Testing Siebel Business Applications

Siebel Innovation Pack 2016, Rev A

February 2017

ORACLE

Contents

1 What's New in This Release

What's New in Testing Siebel Business Applications, Siebel Innovation Pack 2016, Rev A

Table 1. lists the changes in this revision of the documentation to support this release of the software. Siebel Innovation Pack 2016 is a continuation of the Siebel 8.1/8.2 release.

Topic	Description
Automating Functional Tests	Deleted chapter.
Automating Load Tests	Deleted chapter.
Functional Test Object Reference	Deleted appendix.
Implementing Siebel OpenUI Keyword Automation Testing	New chapter.
Setting up Keyword Automation Testing on iOS	New chapter.
Setting Up Android Mobile Devices for Automation Testing	New chapter.
Keywords Reference	New appendix.
Mac Credentials	New appendix.
Reports	New appendix.

What's New in Testing Siebel Business Applications, Siebel Innovation Pack 2016

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

2 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, version 7.7 or later. It does not describe specific features of the Siebel Business Applications product suite.

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

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.
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 the remainder of this chapter "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://www.bitpipe.com/rlist/term/Testing.html
- Economic Impact of Inadequate Infrastructure for Software Testing
  http://www.nist.gov/director/prog-ofc/report02-3.pdf
- Empirix
  http://www.empirix.com/Empirix/Corporate/Resources/
- International Federation for Information Processing
  http://www.ifip.or.at/
  (click on the "Search IFIP" link)
- Making Software Development High Performance
  http://www.swforum.com/
  (click on the "Search" hyperlink)
- Internet/Software Quality Hotlist
  http://www.soft.com/Institute/HotList/index.html
- Mercury Interactive
  http://download.mercury.com/cgi-bin/portal/download/index.jsp
- Software Testing Institute
  http://www.softwaretestinginstitute.com/index.html
- 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 Figure 1) is demonstrated by a slowing of defect discovery and fix rates, while maintaining even levels of test case activity.

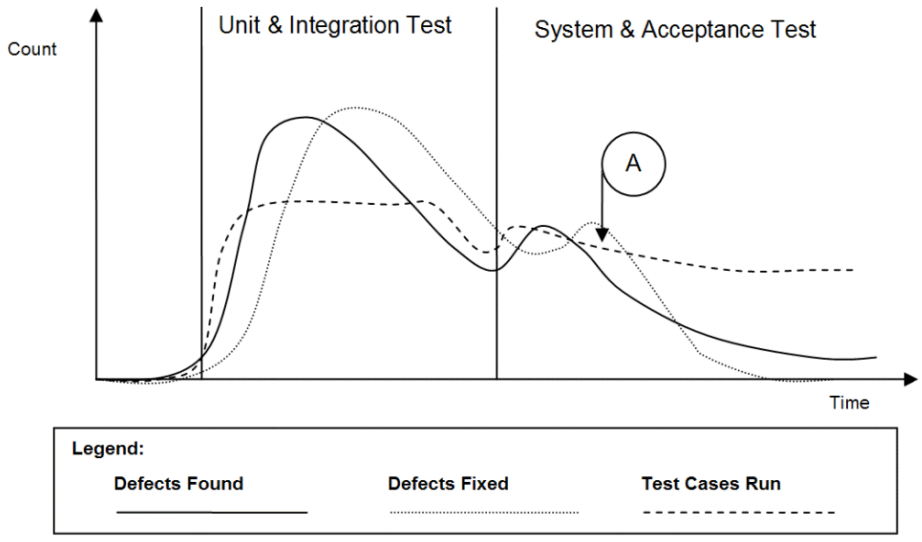


Figure 1. Trend Analysis of Testing and Defect Resolution

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. Figure 2 presents a high-level map of testing processes.

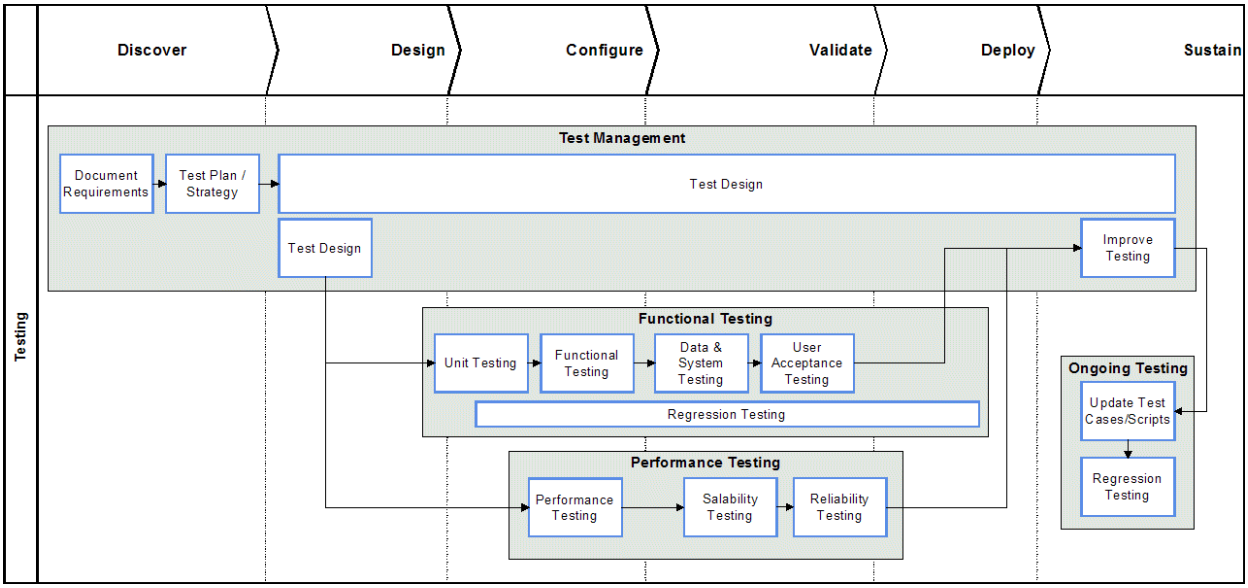


Figure 2. High-Level Testing Process Map

Each of the seven testing processes described in this book are highlighted in bold in the diagram and are outlined briefly in the following topics:

- Plan Testing Strategy on page 18
- Design and Develop Tests on page 19
- Execute Siebel Functional Tests on page 19
- Execute System Integration Tests on page 19
- Execute Acceptance Tests on page 19
- Execute Performance Tests on page 20
- Improve and Continue Testing on page 20

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.

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.

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.

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.

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.

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.

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.

3 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 test planning process is illustrated in Figure 3 on page 22.

The inputs to this process are the business requirements and the project scope. The outputs, or deliverables, of this process include:

- **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 and when to perform testing
  - **Entrance and exit criteria.** Define inputs required to perform a test and 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 for test environments should also be defined. Clear entry and exit criteria for each environment should be defined.

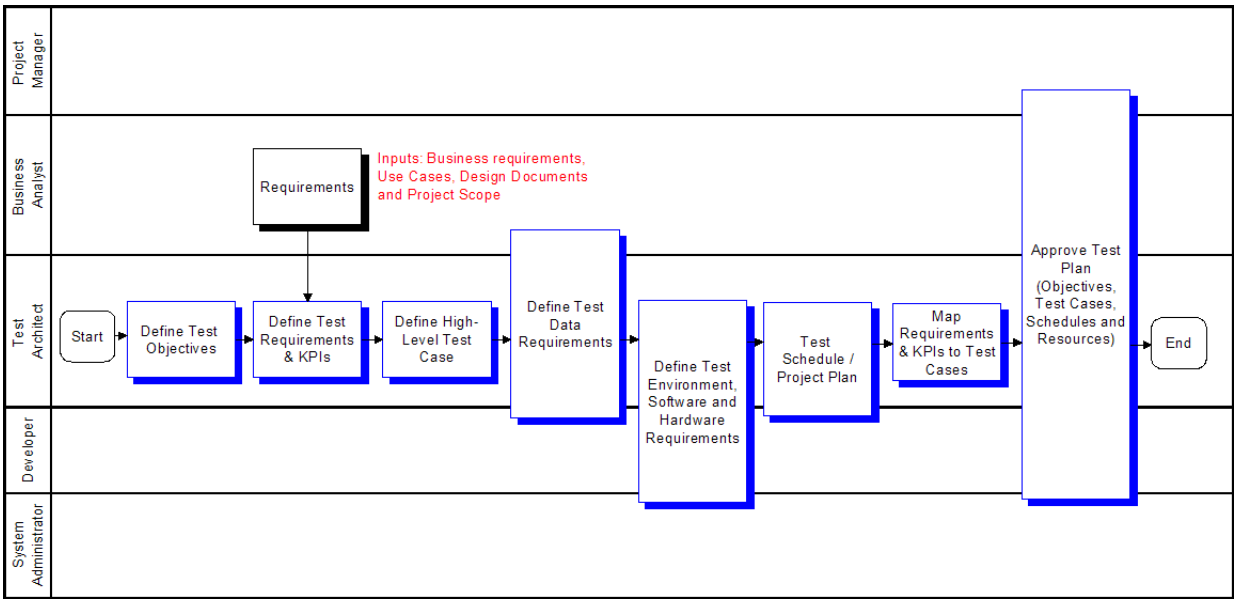


Figure 3. Plan Testing Strategy Process

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 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. Figure 4 shows a sample Test Objectives document.

A test case covers one or more test objective, 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" on page 23.

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

Figure 4. Sample Test Objectives

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. Figure 5 shows a sample test case list. Each test plan contains 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

Figure 5. Sample Test Plan: Test Case List

This example 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

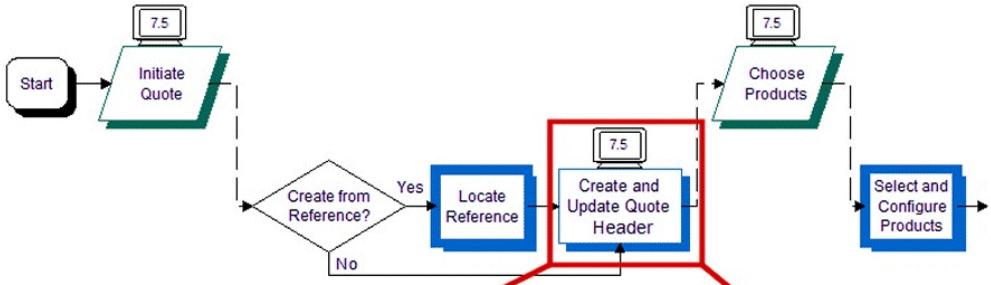
TC5.x – System integration testing

Note how the test schedule is included in Figure 5. For 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 the test plan, there are a number of test types that you must define:

- **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. Figure 6 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	
2		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.

Figure 6. Business Process-Driven Test Case with its Corresponding Process Diagram

- **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" on page 27. 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. Typically, the component inventory is done at the project level, and is not a testing-specific activity. There are two ways projects typically 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 Figure 7 is an example of a component inventory.

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

Figure 7. Sample Component Inventory Document

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.

As shown in Figure 7 on page 27, one column of the component inventory provides a risk score to each component based on the guidelines above. 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 mange 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

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

It is important that you develop test cases in close cooperation between the tester, the business analyst, and the business user. The process illustrated in Figure 8 illustrates some of the activities that should take place in the test development process.

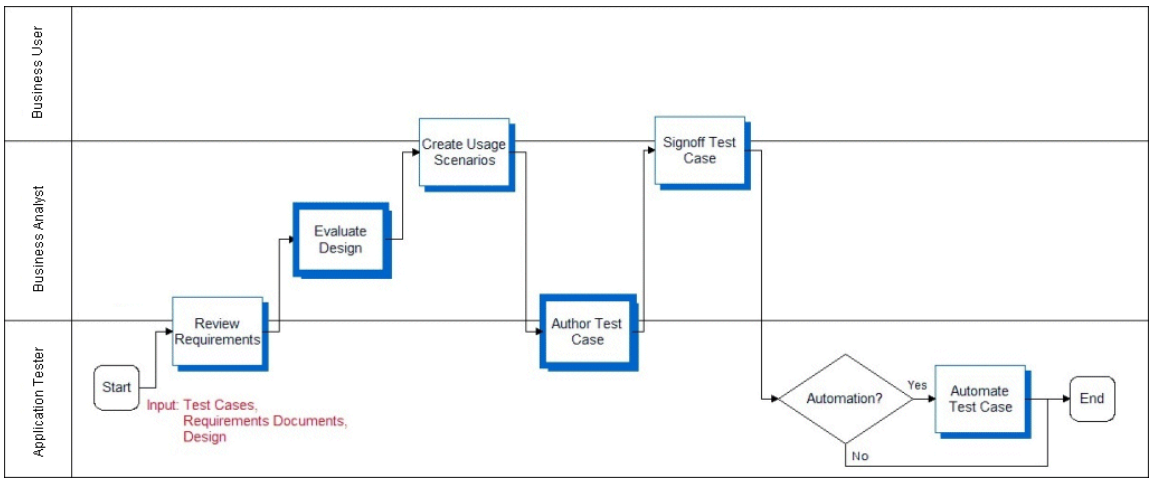


Figure 8. Develop Tests Process

To generate valid and complete test cases, they must be written with full understanding of the requirements, specifications, and usage scenarios.

The deliverables of 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. The design evaluation process is illustrated in Figure 9.

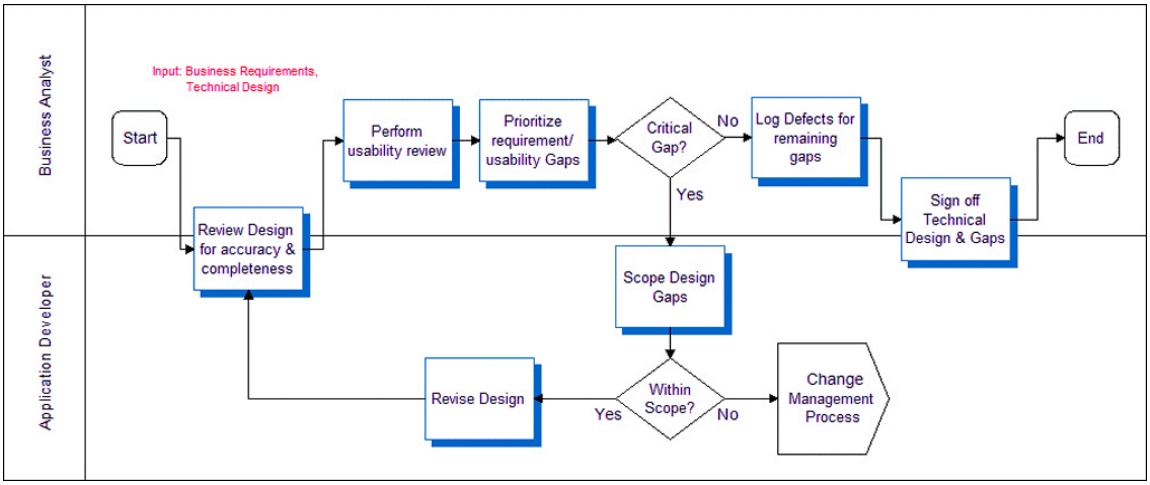


Figure 9. Evaluate Design Process

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, white board 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. Figure 10 illustrates the process for authoring a test case.

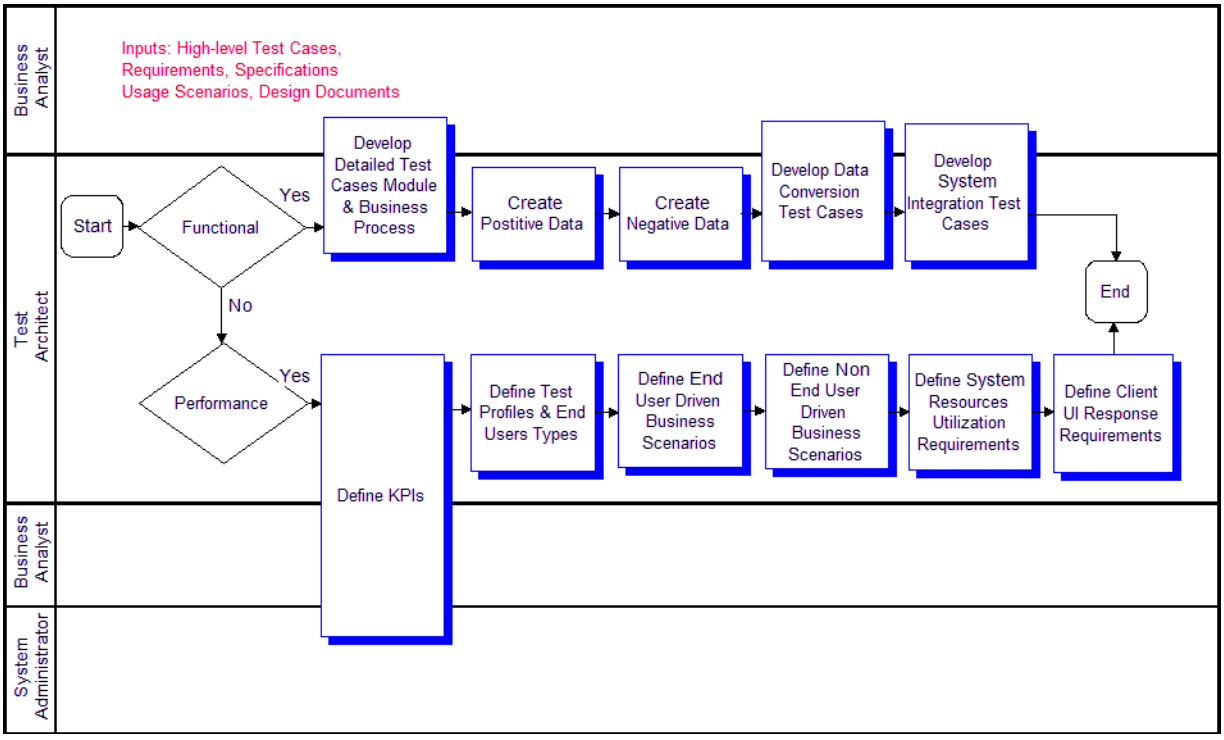


Figure 10. Test Authoring Process

As you can see from the process, functional and performance test cases have different structures based on their nature.

Functional Test Cases

Functional test cases test a common business operation or scenario. Table 2 shows some examples of functional test cases.

Table 2. Common Functional Test Cases

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.

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 Case

A test case describes the actions and objects that you want to test. A case is presented as a list of steps and the expected behavior at the completion of a step. Figure 11 shows an example of a test case. Notice that 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.

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				

Figure 11. Sample Test Case

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. Where functional tests focus on validating support for a scenario, system tests make sure that the structure of the application is correct. Table 3 shows some examples of typical system tests.

Table 3. Common System Test Cases

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.

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. Figure 12 presents a typical test profile for a performance test.

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**

Figure 12. Performance Test Profile

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 to 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 the simulated users attempt simultaneous access to 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.

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 "AutoQuote" 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**

Figure 13. Sample Test Case Excerpt with Wait Time

As shown in Figure 13, the user wait times are specified in the scenario. It is important that wait times be distributed throughout the scenario, and reflect the times that an actual user takes to perform the tasks.

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. Please refer to the instructions provided by your selected tool vendor for details:

- **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

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

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, integration infrastructure, and network infrastructure. Therefore, the objective of the Execute Siebel Functional Tests process is to verify that the Siebel application functions properly before inserting it into the larger system environment. This process is illustrated in Figure 14.

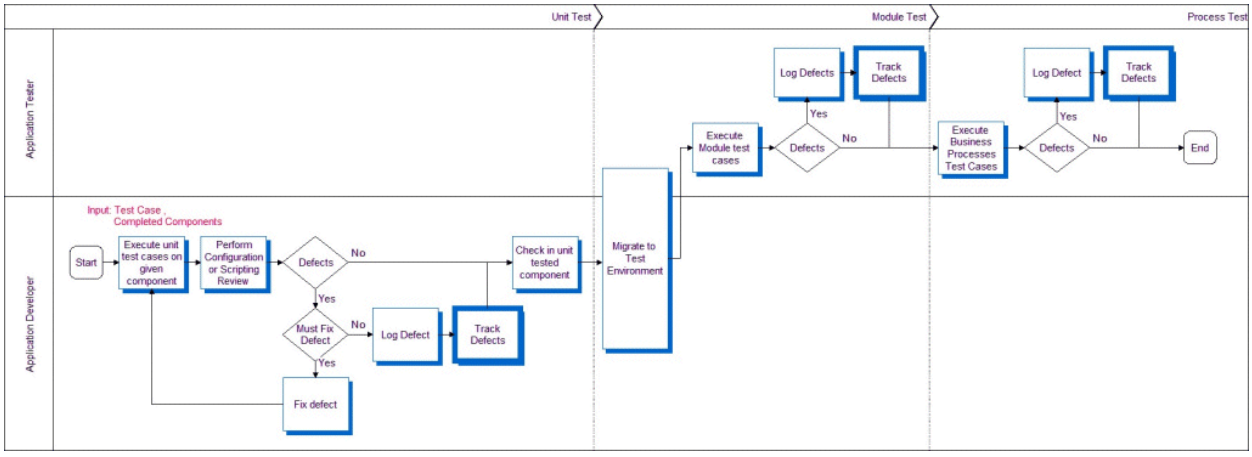


Figure 14. Execute Siebel Functional Tests Process

There are three phases in this process:

- **Unit test.** The unit test validates the functionality of a single component (for example, an applet or a business service).
- **Module test.** The module test validates the functionality of a set of related components that make up a module (for example, Contacts or Activities).

- **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).

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.

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 Figure 15, is designed so that those with the best understanding of the customer priorities are in control of defect prioritization. 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.

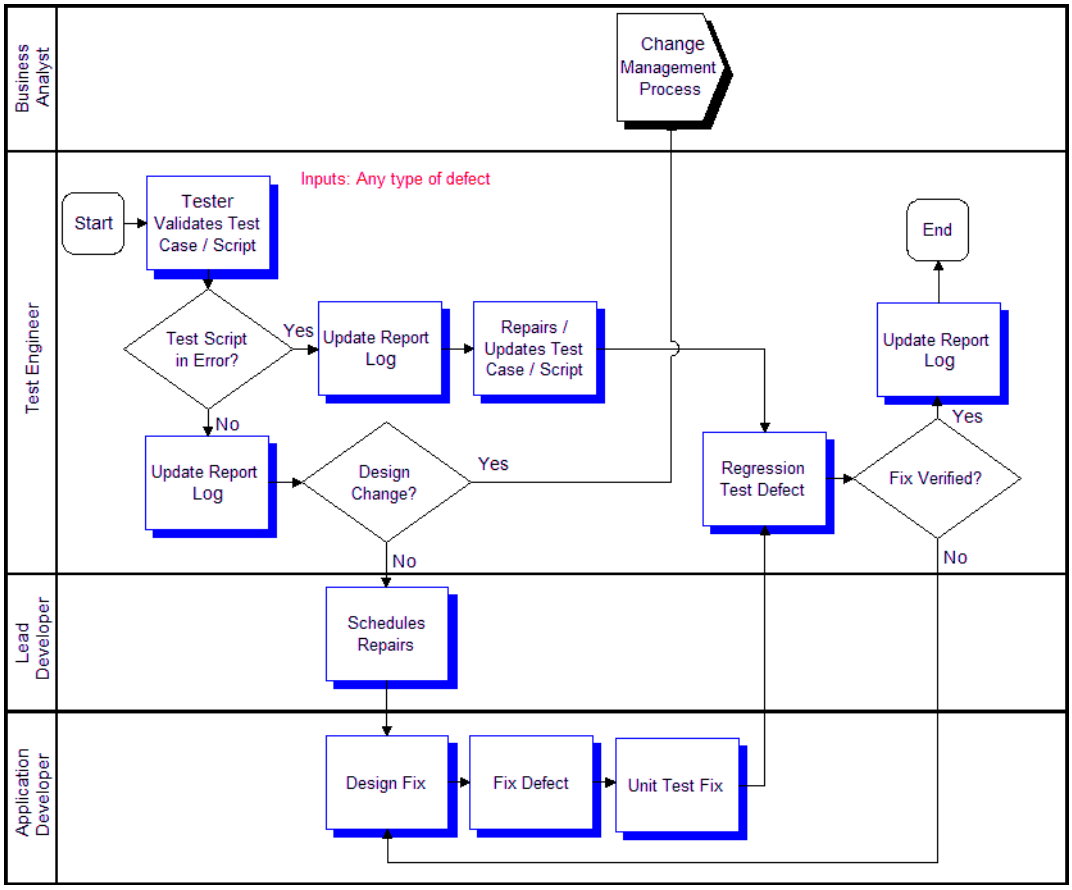


Figure 15. Track Defects Subprocess

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 (reexecution 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 reexecution 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 reexecute 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

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. Figure 16 illustrates this process.

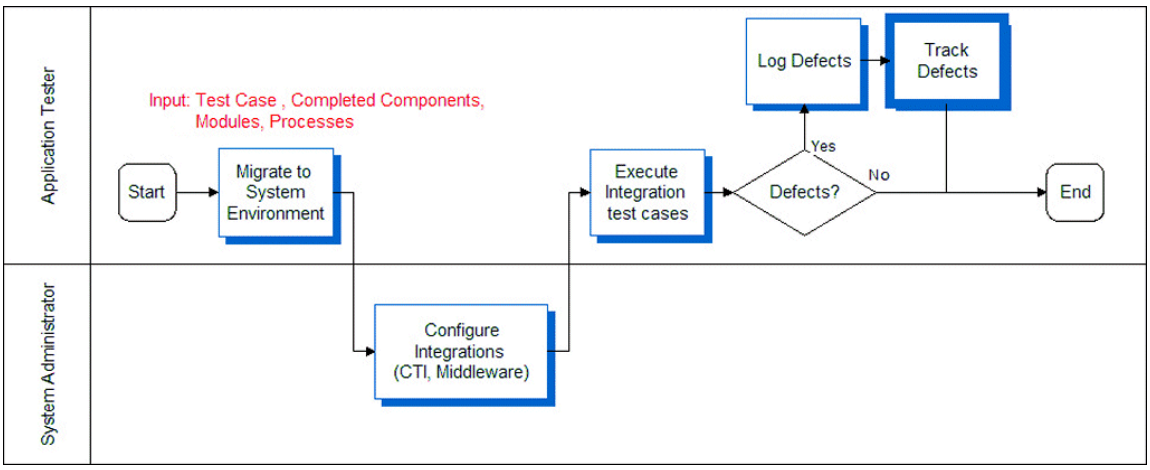


Figure 16. Execute Integration Tests Process

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. Figure 17 illustrates this process.

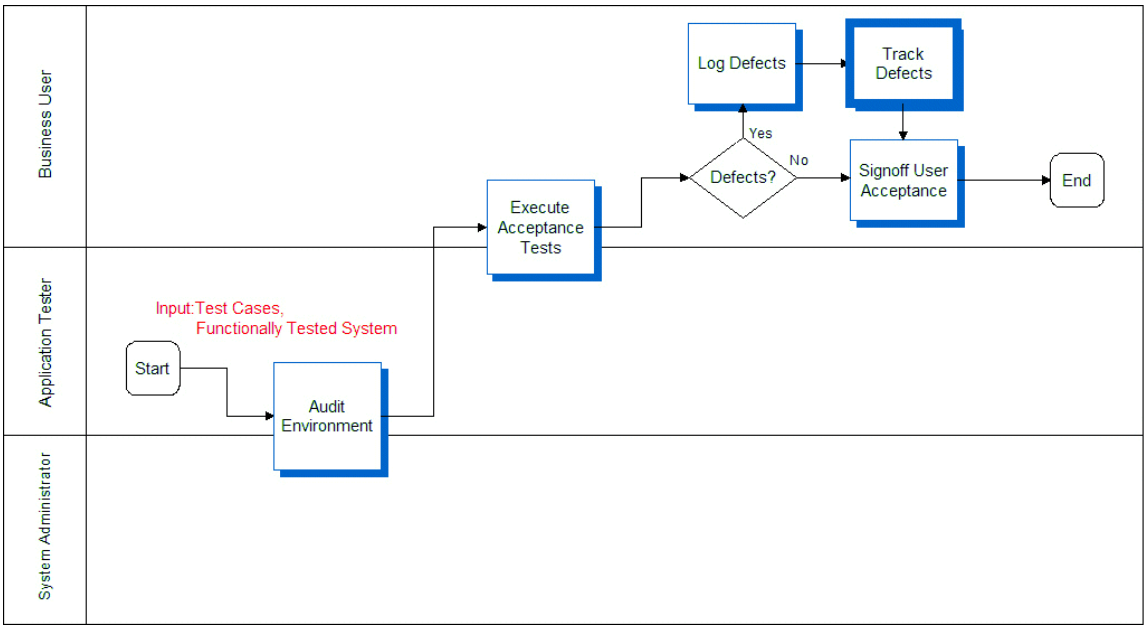


Figure 17. Execute Acceptance Tests Process

7 Execute Performance Tests

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

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.

Figure 18 illustrates the process for performance test execution. The first step involves validating recorded user-type scripts in the system test environment.

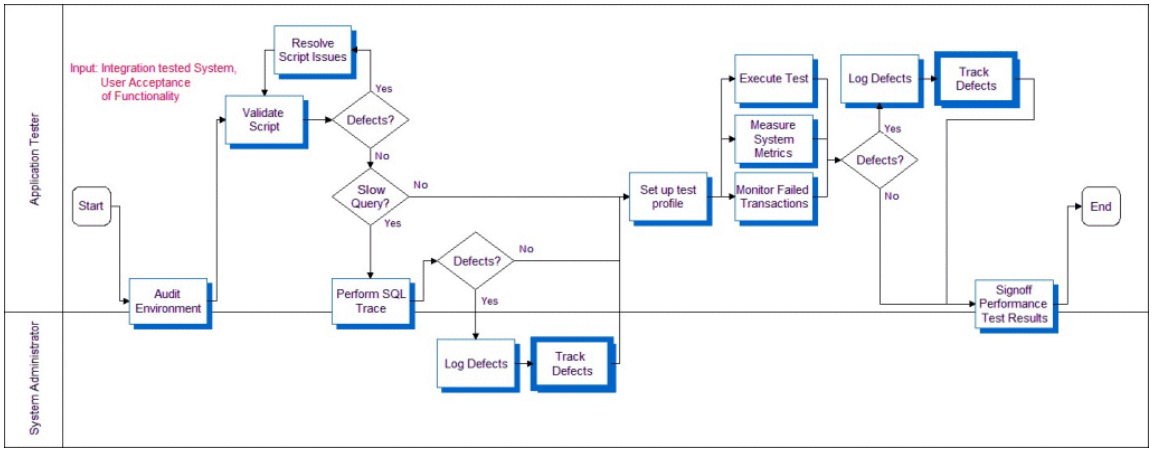


Figure 18. Execute Performance Tests Process

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

This chapter describes the steps you can take to make iterative improvements to the testing process, as illustrated in Figure 19 on page 54. It includes the following topic.

- "Improve and Continue Testing" on page 53.

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" on page 16.

This ongoing lifecycle of the application is an opportunity for continuous improvement in testing. 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.

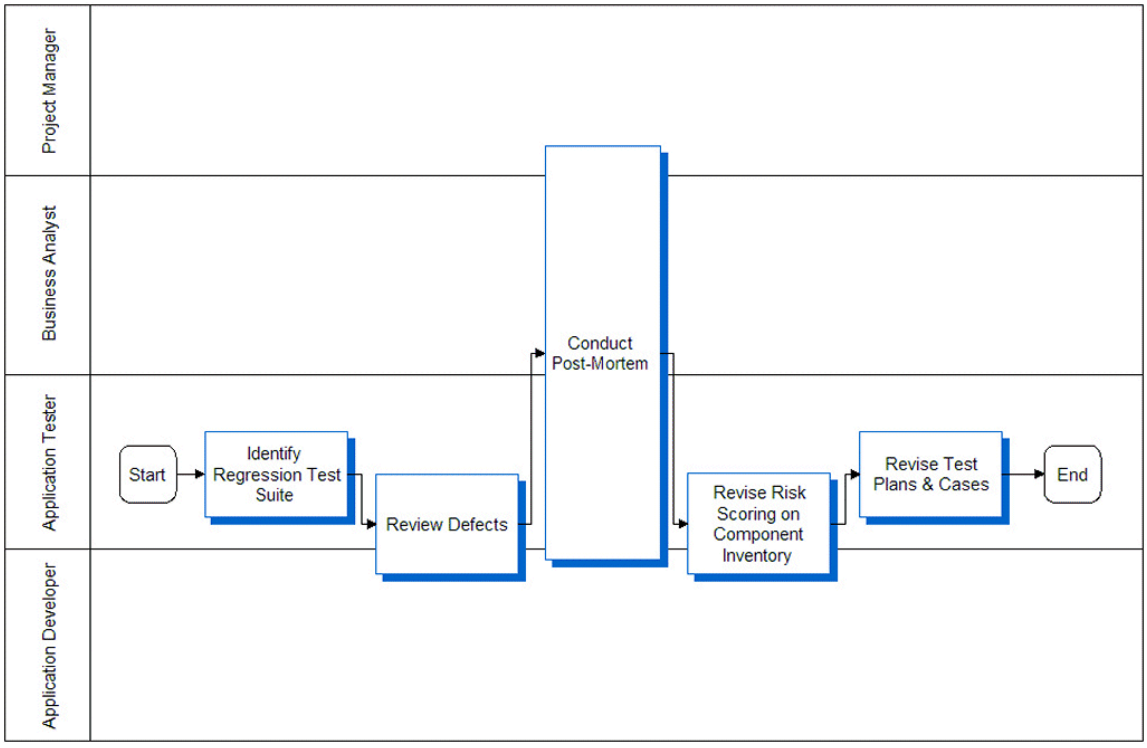


Figure 19. Improve Testing Process

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). 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. 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.

9 Implementing Siebel OpenUI Keyword Automation Testing

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

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. Data required for interaction
4. Optional) Description of the test step.

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

Table 4.

Action or Keyword	Target Object	Data	Test Step
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
Input value	Repository name of applet and button.		Enter a value in the Name field.

Process of Implementing Siebel Open UI Keyword Automation Testing

This topic describes the tasks involved in implementing Siebel Open UI keyword automation testing using 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.

Perform the following tasks to implement keyword automation testing for this scenario:

1. Creating a Test Script
2. Adding Test Steps to Test Scripts
3. Grouping Test Scripts into a Test Set
4. Grouping Test Sets into a Master Suite and Exporting to CSV
5. Configuring the Test Run

Creating a Test Script

This task describes the procedure involved in creating a new test script

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

To create a test script

1. Make sure that the Siebel application is running.
2. Navigate to the Administration - Keyword Automation screen, then the Test Scripts view.
3. Click the plus (+) icon to create a new Test Script.
4. Enter values for the fields shown in the following table.

Action or Keyword	Target Object	Data
Name	xyz	Name of the test script.
Test Script Description	Test	Description of the test script.
Status	Active	Status of the test script
Test Case ID	Test1_001	ID associated with the test script.

5 From the Test Scripts Menu, choose Save Record.

A new test script is created.

Adding Test Steps to Test Scripts

This task describes the procedure involved in adding test steps to test scripts.

This task is a step in Process of Implementing Siebel Open UI Keyword Automation Testing. Ensure that the Siebel application is up and running.

To add test steps to test scripts

1 Navigate to the Administration - Keyword Automation screen, then the Test Scripts view.

2 Drilldown 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 Test Step Form applet, enter field values for the fields 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.

Table 5.

Action or Keyword	Target Object	Data
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.

Table 5.

Action or Keyword	Target Object	Data
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 every step that is performed in the test step.

5 Save the record.

Capturing Automation Attributes for Test Steps

This task shows you how to use the attributes pop-up window to capture automation attributes for test steps.

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

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

- Ensure that the following behavior settings are set by navigating to Tools, User Preferences, Behavior
- Set Navigation Control to Tab.
- Set Theme to Aurora.
- Ensure that the Siebel application is running in AutoOn mode.
- Ensure that the URL command is set to the AutoOn mode, for example, as follows:

http://slc10akj.us.oracle.com/callcenter_enu/start.swe?SWECmd=AutoOn

Being in the 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.

c The Siebel Automation Information dialog box appears.

d Copy the string value from the COPY STRING field and paste it into the Paste Attributes field of the Test Step.

e 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.

About Keyword Reference

The Keyword Reference.doc is available in the SiebelTestAutomation folder. The document lists all the keywords and provides you with information on how to use keywords with examples.

NOTE: The place holders for NULL values IPH1;IPH2;IPH3 are not available in the document

Grouping Test Scripts into a Test Set

This task shows you how to group test scripts into a test set.

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

To group test scripts into a test set

1 Navigate to the Administration - Keyword Automation screen, then the Test Set view.

2 Click the plus (+) icon to create a new Test Set.

3 Enter values for the fields shown in the following table.

Table 6.

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

4 Drilldown on a value in the Name field and then click the plus (+) icon to associate test scripts to the selected Test Set.

5 Update the Sequence field for each associated test script.

Grouping Test Sets into a Master Suite and Exporting to CSV

This task shows you how to group test sets into a master suite and export the master suite to CSV output, which you can then harness for test execution.

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

To group test scripts into a master suite and export to csv output

1. Navigate to the Administration - Keyword Automation screen, then the Master Suite view.
2. Click the plus (+) icon to create a new Master Suite.
3. Enter values for the fields shown in the following table.

Table 7.

Field	Description	Sample Value
Name	Name of the master suite.	Xyz.
Description	Description of the master suite.	Test.
Status	Status of the master suite.	Active.
Team Name	Team name associated with the master suite.	For example, SPARTANS

4. Drilldown on a value in the Name field and then click the plus (+) icon to associate test sets to the selected Master Suite.
5. Click Export Test Steps.

A CSV file is generated containing all the Test Scripts. Test Steps are sorted according to sequence of the following: Test Set, Test Script, and Test Step sequence number. An example test script file, SampleScript.csv, is located here:

...\SiebelTestAutomation\Scripts\SampleScript.csv

The test script file is consumed by the Keyword Framework to execute on the Siebel Application.

Configuring the Test Run

This task shows you how to create an execution configuration record and then run the test scripts:

- Creating the Execution Configuration Record Creating the Execution Configuration Record
- Running the Test Scripts Running the Test Scripts

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

Creating the Execution Configuration Record

This task shows you how to create an execution configuration record.

1. From Site Map, navigate to the Automation Execution Configuration screen.
2. Click the plus (+) icon to create a new Configuration XML file.
3. Complete the Master Suite Id field, and then drill down on the value in the Test Run # field

Complete the following fields (related to the Client machine):

Table 8.

Field	Description	Sample Value
Browser	The target browser to run automation.	Firefox, Chrome
(It's a LOV picklist)		
Test Harness Machine	The Windows client machine on which automation is run.	Machine1.network_domainname.com
Run Reference	The run reference value. This is a user defined value.	BLD_01_001, TestRun_001
Result Path	The location of the Reports folder, where the test run result files are stored.	C:\SiebelTestAutomation\Reports
Is Final?	Select this check box when you have specified all the relevant values and are ready to generate configuration.	Select this check box.

4. In the Server Credentials applet, enter the Application Name (such as, Sales or Call Center), the URL details, and the OS (Operating System) and Language.
5. Click Generate Configuration XML.
6. Save the output file to the Test Harness folder.

Running the Test Scripts

This task shows you how to run the test scripts and check the individual test script results.

To run test scripts

1. Download the SiebelTestAutomation.zip file in the client machine.
2. Place the config.xml file generated in the main folder and the csv file in the products folder.
3. Update the config.xml file.
4. Update the absolute path of the csv test script in the config.xml file.

5 Update the password for the users.

6 Update the URL and the reports path.

7 Double-click the start.vbs file to run test scripts.

8 Navigate to Reports/Logs/<MasterSuiteName>_Mastersuite<timestamp>.html.

The hierarchy of the tests is listed in the Report.

a Drilldown on the test set for test script results.

b Drilldown on each test script for individual test script results.

About the Siebel Test Automation Zip File

The SiebelTestAutomation.zip contains the folder structure required for executing the csv script. The Framework folder contains all the required JARS and library files.

Table 9.

Folders / Files	Type		Execution	
Documents	KeywordReference.html		User guide	e
				Contains framework/third-party-software jars, autoit/perl scripts.
Framework	Exe		Do not modify	Contains necessary Exe files which are used in the SiebelTestAutomation Framework.
	Lib			Contains necessary Library files (Jar files) which are used in the SiebelTestAutomation Framework.
	Perl			Contains necessary Perl scripts which are used in SiebelTestAutomation Framework.

Table 9.

Folders / Files	Type		Execution	
	SiebelTestAutomation.jar			Compiled executable Jar which drives KWD execution.
Reports			Post- execution	The folder gets created post execution, generates execution results for review.
Resource	fileupload		Script authoring	Folder containing user developed files needed for fileupload, inboundwebservices, invokeperl, toolsconfig keywords.
	Inboundwebservicescall			
	invokeperl			
	toolsconfig			
Scripts				
Samplescript.csv			Place the exported csv Script for execution	
config.xml			Pre-setup	XML File which has the configuration details of execution(URL,user name password, script location)
start_batch.vbs			Execution	Triggers script execution.

Table 9.

Folders / Files	Type		Execution	
User_opted_operation.txt				A text file which contains the information like Pause, Resume and abort.User has the option to Pause/ Resume/Abort the current execution by updating the value in User_opted_operation text.
Log_randomnumber.log				Logger file which is generated at the time of execution and it is used for debugging purpose.
Logs.txt				A text file generated at the time of execution which can be used for debugging the script.

10 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

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 Mobile Application Framework (MAF) requires the following software to be installed and setup accordingly for keyword automation testing:

- **Xcode**. For more information, see "Installing XCode on the XCode iOS Simulator".
- **Oracle JDeveloper and MAF**. For more information, see "Installing Oracle JDeveloper and Setting Up the Mobile Application Framework".

Installing XCode on the XCode iOS Simulator

This task shows you how to install Xcode version 6.3.1 on XCode's iOS simulator. Before installing Xcode, note that the following prerequisites apply:

- OS X 10.10 or later (macOS)
- Intel, 64-bit processor

To install XCode on Xcode iOS simulator

1. Start Safari on your iOS device, go to http://developer.apple.com/xcode/downloads, and then sign in with your Apple ID.
2. Locate Xcode version 6.3.1, and then double-click the dmg file to install Xcode.
3. Drag and drop Xcode into the Applications folder on your iOS mobile device.
4. Open Xcode from the Finder/Applications folder.

5 On the Xcode and iOS SDK License Agreement screen that appears, click Agree.

Xcode and all additional required components and tools are downloaded.

Installing Oracle JDeveloper and Setting Up the Mobile Application Framework

This task shows you how to install Oracle JDeveloper and set up the Mobile Application Framework (MAF)

To install Oracle JDeveloper and set up MAF

The following procedure shows you how to install Appium version 1.4.0 on an iOS Device?

1 Download Oracle JDeveloper version 12c (12.1.3.0.0) from the following location:

http://www.oracle.com/technetwork/developer-tools/jdev/downloads/index.html

NOTE: Oracle JDeveloper requires Java Run Time Environment 1.7 or later.

2 Go to the folder where you downloaded the install file and execute the following command:

java –jar <jdeveloper file>

3 Open Oracle JDeveloper, click Help, and then select Check for Updates.

4 On the screen that appears, select the All option for Mobile Application Framework 2.3, and then select Finish.

5 Restart Oracle JDeveloper.

6 Click File, click New, and then select Application.

On the screen that appears, verify that the following items are included in the list:

- MAF Application from Archive File
- Mobile Application Framework Application

The presence of these items indicates that MAF has been set up successfully.

11 Setting Up Android Mobile Devices for Automation Testing

This chapter shows you how to set up Android mobile devices for keyword automation testing. This chapter contains 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® 8, 7, 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 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 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 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.

Make a note of the name and location where the SDK is 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:

- Install Android 5.0.1 (API 21)
- Install Android 6.0 (API 23)
- Install Extras.

5. Click Install Packages.

To accept the license agreement for each package, double-click on each package name, and then click Install.

The loading progress is shown at the bottom of the 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 Environment Variables as follows:
   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:
      - Enter the following Variable name: ANDROID_HOME
      - 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 Copy or make a note of the path to both these folders, for example, 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

Verifying Android Installation and Configuration

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

To verify if the Android is installed and configured correctly

- 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 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.

You use the AVD Manager 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

Table 10.

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= Android_Chrome - Automation on Chrome Browser.
- APPLICATION_TYPE= Android_Browser - Automation on AVD Native Browser.

Since emulators take more time to start, start the emulator manually before triggering a run.

- APPLICATION_TYPE= Android_MAF - Automation on Siebel MAF Application

A Keywords Reference

This appendix defines the keywords that are available for Siebel Open UI keyword automation testing and describes how to use each keyword. This appendix contains the following topics:

Keywords Definition

Table 11. Siebel Open UI Keyword Automation Testing - Keywords Definition

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
AttachmentManager	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
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 top unread message in the notification list.	Yes	No
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

Table 11. Siebel Open UI Keyword Automation Testing - Keywords Definition

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
DragAndDrop	Drags a record and drops it on 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	No
GetAboutRecord	Reads the value from the About Record pop-up window and stores the value in a user variable.	Yes	Yes
GetConfigParam	Reads a value from test.ini or config.xml 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
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 collapsess 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

Table 11. Siebel Open UI Keyword Automation Testing - Keywords Definition

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
InvokeObject	Invokes an object in a specified field in a list or form applet.	Yes	Yes
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
MultiSelectRecordsInListApplet	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	Removess 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
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 Drop-down 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

Table 11. Siebel Open UI Keyword Automation Testing - Keywords Definition

Keyword Name	Description	Applies to Desktop?	Applies to Mobile?
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	No
VerifyFocus	Verifies the focus present on an applet, view, field, or row depending on the value provided (True or False).	Yes	Yes
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	No
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 top 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

Keywords Description

This topic provides descriptions of each keyword that is supported for Siebel Open UI keyword automation testing.

AttachmentManager

You use the AttachmentManager keyword to perform actions in the Attachment Download Manager page in MAF applications. AttachmentManager uses the Id attributes.

NOTE: The AttachmentManager keyword works only on MAF iOS and MAF Android devices.

Signature

The AttachmentManager keyword supports the following signature:

```
AttachmentManager(Appletid, Name of the Entity|Name of the File Name, ...;DOWNLOAD/
DOWNLOADALL/REMOVE/CLOSE/TOP)
```

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
downloadMgrDiv	AdminSalesTool|Zonall_Launch.zip ,AdminSalesTool|ZonallPatientProfile.zip;DOWNLOAD		Downloads the selected files.
downloadMgrDiv	AdminSalesTool|Zonall_Launch.zip ,AdminSalesTool|ZonallPatientProfile.zip;DOWNLOADALL		Downloads the all files.
downloadMgrDiv	AdminSalesTool|Zonall_Launch.zip ,AdminSalesTool|ZonallPatientProfile.zip;REMOVE		Removes the selected files.

Target Object	Inputs	Closing Action	Comments
downloadMgrDiv	AdminSalesTool|Zonall_Launch.zip ;TOP		Moves the file to the top position.
downloadMgrDiv	DivAdminSalesTool|Zonall_Launch. zip;Close		Closes the AttachmentDownload Manager. **NOTE:** Use close action with only one name or value pair.

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		NULL	Clicks the SUI (Synergy theme) toolbar button.
NULL|SiebTabViews		NULL	In SUI theme, invokes the L2 level links that appear when a button is clicked.
SIS Account List Applet|NewQuery		NULL	Clicks the Newquery button in a list applet.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|DeleteRecord		OK	Clicks the Delete Record button in a list applet.
SIS Account Entry Applet|GotoNextSet		NULL	Clicks the Go to Next Set button in a form applet.
Product Pick Applet(Eligibility)| PickRecord		NULL	Clicks the drop-down list in a pop-up window.
NULL|PickRecord		NULL	Clicks a button in a pop-up window.
Close		OK	Clicks Close or X button in a pop-up window and then clicks OK.
Close		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		OK	Clicks the Delete Record button in a list applet.
SHCE Sales Contact List Applet - Mobile|Create		NULL	Clicks the Create button in a list applet.
Close		OK	Clicks Close or X button in a pop-up window and then clicks OK.
NULL|NewQuery		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.

Signatures

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>		Drills down on the Last Name value. For example, drills down on the last name value Ramakrishna.
ClickLinkAppletRN|ClassName	NULL		Clicks on the link based on the Class Name.
NULL|SiteMap	NULL		Clicks on the toolbar item (Site Map).
Account Contact List Applet|Last Name|2	Pinas		Drills down on the Last Name of the second record. For example, drills down on the Last Name of the second record value Pinas.
SIS Account List Applet|ToggleListRowCount	NULL		Clicks the Show More link in the list applet.
SIS Account List Applet|ToggleListRowCount	Show More		Clicks the Show More link in the List applet.
SIS Account List Applet|ToggleListRowCount	Show Less		Clicks the Show Less link in the List applet.
LS Pharma Inbox Applet|ButtonMaximizeApplet	Expand		Expands the list in the Home page.

Target Object	Inputs	Closing Action	Comments
LS Pharma Inbox Applet|ButtonMinimizeApplet	Collapse		Collapses the list in the Home page.
Contact Home Public and Private View Link List Applet|Name	My Contacts		Clicks the link of the recently viewed contacts screen (My Contacts).
NULL|Save Query As Applet.SaveAs	NULL		Clicks the button link in the pop-up window.
Account Home Public and Private View Link List Applet|2	All Accounts		Click the frequently viewed links.

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>		Drills down on the name value in the first row.
SHCE Sales Contact List Applet - Mobile|Last Name	<last name>		Drills down on the name value.
LS Home Page Calendar Applet-Mobile|@Var	NULL		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		Clicks the specified timeslot in the Calendar applet.
LS Home Page Calendar Applet-Mobile|@AccountCall+slotCol-0-10:00	NULL		Supports clicking on the dynamic Timeslot in the Calendar applet (Account call).

Target Object	Inputs	Closing Action	Comments
LS Home Page Calendar Applet-Mobile|@FirstName+@LastName+@title+slotCol-0-10:00	NULL		Supports clicking on the dynamic Timeslot in the Calendar Applet (Contact call).
LS Home Page Calendar Applet-Mobile|@Accountname+@Date	NULL		Supports clicking on the dynamic activity in the Calendar applet.

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		Clicks the Sync button and moves to offline mode.
GoOffline	uploadGoOnline;Online		Clicks the Sync button and selects the option uploadGoOnline in offline mode.

Target Object	Inputs	Closing Action	Comments
GoOffline	syncStayOffline;Online		Clicks the Sync button and selects the option syncStayOffline in offline mode.
GoOffline	uploadStayOffline;Online		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 top unread notification message in the notifications list. You also use the keyword to click the Mark All as Read option in the notification list, and then close the notification 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 top unread notification.

Target Object	Inputs	Closing Action	Comments
MsgBrdCstIcon	Account_10142015_041155918	Close	Clicks the top unread message in the notification list and closes the control.
MsgBrdCstIcon	Mark All As Read	Close	Clicks the Mark all as Read option in the notifications and closes the control.
MsgBrdCstIcon	Mark All As Read	KeepOpen	Clicks the Mark all as Read option in the notifications and keeps the control open.
MsgBrdCstIcon	NULL	Close	Closes the notification control.
MsgBrdCstIcon	Account_10142015_041155918	KeepOpen	Clicks the top unread message in the notifications 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 left in the list applet.

Target Object	Inputs	Closing Action	Comments
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 right 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.
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 pop-up window and clicks Cancel.
SIS Account List Applet|Columns Displayed (SWE)	NULL;NULL;NULL	Columns Displayed Popup Applet (SWE).ButtonReset	Opens the ColumnDisplayed pop-up 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 left 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 right in the list applet.

Target Object	Inputs	Closing Action	Comments
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: = (equalsl), > (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
	@var1|>=|3.56		Verifies a variable value by comparing it with the expected value 3.56.
	@Var1|=|@Var2		Verifies a variable value by comparing it with the expected value.
	@Var|startswith|nc		Verifies a variable value by comparing it with the expected value that starts with nc.
	@Var|endswith|nc		Verifies a variable value by comparing it with the expected value that ends with nc.
	@Var|=|{[123]}		Verifies a variable value by comparing it with the expected value.
	@Var|=|"text123;		"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
	@var1|>=|3.56		Verifies a variable value by comparing it with the expected value 3.56.
	@Var1|=|@Var2		Verifies a variable value by comparing it with the expected value.
	@Var|startswith|nc		Verifies a variable value by comparing it with the expected value that starts with nc.
	@Var|endswith|nc		Verify a variable value by comparing it with the expected value that ends with nc.
	@Var|=|{[123]}		Verifies a variable value by comparing it with the expected value.
	@Var|=|"text123;		"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.

DragAndDrop

You use the DragAndDrop keyword to drag a record from an applet and drop it on 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 portrait mode and both source and destination applets must be active.

Signatures

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 drag a record from an applet and drop it on a specific field in desktop applications.

Target Object	Inputs	Closing Action	Comments
FINCORP Deal Account Pick Applet|NULL|3	Opportunity List Applet|Account		Drags a record from an applet and drops it on a specific field.
Opportunity List Applet|Account	Opportunity List Applet|Primary Revenue Win Probability		Swaps the columns in the applet.

Mobile Examples

The following table describes how to use the DragAndDrop keyword to drag a record from applet and drop 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		Drags a record from an applet and drops it on a specific field.
LS Home Page Contact List Applet - Mobile |NULL|3	LS Home Page Calendar Applet-Mobile |slotCol-0-08:00		Drags a record from the call applet and drops it on 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			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:

NOTE: To upload a file, the file must be placed in the FileUpload folder at the TestHarness location.

NOTE: 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

The FileUpload keyword does not apply to mobile applications.

GetAboutRecord

You use the GetAboutRecord keyword to obtain parameter values from the About Record pop-up window 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		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		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		Stores the Row, UpdatedOn, Created_on field values from the About Record pop-up window in a variable.

GetConfigParam

You use the GetConfigParam keyword to read the values in the test.ini and config.xml. files. The keyword retrieves one value at a time, given the correct parameter name (tagname).

Signature

The GetConfigParam keyword supports the following signature:

```
GetConfigParam(APPLICATION-NAME/Null|tagname|@variable name)
```

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.

Desktop and Mobile Examples

This table describes how to use the GetConfigParam keyword to read the values from the config.xml.

Target Object	Inputs	Closing Action	Comments
	Null|SIEBEL-TOOLS-USER-NAME|@userName		Gets the SIEBEL-TOOLS-USER-NAME value.
	BIP;COMPONENT|TAGNAME;@variable		If the input is Component, then the value of the tag name will be retrieved and stored in the variable.
	BIP;USER-NAME|VALUE;@variable		If the input is User-Name, then the corresponding password will be obtained and stored in the variable.

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		Stores the total record count in a user variable.
SIS Account List Applet|NULL	@Var1		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		Stores the total record count in a user defined variable.
SHCE Sales Account List Applet - Mobile|	@Var1		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.

Signatures

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		Stores the Release state in a variable.
SIS Product List Admin Applet|XA Class Name|3	@variable2		Stores XA Class Name state for the third record in a variable.
SIS Product List Admin Applet|SiebAppletMenu|Delete Record (SWE)	@variable3		Stores the Delete record (SWE) state in a variable.
Menu-File|File - Send Fax	@variable4		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		Gets the state of a variable and stores the state in @var.
SHCE Sales Account List Applet - Mobile|SiebAppletMenu|Record Count (SWE)	@var		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		Gets a value based on ClassnName.
SIS Account List Applet|Account Status|2	@Var		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		Gets a value from the SalesRep2 field in the form applet and stores the value in a variable.

Target Object	Inputs	Closing Action	Comments
Opportunity Form Applet - Child|Description2	@Var		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		Gets a value from the pop-up input field.
SIS Account List Applet|Name	@Var[3]		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		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		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		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		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
			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			Clicks on the specified threadbar link.
Views_tb_1			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.

Signatures (Desktop)

The GoToView keyword supports the following signature on mobile 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.
- 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 desktop 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			Navigates to the accounts screen, first level view bar.
NULL|Account List View|L2			Navigates to the accounts screen, second level view bar.
NULL|TNT SHM Opportunity Agenda View|L3			Navigates to the accounts screen, third level view bar.
NULL|Quote Item XA View|L4			Navigates to the accounts screen, fourth level view bar.
Sitemap.Asset Management Screen|Sitemap.Asset Mgmt - Assets View|NULL			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			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		Expands the first record.
Order Entry - Line Item List Applet (Sales)|8	Collapse		Collapses the eighth record.
Order Entry - Line Item List Applet (Sales)|2	GetChildItemsCount|Variable		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,Tagname2|@var2/NULL;Tagname|Value/@var/@STOREVAR,Tagname|value/@var/@STOREVAR)
```

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.

XML structure

To pass the dynamic variable into the XMLFIle PFB, use the following XML structure:

```
<tagname>$var</tagname>
```

```
<tagname>="$var"</tagname>
```

INI File Structure

An example of the INI file structure is as follows:

```
[WS]

AUTHENTICATION_TYPE=URL\HEADER

SERVER= SERVER NAME: PORT NUMBER

CREDENTIALS= APPLICATION LOGIN|PASSWORD
```

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
	Prod1.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|@STOREVAR		Stores the tagname value in the response (ActiveFlag) in the @STOREVAR variable.
	Prod1.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|Y		Passes the values to a SOAP request dynamically.
	Currency.xml;ConversionRateResult|0.0161		Sends the XML request and verifies the response.
	Currency.xml;swip:WorkspaceName|@var,swip:WorkspaceReuseFlag|@var2;ActiveFlag|@var2		Passes the values to a SOAP request dynamically.
	new.xml;NULL;OrderItem|ProductId|88-46TS9:ActionCode|@STOREsdf		Stores ActionCode from Order item tags in the @STOREsdf variable with the unique productID 88-46TS9.

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$		Creates a unique value when $ is appended to the end of the value.
SIS Account list Applet|Name	Test!+!@storedVar!+!Kwd		Appends the dynamic variable value(storedVar) in between static text by using the delimiter !+!.
SIS Account list Applet|Name|Active_Record	john$		Enters the value in to the field for the highlighted row in the list applet.
Opportunity Form Applet - Child|CloseDate	today+10		Enters the today+10 value into the CloseDate Field.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Main Phone Number	LIKE 650		Enters the phone number LIKE 650*.
NULL|Save Query As Applet._SweQueryName	Test1234		Enters the value in to the pop-up input field.
SIS Account list Applet|Name	NULL		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$		Creates a unique value when $ is appended to the end of the value.
SHCE Sales Account List Applet - Mobile|Location|2	Paris		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)		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)		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)		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)		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)		CloseApplet	Invokes the Record Count applet menu item where users must click OK in the pop-up window that appears.
SHCE Sales Account List Applet - Mobile|Get Bookmark URL (SWE)		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			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			Invokes the list applet field object.
SIS Product List Admin Applet|Product Line|3			Invokes the list applet field object in the third row.
SIS Product List Admin Applet|Product Line|Active_Record			Invokes the list applet field object from the current active row.
Opportunity Form Applet - Child|Account2			Invokes the form applet field object.

Target Object	Inputs	Closing Action	Comments
Opportunity Form Applet - Child|Revenue2			Invokes a Form applet field object.
NULL|Revenue2			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			Invokes the Currency pop-up window.
SHCE Sales Opportunity List Applet ReadOnly - Mobile|Account			Invokes the drop-down list applet.
NULL|Currency Code			Invokes the Currency code dialog box.

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
	CORE_UIF;SADMIN;Y		Starts the browser, clears the browser cookies and logs in with the username SADMIN.
	CORE_UIF;SADMIN;N		Starts the browser and logs in with the username SADMIN.
	CORE_UIF;PortalApplication;Y		Handles the login page in the Portal application.
	PHARMAM;LaunchApp;Y		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
	PHARMAM;SPORTER		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)
```

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		Locks the selected column.
SIS Account List Applet|Name	UNLOCK		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		Locks the selected column.
Contact List Applet - Mobile|First Name	UNLOCK		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 applications and mobile.

Target Object	Inputs	Closing Action	Comments
			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).

Signatures

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.
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		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		Selects 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		Queries the record with the specified search criteria.
Contact List Applet|NewQuery	LastName| David,FirstName|Albert		Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|AAAX*		Queries the record with the specified search criteria.

Target Object	Inputs	Closing Action	Comments
Contact List Applet|NewQuery	Work Phone #|650678-0987		Queries the record with the specified search criteria.
Opportunity Form Applet - Child|NewQuery	Opportunity Currency2|USD		Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|AAAX*		Queries the record with the specified search criteria.
Contact List Applet|NewQuery	First Name|NULL		Searches for an empty query (that is for records where First Name is NULL).
Contact List Applet|NewQuery	M/M|Mr.		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		Queries the record with the specified search criteria.
CG Account List Applet - Mobile|NewQuery	Name|abc		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.

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	Query: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	Query: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	Query:Active Last Name|Dupont,Active First Name|Dupont;Remove	RN of OK	Removes an account team from an account in a form applet.
SIS Account List Applet|Sales Rep	Query:NULL;RemoveAll	RN of OK	Removes all accounts in a list applet.
SIS Account List Applet|Sales Rep	Query: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	Query:Name|abc;RemoveAll	Idcancel	Removes all accounts in a list applet.
SHCE Sales Account List Applet - Mobile|Primary Account Address Name	Query: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		Selects the check box.
SIS Account List Applet|Fund Eligible Flag|3	FALSE		Clears the check box in the third row of the list applet.
Opportunity List Applet|Committed	@Variable		Selects or clears the check box depending on the variable value (True or False).
MultiAdd Product On Order|Select All|Select All	True/False		Selects the check box in the column header.

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		Selects the check box for the specified object based on the provided user value.
SHCE Address List Applet - Mobile|SSA Primary Field|3	FALSE		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],Query: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.

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	PopupQuery Pick	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			Selects a PDQ value named <*abc*>.
@Variable			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			Selects a PDQ value named <*abc*>.
@Variable			Selects a PDQ value stored in a variable.

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		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		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		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		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		Selects a value from a drop-down list or LOV (List of Values) in a form or list applet.
SHCE Sales Account List Applet - Mobile|Type	@var		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		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.

Signatures

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		Selects a specified radio button.
NULL|SWE Export Applet.rdbRowsToExport	All Rows In Current Query		Selects a specified radio button.
Sort Order Popup Applet (SWE)|rdbDesc1	@Variable		Selecta 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 ecord in a list applet in desktop applications.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Name	Metropolitan Investments		Selects the specified field value record in the list applet.
AppletSIS Account List Applet|Name|5	NULL		Selects a record (the fifth record) based on the specified row index (5).
AppletSIS Account List Applet|Name|5	NULL		Selects the fifth record in the list applet.
AppletQuote List Applet|Quote Number	@Variable		Selects the specified field value from a variable record in the list applet.
AppletSIS Account List Applet|Name|4	Abc		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		Selects a record in the list applet, according to the specified input value.
AppletSHCE Sales Account List Applet - Mobile|Name|1	FinanceOne Corporation		Selects the specified value in the specified row number.
SHCE Sales Account List Applet - Mobile|Name	FinanceOne Corporation		Matches the value in the visible records.
AppletSHCE Sales Account List Applet - Mobile|Name|1	@accountName		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		Selects the specified value from the toggle control in a list applet.

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			Selects the All Account list view from the Visibility Filter drop-down list.
Contact List Applet|Manager's Contact List View			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_DOWN; ARROW_LEFT; ARROW_UP; 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_DOWN; PAGE_UP; SHIFT LEFT; RIGHT; SPACE; SUBTRACT; TAB; UP; BRACERIGHT; BRACELEFT

Desktop and Mobile 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			Selects all records.
CTRL+S			Saves.
CTRL+TAB			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],DD|MM|YYYY|HH|MM|SS (or)
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	01|01|2015		Chooses a date in the Calendar pop-up window.
Activity List Applet With Navigation|Planned	Today		Sets the current date and time.
Activity List Applet With Navigation|Planned|Active_Record	Today		Sets the current date and time in the field of the active record.
Activity List Applet With Navigation|Planned	05|08|2014|10|17|20		Sets the specified date and time.
Activity List Applet With Navigation|Planned	05|08|2014|10|17|20		Sets the specified date and time.
Opportunity Form Applet - Child|CloseDate2	Today+365		Sets the specified date plus (+)365 days.
Opportunity Form Applet - Child|CloseDate2	Today-365		Sets the specified date minus (-) 365 days.
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR		Sets a specified date in the current year.

Target Object	Inputs	Closing Action	Comments
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR-1		Sets the specified date in the previous year.
Opportunity Form Applet - Child|CloseDate2	28|01|CURRENTYEAR+2		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		Sets the current date and time.
SHCE Sales Opportunity Quote List Applet ReadOnly - Mobile|Start Date	Today+9		Sets the current date and time.
SHCE Quote Entry Applet - Mobile|StartDate	11|5|2014		Sets the specified date in the target object.
SHCE Sales Order Entry List Applet - Mobile|Order Date	11|5|2014|12|12|30		Sets the specified date and time in the target object.
SHCE Sales Order Entry List Applet - Mobile|Order Date	1|8|2014|12|30|20		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)
```

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		Sorts the columns in ascending order.
SIS Account List Applet|Type	DESC		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		Sorts the columns in ascending order.
SHCE Sales Contact List Applet - Mobile|First Name	DESC		Sorts the columns in descending order.

TreeExplorer

You use the TreeExplorer keyword to perform operations in the explorer tree applet, such as, expand and collapse the tree, select any item 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 TreeExlorer 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		Expands the tree structure.
Account Tree Applet|1.9	Collapse		Collapses the tree structure.
Account Tree Applet|1.9	IsNodeExists		Verifies the existence of a node.
Account Tree Applet|1.9	IsNodeExists;@Var;True/False		Verifies the existence of a node in the case of a negative scenario.
Account Tree Applet|1.9	SelectTreeItem		Selects the specified tree item.
Account Tree Applet|1.2	GetTreeChildItemsCount|@var		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)
```

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		Verifies the lock status of the specified column.
LockStatusSIS Account List Applet|Name	UNLOCK		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		Verifies the lock status of the specified column.
LockStatusContact List Applet - Mobile|First Name	UNLOCK		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		Verifies that records are sorted in ascending order.
SIS Account List Applet|Type	DESC		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		Verifies that records are sorted in ascending order.
SHCE Sales Contact List Applet - Mobile|First Name	DESC		Verifies that records are sorted in descending order.

VerifyError

You use the VerifyError keyword to verify the error messages that appear in applications.

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
	'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.

Mobile Examples

The following table describes how to use the VerifyError keyword to verify the error messages that appear in applications in mobile applications (on mobile devices).

Target Object	Inputs	Closing Action	Comments
	'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		Verifies that the image has downloaded completely.
eDetailer Messaging Plan Items Preview List Applet - Mobile|ThumbnailFN3_01132017.jpg	Y		Verifies that the image has downloaded completely.

Mobile Examples

The VerifyFileLoad keyword does not apply to mobile applications.

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		Verifies that focus is on the specified list applet.
SIS Account List Applet|NULL	FALSE		Verifies that focus is not on the specified list applet.
SIS Account List Applet|Row Status|2	TRUE		Verifies that focus is on the specified field in the list applet.
SIS Account Entry Applet|Name	TRUE		Verifies that focus is on the specified field in the form applet.
SIS Account Entry Applet|Name	FALSE		Verifies that focus is not on the specified field in the form applet.
SIS Account List Applet|2	TRUE		Verifies that focus is on the specified row number in the list applet.
SIS Account List Applet|2	FALSE		Verifies that focus is not on the specified row number in the list applet.
SIS Account List Applet|2,3	TRUE		Verifies that focus is on the specified row numbers in the list applet.
Search Admin|NULL	TRUE		Verifies that focus is on the specified view.
Menu-File|Null	TRUE		Verifies that focus is on the application-level menu.
Menu-File|Null	FALSE		Verifies that focus on the application-level menu.
Menu-File|File - Custom Print	TRUE		Verifies that focus is on the application-level menu item.
Menu-File|File - Custom Print	FALSE		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		Verifies that focus is on the specified field in the form applet.
SHCE Sales Account List Applet - Mobile|NULL	TRUE		Verifies that focus is on the specified list applet.
SHCE Sales Account List Applet - Mobile|Name|1	TRUE		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		Verifies that all values (118) are present in the Type drop-down list.
SIS Account List Applet|Type|1	Auto Select:contains,ac		Verifies that there are values that contain the keyword ac.

Target Object	Inputs	Closing Action	Comments
SIS Account List Applet|Type|1	Exists:Customer123;True		Verifies whether a value exists or not.
Opportunity Form Applet - Child|SalesStage2	Auto Select:starts,App		Verifies that there are values that start with App.

Mobile Examples

The VerifyInPicklist keyword does not apply to mobile applications.

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.

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		Verifies the existence of an applet.
NULL|SUI_OPEN_TOOLBAR	NULL;TRUE		Verifies that the toolbar is open.
NULL|SUI_CLOSED_TOOLBAR	NULL;TRUE		Verifies that the toolbar is open.
SIS Account List Applet|SiebVisList	All Accounts Across Organizations;TRUE		Verifies that the item exists in the Visibility list in the list applet.
SIS Account List Applet|DeleteRecord	NULL;TRUE		Verifies that the Delete button exists in the list applet.
SIS Account List Applet|DeleteRecord	Delete;TRUE		Verifies that the Delete button UI name exists in the list applet.
SIS Account Entry Applet|DeleteRecord	NULL;TRUE		Verifies that the Delete button exists in the form applet.
SIS Account List Applet|NewQuery	NULL;TRUE		Checks that the NewQuery button exists in the list applet.
SIS Account List Applet|Type	Account Type;TRUE		Checks that the Column Account Type exists in the list applet.
SIS Account List Applet|QueryComboBox	NULL;TRUE		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		Verifies that the menu item About Record (SWE) exists in the list applet.
SIS Account List Applet|Account Type Code|1	Customer;TRUE		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		Verifies that the application menu item Create Bookmark exists in the file menu.
NULL|SWE Export Applet.rdbRowsToExport	All Rows In Current Query;TRUE		Verifies that the Radio button exists in the pop-up.

Target Object	Inputs	Closing Action	Comments
NULL|Account List View|L2	Accounts List;TRUE		Verifies that the application links exist.
SIS Account List Applet|DeleteRecord	Delete;FALSE		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		Verifies the UI name of the field.
SHCE Sales Account List Applet - Mobile|NewQuery	Query;TRUE		Verifies that the Query button exists in the list applet.
SHCE Sales Account List Applet - Mobile|QuerySrchSpec	Accounts;TRUE		Verifies that items exist in the QuerySearch text box in the list applet.
SHCE Sales Account List Applet - Mobile|DeleteRecord	Delete;TRUE		Verifies that the Delete button UI name exists in the list applet.
SHCE Sales Account List Applet - Mobile|QueryComboBox	NULL;TRUE		Verifies that the QueryComboBox drop-down list exists in the list applet.
SHCE Sales Account List Applet - Mobile|Type|1	Customer;TRUE		Checks the title of the Account Type Code field with the row number in the list applet.
NULL|Account List View	Accounts List;TRUE		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		Verifies if the count is greater than or equal to 6.
SIS Account List Applet|Record Count (SWE)	<=,6		Verifies if record count is less than or equal to 6.
SIS Account List Applet|Record Count (SWE)	<>,6		Verifies if the record count with less than or greater than 6.
SIS Account List Applet|Record Count (SWE)	< ,6		Verifies the record count is less than 6.
SIS Account List Applet|Record Count (SWE)	> ,6		Verifies if the record count is greater than 6.
SIS Account List Applet|Record Count (SWE)	> =,@var		Verifies if the record count is greater than or equal to any variable.
SIS Account List Applet|NULL	> =,2		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 rcords 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		Verifies if the record count is greater than or equal to 6.
SHCE Sales Account List Applet - Mobile|Record Count (SWE)	<,@var		Verifies if the record count is less than any variable.
SHCE Sales Account List Applet - Mobile|NULL	<,2		Verifies if the record count is less than 2.

VerifyState

You use tee VerifyState keyword to verify the state of an object.

Signatures

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 disabled or read only, 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		Verifies whether the button state is enabled or disabled.
SIS Product List Admin Applet|Billable Flag|3	FALSE		Verifies whether the check box field is enabled or disabled.

Target Object	Inputs	Closing Action	Comments
SIS Product Form Applet - ISS Admin|SiebAppletMenu|New Record (SWE)	TRUE		Verifies the state of menu item in the form applet.
Menu-Query|Query - QueryAssist	FALSE		Verifies the state of application level menu items.
NULL|Tree	TRUE		Verifies the state of the application.
NULL|Tab	TRUE		Verifies the state of the application.
NULL|Side Menu	TRUE		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		Verifies the state of the variable and stores the value in @var.
SHCE Sales Account List Applet - Mobile|SiebAppletMenu|Record Count (SWE)	TRUE		Verifies the state of the menu item.

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|E
xpectedd 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. VerifyTopNotication checks for the message in the notification 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_041155918;NULL;Match	Close	Verifies whether or not the top unread message appears in the notification list, and closes the control.
MsgBrdCstIcon	Account_10142015_041155918;NULL;NoMatch	Close	Verifies whether or not the top unread message appears in the notification list, and closes the control.
MsgBrdCstIcon	Account_10142015_041155918;Color|Red;Match	Close	Verifies whether the color of the top unread message in the notification list matches the input value, and closes the control.
MsgBrdCstIcon	Account_10142015_041155918;Color|Red;Match	KeepOpen	Verifies whether the color of the top unread message in the notification list matches the input value, and keeps the control open.
MsgBrdCstIcon	SVP Action|populate Actions;NULL;Match	Close	Verifies whether the top unread message in the notification 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 top unread message appears in the notification 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, etc.

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		Verifies the field value by comparing it with the input value.
Opportunity List Applet|Committed	=,Y		Verifies the check box field value by comparing it with the input value.
Contact List Applet|M/M|2	=,Mr.		Verifies the field value by comparing it with the input value.
SIS Account List Applet|Name	=,@Var1		Verifies the field value by comparing it with the input value.
SIS Account List Applet|Name|2	=,@Var1		Verifies the field value by comparing it with the input value.
Contact List Applet|Last Name	startswith,Ab		Verifies the field value by comparing it with the input value, starts with Ab.
SIS Account List Applet|Name	<>,A*		Verifies the field value by comparing it with the input value, not equal to A.
Opportunity List Applet|Name	=,LIKE Q*		Verifies the field value by comparing it with the input value.
Opportunity List Applet|Name	<>,"Q*		"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		Verifies the field value by comparing it with the input value.
SHCE Sales Account List Applet - Mobile|Name	=,@accountname		Verifies the field value by comparing it with the input value.
SHCE Sales Account List Applet - Mobile|Name	startswith,Ab		Verifies the field value by comparing it with the input value, starts with Ab.
SHCE Sales Account List Applet - Mobile|Name	=,A		Verifies the field value by comparing it with the input value, not equal to A.

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

InvokePerl supports the following signature:

```
InvokePerl(Select Machine|scriptname.pl:@VAR/String[String1,String2....N]/
NULL|Output Variable)
```

where:

- *Select Machine* is the Siebel Server or the Test Harness machine.

  Siebel Server reads the server credentials from the XML file and executes the Perl file on the server machine. The Test Harness machine executes the script name in the Perl file locally.

- *@VAR/String[String1,String2....N* are the supported variables or strings passed to the Perl file.

- *@Output Variable* is the supported variable obtained from the Perl file.

Declaring PerlExecStatus in the Perl File

You must declare the PerlExecStatus variable in the Perl file. For example, at the end of the Perl file, print the PerlExecStatus as follows:

```
print"PerlExecStatus = $PerlExecStatus";
```

Use the following syntax in the Perl file to save a value to PerlExecStatus:

```
print"\@invokePerl1=$ARGV[1].";

print"\@invokePerl2=$ARGV[0].";

print"perlexecstatus = $PerlExecStatus";
```

Test Harness Folder Structure

The Perl file to be executed must be placed in the invokePerl folder on the Test Harness machine.

The Exefiles folder must be placed on the Test Harness Machine.

INI File Structure

An eample of the INI file structure is as follows:

```
[SERVER]

SERVER_MACHINE=macine-name

----SERVER_MACHINE will be passed as $ARGV[0] to the perl file

SERVER_LOGIN=ORADEV\\userid|password

----SERVER_LOGIN will be passed as $ARGV [1]&& $ARGV [2] to the perl file

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

DBSERVER_LOGIN=SADMIN|MSSQL

----DBSERVER_Login will be passed as $ARGV [8]&& $ARGV [9] to the perl file

CLIENT_MACHINE=machinename

----CLIENT_MACHINE will be passed as $ARGV [10] to the perl file-TestHarness Path
$ARGV[11]

ENTSERVER_AND_PORT=servername:port

----ENTSERVER_AND_PORT will be passed as $ARGV[12]

SERVER_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]


[PERLPATH]

PERL_PATH= \\\\servername\\install\\PERL_UTILS\\Perl1\\bin\\perl.exe
```

Examples

The following table describes how to use the InvokePerl keyword when configuring Siebel Tools and Siebel Server.

Target Object	Input	Closing Action
	SiebelServer Machine|test.pl:NULL|NULL	
	SiebelServer Machine|test.pl:value1,value2|NULL	
	SiebelServer Machine|test.pl:value1|NULL	
	SiebelServer Machine|test.pl:value1,@invokePerl|NULL	
	SiebelServer Machine|test.pl:@invokePerl|NULL	
	SiebelServer Machine|test.pl:@invokePerl,@invokePerl|NULL	

Target Object	Input	Closing Action
	SiebelServer Machine|test.pl:@invokePerl,@invokePerl|@invokePerl1	
	SiebelServer Machine|test.pl:@invokePerl,@invokePerl|@invokePerl1,@invokePerl2	
	TestHarness Machine|test.pl:NULL|NULL	
	TestHarness Machine|test.pl:value1,value2|NULL	
	TestHarness Machine|test.pl:value1,@invokePerl|NULL	
	TestHarness Machine|test.pl:@invokePerl,@invokePerl|@invokePerl1	
	TestHarness Machine|test.pl:@invokePerl,@invokePerl|@invokePerl1,@invokePerl2E	

ToolsConfig

You use the ToolsConfig keyword to configure Siebel Tools by importing pre-existing sif files that contain the customizations.

Signature

The ToolsConfig keyword supports the following signature:

```
ToolsConfig(Sif1,Sif2|Merge/Overwrite|Projects to be locked in db)
```

where:

- ToolsConfig (revert|Projects) must be unlocked in the database.
- Sif1,Sif2....Sifn are the Sif files to be copied from the Test Harness Sif folder to the client machine.
- Revert means to go back to the original SRF.

Actions

ToolsConfig is a composite keyword, which includes the following actions:

1. Stop the Siebel Server.
2. Back up the SRF.
3. Import the SIF File.
4. Compile the SRF.
5. Start the Siebel Server.

INI File Structure

An example of the INI file structure is as follows:

```
[SIEBEL_TOOLS]

SIEB_SERVER_PATH=/export/home/../../ses/siebsrvr/objects/enu

TOOLS_PATH=C:\\..


[CLIENT]

SERVER_MACHINE=machine name

CLIENT_MACHINE=machine name

SERVER_LOGIN=domain\\userid|password

SERVER_DB_TYPE=MSSQL/ORACLE

SERVER_OS_TYPE=UNIX/WINDOWS

SERVER_DB_NAME=server name

SERVER_DB_TABLEOWNER=db owner name

DBSERVER_AND_PORT=servername,port

TOOLS_MACHINE=machine name

TOOLS_LOGIN=domain\\username|password

[PERLPATH]

PERL_PATH= \\\\servername\\install\\PERL_UTILS\\Perl1\\bin\\perl.exe
```

Examples

The following table describes how to use the InvokePerl keyword when configuring Siebel Tools and Siebel Server.

Target Object	Inputs	Closing Action	Comments
	RTC.sif|Merge|Account,Account (SSE),Activity,Activity SSE),Communication Administration,Activity ScreenHomePage		Imports and compiles single sif file.

Target Object	Inputs	Closing Action	Comments
	RTC.sif ,RTC1.sif,RTC2.sif|Overwrite|Account,Account (SSE),Activity,Activity (SSE),Communication Administration,Activity ScreenHomePage		Imports and compiles multiple sif file.
	revert|Account,Account (SSE)		Cancels the changes and reverts to the original SRF.

ServerConfig

You use the ServerConfig keyword to configure and start the Siebel server.

Signature

The ServerConfig keyword supports the following signatures:

```
ServerConfig(Restart)

ServerConfig(commandsFile.txt)
```

where:

- Server is the server machine where the Siebel Server is running. The machine details are available in the Server section of the INI file.
- commandsFile.txt is the text file containing the list of commands to be executed in the Server Manager.

NOTE: If you use the Launch keyword after the ServerConfig keyword, then ClearBrowser must be used in the Launch after ServerConfig.

Test Harness Folder Structure

All Perl files and command (cmd.txt) files related to the ServerConfig keyword must be placed in serverConfig folder on the Test Harness machine.

The Exefiles folder must be placed on the Test Harness machine.

Actions

ServerConfig is a composite keyword, which includes the following:

1. Start server manager.
2. Command to be executed.
3. Stop the World Wide Web publishing service.

4 Stop the gateway server.

5 Stop the Siebel Server.

6 Start the gateway server.

7 Start the Siebel Server.

8 Start the World Wide Web publishing service

The following procedure shows you how to start the server.

To start the server

1 Stop the World Wide Web publishing service.

2 Stop the gateway server.

3 Stop the Siebel Server.

4 Start the gateway server.

5 Start the Siebel Server.

6 Start the World Wide Web publishing service.

INI File Structure

An example of the INI file structure is as follows:

```
[SERVER] - WINDOWS

SERVER_MACHINE=machine name

SERVER_LOGIN=ORADEV\\userid|password

SERVER_OS_TYPE=WINDOWS

SERVER_DB_TYPE=MSSQL

SERVER_DB_NAME=db name

DBSERVER_AND_PORT=dbservername,portnumber

DB_TABLE_OWNER=XYZ

DBSERVER_LOGIN=user1|user2

CLIENT_MACHINE=machine name

ENTSERVER_AND_PORT=servername:port

SERVER_PATH = C:\\build\\ses\\siebsrvr\\BIN

[PERLPATH]

PERL_PATH= \\\\servername\\install\\PERL_UTILS\\Perl1\\bin\\perl.exe
```

Examples

The following table describes how to use the InvokePerl keyword when configuring Siebel Tools and Siebel Server.

Target Object	Inputs	Closing Action	Comments
	Restart		Start the Server.
	cmd.txt		Connect to the server manager and execute the commands in the cmd.txt file.
	Restart|cmd.txt		Connect to the server manager and execute the commands in the cmd.txt file and start 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
- Oracle Policy Automation (Third Party Control)
- Find Result Pane
- Search Result Pane
- Gantt DragAndDrop
- Charts
- Promotion Designer
- Gantt (Diary)
- Task Pane items
- Gantt Chart - Promotion List

B 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.

C Reports

Reports provide the Pass and Fail results with detailed step level comments and screenshots.

Supported Features

The following features are provided to retrieve the keyword automation results:

- A column (called Screenshot) in the test script to capture the screenshot.
- A search text box in the Test Set Result to filter the results based on the following values: Fail, Pass, and Scrumpy Id values.
- Ability to specify whether to capture a screenshot or not (for Pass and Fail test results) as follows:
  - Specify Y in the Screenshot column if you want to capture a screen shot.
  - Leave the Screenshot column empty if you do not want to capture a screenshot.
- Ability to enter the value Pass or Fail in the search text box to get the pass results or fail results accordingly.

Note the following:

- If a screenshot is required for the Pass or Fail test cases, then the screen or view to be captured must be in Active mode.
- The executable client machine must open in InetPub.

Example

The following table describes how to run a report for the GoToView keyword and capture a screenshot.

Table 12.

Keyword	Target Object	Inputs	Closing Action	Screenshot	Comments
Launch		CORE_UIF;user1;Y			Screenshot is not captured.
GoToView	NULL|Accounts Screen|L1			Y	Captures screenshot.
GoToView	NULL|Account List View|L2			Y	Captures screenshot.

Index

U

V