51426)	ExpectedMessageSubstring: "Unable to complete SIM Swap. There is a stolen or admin bar for the selected service. Remove the stolen or admin bar to continue with your request."(SBL-ORD-51426)
4	Script - COM_SampleDB_ Telco_SelectAll_Remove	Update	12	Item RN: Select All | Select All	Item RN: SelectAll | Select All
4	Script - COM_SampleDB_ Telco_SelectAll_Remove	Update	13	End Action: OK	End Action: NULL
4	Script - COM_SampleDB_ Telco_SelectAll_Remove	Update	14	Item RN: Select All | Select All	Item RN: SelectAll
4	Script - COM_SampleDB_ Telco_SelectAll_Remove	Update	22	Item RN : Select All | Select All	Item RN: SelectAll

Sl #	Script Name	Action to be Performed	Test Step #	Current Data	Proposed Change
4	Script - COM_SampleDB_ Telco_SelectAll_Remove	Update	24	Item RN: Select All | Select All	Item RN: SelectAll
5	Script - COM_SampleDB_ Telco_ChangeSIM _SubmitOrder	Update	34	Row Number is NULL	Row Number: Active_Record
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Update	4	Value: TelcoExpress_ 05252017_22343988	Value: @TelcoAccName
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Update	6	Value: TelcoExpress_ 05252017_22343988	Value: @TelcoAccName
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Update	9	ItemRN: Select All	ItemRN: SelectAll
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Update	11	ItemRN: Select All	ItemRN: SelectAll
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Update	13	ItemRN: Action	ItemRN: Product Name
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Insert @ Test Step #8	9	N/A	Click New, enter the following values, and then save the record: Keyword: ClickButton Applet RN: TOUI AssignBar Buttons ItemRN: Cancel Request End Action: NULL **Note:** Select Renumber from the Menu.
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Insert @ Test Step #9	10	N/A	Click New, enter the following values, and then save the record: Keyword: ClickLink Applet RN: TOUI CustDash Action Launchpad ItemRN: Expand Applet Value/Variable: NULL **Note:** Select Renumber from the Menu.
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Insert @ Test Step #10	11	N/A	Click New, enter the following values, and then save the record:

Sl #	Script Name	Action to be Performed	Test Step #	Current Data	Proposed Change
					Keyword: ClickButton Applet RN: TOUI CustDash Action Launchpad Item RN: AddRemoveBar EndAction: NULL **Note:** Select Renumber from the Menu.
6	Script - COM_SampleDB_ Telco_AddRemove_ Flow2	Insert @ Test Step # 21	22	N/A	Click New, enter the following values, and then save the record: Keyword: SortColumn Applet RN: TOUI AssignBar Order Items Item RN: Service Id Sorting Order: Asc **Note:** Select Renumber from the Menu.
7	Script - COM_SampleDB _Partner Commerce_Compare	Update	6	ItemRN: Select All	ItemRN: SelectAll
8	Script - COM_SampleDB _eSales_Flow1	Update	1	User Name: GUESTCST	User Name: PortalApplication
8	Script - COM_SampleDB _eSales_Flow1	Update	15	Row Number: 3 Value/Variable: NULL	Row Number: remove value in this field. Value/Variable: IPTV Service
8	Script - COM_SampleDB _eSales_Flow1	Update	38	ItemRN: Select All | Select All	ItemRN: SelectAll

18 Reports

Reports

A report provides the Pass and Fail results with detailed step level comments and screenshots. This chapter describes the functionality provided to support report generation. It includes the following topics:

- *About Report Generation*
- *Functionality for Report Generation*
- *Generating a Combined Report*

About Report Generation

You can control whether or not to generate reports and capture screenshots and test results during test script execution – for more information, see *Functionality for Report Generation*.

Reports are generated in both HTML and JSON format for each test script that is run individually (Unit Mode) or as part of a Master Suite (Batch Mode). The following table shows the report files that are generated in HTML and JSON format.

Format	Report Files	Location
JSON	• <Test Script>timestamp.json • <Test Set>timestamp.json • <Master Suite>timestamp.json	**\Reports\<MasterSuite>timestamp\logs\json**
JSON	• Report.html (uses Report.js and webresources folder) • Report.js (aggregation of json files)	**\Reports\<MasterSuite>timestamp\logs**
HTML	• <Test Script>timestamp.html • <Test Set>timestamp.html. • <Master Suite>timestamp.html	**\Reports\<MasterSuite>timestamp>\logs**

A single report (Report.html) is automatically generated which provides a consolidated snapshot across test script, test set, and Master Suite data along with a detailed failure analysis of results. You can also manually generate a combined results report using multiple Report.html files - for more information, see *Generating a Combined Report*.

Report.html provides the following key information:

- A consolidated view of a batch run with data from JSON report files.
- A Test Summary showing Pass, Fails, Pass%, and Not Executed results at Master Suite level.
- A Test Set result summary and consolidated failure analysis across all test scripts.

- A Failure Analysis summary which:
  - Lists unique error log messages across all results, ranking them by number of test scripts failed in descending order.
  - Contains links to the test script where the error occurred.
  - For data-driven tests, shows the iteration where the error occurred in brackets.

Report.html depends on the following:

- The webresources folder under `...\DesktopintSiebelAgent\plugins\SiebelTestAutomation`, which contains the following: css, fonts and img folders.
- Report.js, which wraps the data from individual JSON files. Report.js must be in the same folder as Report.html.

Functionality for Report Generation

The following functionality is provided to retrieve the keyword automation results during report generation:

- Ability to specify whether or not to capture a screenshot (for Pass and Fail test results) as follows:
  - Select the Screenshot checkbox in the Test Script - Test Step Form applet screen if you want to capture a screenshot.
  - Deselect the Screenshot checkbox in the Test Script - Test Step Form applet screen if you do not want to capture a screenshot.

  Note that if a screenshot is required for Pass or Fail test cases, then the screen or view to be captured must be in Active mode.

  Note also that the executable client machine must open in InetPub.
- Ability to control whether or not to generate reports and capture screenshots and test results during test script execution as follows:
  - To enable or disable detailed test results and screenshot capture during unit mode/single test script playback, configure the DetailedReport check box as shown in *Plugin Configurations*.
  - To enable or disable report generation and screenshot capture during the test script automation batch run, configure the Parameters field as shown in *Configuring the Siebel Test Execution Job*.

Generating a Combined Report

You can manually generate a combined results report using multiple Report.html files. The purpose of generating a combined report is to aggregate *totals* and consolidate the *failure analysis* across multiple Master Suite (batch) runs. To generate a combined report as shown in the following procedure, Perl is required on the DISA machine.

To generate a combined report

1. Copy the Master Suite report folders (under `\Reports`) to the following location on one of your DISA machines:

   `...\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Reports`

2. Go to the following folder and locate the CombinedReportScript.pl file:

   ```
   ...\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\perl
   ```

3. From this folder, start a Command Prompt and run the following command (perl file), providing the DISA home path and Reports folder path as parameters:

   ```
   perl CombinedReportScript.pl <DISA home path> <Reports folder path>
   ```

   For example:

   ```
   perl CombinedReportScript.pl "C:/DISA" "C:/DISA/DesktopIntSiebelAgent/plugins/SiebelTestAutomation/Reports"
   ```

   After this completes, a CombinedReports folder (`\Reports\CombinedReports\`) is created if not already created with a timestamped subfolder. In the subfolder, the combinedreports.js and Report.html files are created and Report.html opens in the default browser.
4. Repeat this procedure as required – for example, whenever any new folders are added, removed or modified – in order to generate a new combined report:
   - Add more folders to the `\Reports` folder.
   - Run the perl file (perl CombinedReportScript.pl).
   - A new subfolder will be created under `\Reports\CombinedReports\` with a unique timestamp.

Offline uploading of Test Results

The KWDResultUpload.jar is a utility provided as part of DISA to upload test results to the database in case the database was down during a test run.

The KWDResultUpload.jar is located at: `C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework\`

Run this utility when a test run fails to upload results to the database, such as due to a database outage. Run the utility to upload reports, create test passes, and upload test step results automatically.

Running the utility:

1. Open Command Prompt as Administrator.
2. Run the following command (ensure directory paths and values are updated accordingly):

Command Line Arguments

Arguments	Descriptions
`/j <User id>`	Siebel User ID used for REST/SOAP operations.
`/g <password>`	Password for the Siebel User ID.
`/r <Siebel Web Server URI>`	URI of the EAI-enabled Siebel Server for REST/SOAP operations. **Format**: *`https://servername:port/<Application Context>`* **Example**: *`https://servername.com:16690/siebel`*

Arguments	Descriptions
`/ c <EAI Component>`	(Optional) Ability to provide Custom EAI component. `default-> /app/eai/enu/, Custom -> /app/my_eai/deu here my_eai` is user defined EAI Component.
`/i <Run id>`	Run Id for which results need to be uploaded.
`/n <DISA machine name>`	(optional) Hostname of the machine where DISA is installed.
`/l <Reports Location>`	Directory path for the parent folder of Run ID folder containing report files.

The Report Location can be any valid directory path to the parent folder, where the folder with Run Id as the name is placed. The following locations are examples for Reports Location parameter:

- `"C:\STEReportsLogs"`
- `"C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\TestExecutions"`
- `"C:\MyReports"`

Note:

- Ensure to provide the parent directory of the report location.
- Run the utility on the same machine where result reports are available.

Examples

1. Report Location `c:\STEReportsLogs` (alternative command)
   - ```
     java -jar C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework
     \KWDResultUpload.jar /j SADMIN /g ldap /r
     https://
      my_vm.my_domain
      .com:16690/siebel
     /i 88-3EQNA3 /l "C:\STEReportsLog"
     ```
2. Report Location `c:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\TestExecutions`
   - ```
     java -jar C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework
     \KWDResultUpload.jar /j SADMIN /g ldap /r
     https://
      my_vm.my_domain
      .com:16690/siebel
     /i 88-3EQNA3 /l "C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\TestExecutions"
     ```
3. Report Location `c:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\TestExecutions` (with additional options)
   - ```
     java -jar C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\Framework
     \KWDResultUpload.jar /j SADMIN /g ldap /r
     https://
      my_vm.my_domain
      .com:16690/siebel /i 88-3EQNA3 /n my_disa_vm.my_domain
      .com
     /c /app/eai/enu /l "C:\DISA\DesktopIntSiebelAgent\plugins\SiebelTestAutomation\TestExecutions"
     ```
     ```

19 Mac Credentials

Mac Credentials

The MAC login credentials for mobile applications are as follows:

```
<MAC-CREDENTIALS>
<MAC-USERNAME>id1</MAC-USERNAME>
<MAC-PASSWORD>pwd1</MAC-PASSWORD>
</MAC-CREDENTIALS>
```

Use this format to update the MAC credentials in the config.xml file.