Oracle® Scripting

Developer's Guide

Release 12.2

Part No. E49094-01

September 2013

ORACLE

Oracle Scripting Developer's Guide, Release 12.2

Part No. E49094-01

Primary Author: Dhanya Menon

Contributing Author: Ashita Mathur

Contents

2 Customizing Oracle Scripting

3 Predefined Script Author Commands

Index

Send Us Your Comments

Oracle Scripting Developer's Guide, Release 12.2

Part No. E49094-01

Oracle welcomes customers' comments and suggestions on the quality and usefulness of this document. Your feedback is important, and helps us to best meet your needs as a user of our products. For example:

- Are the implementation steps correct and complete?
- Did you understand the context of the procedures?
- Did you find any errors in the information?
- Does the structure of the information help you with your tasks?
- Do you need different information or graphics? If so, where, and in what format?
- Are the examples correct? Do you need more examples?

If you find any errors or have any other suggestions for improvement, then please tell us your name, the name of the company who has licensed our products, the title and part number of the documentation and the chapter, section, and page number (if available).

Note: Before sending us your comments, you might like to check that you have the latest version of the document and if any concerns are already addressed. To do this, access the new Oracle E-Business Suite Release Online Documentation CD available on My Oracle Support and www.oracle.com. It contains the most current Documentation Library plus all documents revised or released recently.

Send your comments to us using the electronic mail address: appsdoc_us@oracle.com

Please give your name, address, electronic mail address, and telephone number (optional).

If you need assistance with Oracle software, then please contact your support representative or Oracle Support Services.

If you require training or instruction in using Oracle software, then please contact your Oracle local office and inquire about our Oracle University offerings. A list of Oracle offices is available on our Web site at www.oracle.com.

Preface

Intended Audience

Welcome to Release 12.2 of the *Oracle Scripting Developer's Guide.*

See Related Information Sources on page x for more Oracle E-Business Suite product information.

Documentation Accessibility

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

Access to Oracle Support

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

Structure

Related Information Sources

Integration Repository

The Oracle Integration Repository is a compilation of information about the service endpoints exposed by the Oracle E-Business Suite of applications. It provides a complete catalog of Oracle E-Business Suite's business service interfaces. The tool lets users easily discover and deploy the appropriate business service interface for integration with any system, application, or business partner.

The Oracle Integration Repository is shipped as part of the E-Business Suite. As your instance is patched, the repository is automatically updated with content appropriate for the precise revisions of interfaces in your environment.

You can navigate to the Oracle Integration Repository through Oracle E-Business Suite Integrated SOA Gateway.

Online Documentation

All Oracle E-Business Suite documentation is available online (HTML or PDF).

- **PDF** - See the Oracle E-Business Suite Documentation Library for current PDF documentation for your product with each release. The Oracle E-Business Suite Documentation Library is also available on My Oracle Support and is updated frequently

- **Online Help** - Online help patches (HTML) are available on My Oracle Support.

- **Release Notes** - For information about changes in this release, including new features, known issues, and other details, see the release notes for the relevant product, available on My Oracle Support.

- **Oracle Electronic Technical Reference Manual -** The Oracle Electronic Technical Reference Manual (eTRM) contains database diagrams and a detailed description of database tables, forms, reports, and programs for each Oracle E-Business Suite product. This information helps you convert data from your existing applications and integrate Oracle E-Business Suite data with non-Oracle applications, and write custom reports for Oracle E-Business Suite products. The Oracle eTRM is available on My Oracle Support.

Guides Related to All Products

Oracle E-Business Suite User's Guide

This guide explains how to navigate, enter data, query, and run reports using the user interface (UI) of Oracle E-Business Suite. This guide also includes information on setting

user profiles, as well as running and reviewing concurrent programs.

You can access this guide online by choosing "Getting Started with Oracle Applications" from any Oracle E-Business Suite product help file.

Guides Related to This Product

Oracle Scripting Implementation Guide

This guide describes how to create modify, and deploy scripts to the applications database using Script Author. It also describes how to create and modify survey campaigns, cycles, and deployments, how to set up invitations and reminders for the survey campaign, and how to view responses for active or closed survey campaigns. To deploy scripts to applications, you must integrate Oracle Scripting with Oracle TeleSales, Oracle TeleService, Oracle Marketing, and Oracle One-to-One Fulfillment.

Oracle Scripting User Guide

Oracle Scripting provides enterprises with scripts of questions and answers to help agents gather data, provide information to customers, conduct web-based marketing research surveys, and help web customers make a decision. You use the Script Author to build a script for execution, the Scripting Administration console to manage scripting files, the Survey Administration console to administer survey campaigns, and the Scripting Engine to execute scripts.

Installation and System Administration

Oracle Alert User's Guide

This guide explains how to define periodic and event alerts to monitor the status of your Oracle E-Business Suite data.

Oracle E-Business Suite Concepts

This book is intended for all those planning to deploy Oracle E-Business Suite Release 12.2, or contemplating significant changes to a configuration. After describing the Oracle E-Business Suite architecture and technology stack, it focuses on strategic topics, giving a broad outline of the actions needed to achieve a particular goal, plus the installation and configuration choices that may be available.

Oracle E-Business Suite CRM System Administrator's Guide

This manual describes how to implement the CRM Technology Foundation (JTT) and use its System Administrator Console.

Oracle E-Business Suite Developer's Guide

This guide contains the coding standards followed by the Oracle E-Business Suite development staff. It describes the Oracle Application Object Library components needed to implement the Oracle E-Business Suite user interface described in the *Oracle E-Business Suite User Interface Standards for Forms-Based Products*. It also provides information to help you build your custom Oracle Forms Developer forms so that they integrate with Oracle E-Business Suite. In addition, this guide has information for customizations in features such as concurrent programs, flexfields, messages, and logging.

Oracle E-Business Suite Installation Guide: Using Rapid Install

This book is intended for use by anyone who is responsible for installing or upgrading Oracle E-Business Suite. It provides instructions for running Rapid Install either to carry out a fresh installation of Oracle E-Business Suite Release 12.2, or as part of an upgrade to Release 12.2.

Oracle E-Business Suite Maintenance Guide

This guide contains information about the strategies, tasks, and troubleshooting activities that can be used to help ensure an Oracle E-Business Suite system keeps running smoothly, together with a comprehensive description of the relevant tools and utilities. It also describes how to patch a system, with recommendations for optimizing typical patching operations and reducing downtime.

Oracle E-Business Suite Security Guide

This guide contains information on a comprehensive range of security-related topics, including access control, user management, function security, data security, and auditing. It also describes how Oracle E-Business Suite can be integrated into a single sign-on environment.

Oracle E-Business Suite Setup Guide

This guide contains information on system configuration tasks that are carried out either after installation or whenever there is a significant change to the system. The activities described include defining concurrent programs and managers, enabling Oracle Applications Manager features, and setting up printers and online help.

Oracle E-Business Suite User Interface Standards for Forms-Based Products

This guide contains the user interface (UI) standards followed by the Oracle E-Business Suite development staff. It describes the UI for the Oracle E-Business Suite products and tells you how to apply this UI to the design of an application built by using Oracle Forms.

Other Implementation Documentation

Oracle Approvals Management Implementation Guide

This guide describes transaction attributes, conditions, actions, and approver groups that you can use to define approval rules for your business. These rules govern the process for approving transactions in an integrated Oracle application. You can define approvals by job, supervisor hierarchy, positions, or by lists of individuals created either at the time you set up the approval rule or generated dynamically when the rule is invoked. You can learn how to link different approval methods together and how to run approval processes in parallel to shorten transaction approval process time.

Oracle Diagnostics Framework User's Guide

This guide contains information on implementing, administering, and developing diagnostics tests for Oracle E-Business Suite using the Oracle Diagnostics Framework.

Oracle E-Business Suite Flexfields Guide

This guide provides flexfields planning, setup and reference information for the Oracle E-Business Suite implementation team, as well as for users responsible for the ongoing maintenance of Oracle E-Business Suite product data. This guide also provides information on creating custom reports on flexfields data.

Oracle E-Business Suite Integrated SOA Gateway Implementation Guide

This guide explains the details of how integration repository administrators can manage and administer the entire service enablement process based on the service-oriented architecture (SOA) for both native packaged public integration interfaces and composite services - BPEL type. It also describes how to invoke Web services from Oracle E-Business Suite by working with Oracle Workflow Business Event System, manage Web service security, and monitor SOAP messages.

Oracle E-Business Suite Integrated SOA Gateway User's Guide

This guide describes how users can browse and view the integration interface definitions and services that reside in Oracle Integration Repository.

Oracle E-Business Suite Multiple Organizations Implementation Guide

This guide describes how to set up multiple organizations and the relationships among them in a single installation of an Oracle E-Business Suite product such that transactions flow smoothly through and among organizations that can be ledgers, business groups, legal entities, operating units, or inventory organizations. You can use this guide to assign operating units to a security profile and assign this profile to responsibilities such that a user can access data for multiple operating units from a single responsibility. In addition, this guide describes how to set up reporting to generate reports at different

levels and for different contexts. Reporting levels can be ledger or operating unit while reporting context is a named entity in the selected reporting level.

Oracle e-Commerce Gateway Implementation Guide

This guide describes implementation details, highlighting additional setup steps needed for trading partners, code conversion, and Oracle E-Business Suite. It also provides architecture guidelines for transaction interface files, troubleshooting information, and a description of how to customize EDI transactions.

Oracle e-Commerce Gateway User's Guide

This guide describes the functionality of Oracle e-Commerce Gateway and the necessary setup steps in order for Oracle E-Business Suite to conduct business with trading partners through Electronic Data Interchange (EDI). It also describes how to run extract programs for outbound transactions, import programs for inbound transactions, and the relevant reports.

Oracle iSetup User's Guide

This guide describes how to use Oracle iSetup to migrate data between different instances of the Oracle E-Business Suite and generate reports. It also includes configuration information, instance mapping, and seeded templates used for data migration.

Oracle Product Hub Implementation Guide

This guide explains how to set up hierarchies of items using catalogs and catalog categories and then to create user-defined attributes to capture all of the detailed information (such as cost information) about an object (such as an item or change order). It also explains how to set up optional features used in specific business cases; choose which features meet your business' needs. Finally, the guide explains the set up steps required to link to third party and legacy applications, then synchronize and enrich the data in a master product information repository.

Oracle Product Hub User's Guide

This guide explains how to centrally manage item information across an enterprise, focusing on product data consolidation and quality. The item information managed includes item attributes, categorization, organizations, suppliers, multilevel structures/bills of material, packaging, changes, attachments, and reporting.

Oracle Web Applications Desktop Integrator Implementation and Administration Guide

Oracle Web Applications Desktop Integrator brings Oracle E-Business Suite functionality to a spreadsheet, where familiar data entry and modeling techniques can be used to complete Oracle E-Business Suite tasks. You can create formatted spreadsheets on your desktop that allow you to download, view, edit, and create Oracle

E-Business Suite data, which you can then upload. This guide describes how to implement Oracle Web Applications Desktop Integrator and how to define mappings, layouts, style sheets, and other setup options.

Oracle Workflow Administrator's Guide

This guide explains how to complete the setup steps necessary for any Oracle E-Business Suite product that includes workflow-enabled processes. It also describes how to manage workflow processes and business events using Oracle Applications Manager, how to monitor the progress of runtime workflow processes, and how to administer notifications sent to workflow users.

Oracle Workflow Developer's Guide

This guide explains how to define new workflow business processes and customize existing workflow processes embedded in Oracle E-Business Suite. It also describes how to define and customize business events and event subscriptions.

Oracle Workflow User's Guide

This guide describes how Oracle E-Business Suite users can view and respond to workflow notifications and monitor the progress of their workflow processes.

Oracle XML Gateway User's Guide

This guide describes Oracle XML Gateway functionality and each component of the Oracle XML Gateway architecture, including Message Designer, Oracle XML Gateway Setup, Execution Engine, Message Queues, and Oracle Transport Agent. It also explains how to use Collaboration History that records all business transactions and messages exchanged with trading partners.

The integrations with Oracle Workflow Business Event System, and the Business-to-Business transactions are also addressed in this guide.

Oracle XML Publisher Administration and Developer's Guide

Oracle XML Publisher is a template-based reporting solution that merges XML data with templates in RTF or PDF format to produce outputs to meet a variety of business needs. Outputs include: PDF, HTML, Excel, RTF, and eText (for EDI and EFT transactions). Oracle XML Publisher can be used to generate reports based on existing Oracle E-Business Suite report data, or you can use Oracle XML Publisher's data extraction engine to build your own queries. Oracle XML Publisher also provides a robust set of APIs to manage delivery of your reports via e-mail, fax, secure FTP, printer, WebDav, and more. This guide describes how to set up and administer Oracle XML Publisher as well as how to use the Application Programming Interface to build custom solutions. This guide is available through the Oracle E-Business Suite online help.

Oracle XML Publisher Report Designer's Guide

Oracle XML Publisher is a template-based reporting solution that merges XML data with templates in RTF or PDF format to produce a variety of outputs to meet a variety of business needs. Using Microsoft Word or Adobe Acrobat as the design tool, you can create pixel-perfect reports from the Oracle E-Business Suite. Use this guide to design your report layouts. This guide is available through the Oracle E-Business Suite online help.

Training and Support

Training

Oracle offers a complete set of training courses to help you master your product and reach full productivity quickly. These courses are organized into functional learning paths, so you take only those courses appropriate to your job or area of responsibility.

You have a choice of educational environments. You can attend courses offered by Oracle University at any of our many Education Centers, you can arrange for our trainers to teach at your facility, or you can use Oracle Learning Network (OLN), Oracle University's online education utility. In addition, Oracle training professionals can tailor standard courses or develop custom courses to meet your needs. For example, you may want to use your organization structure, terminology, and data as examples in a customized training session delivered at your own facility.

Support

From on-site support to central support, our team of experienced professionals provides the help and information you need to keep your product working for you. This team includes your Technical Representative, Account Manager, and Oracle's large staff of consultants and support specialists with expertise in your business area, managing an Oracle server, and your hardware and software environment.

Do Not Use Database Tools to Modify Oracle E-Business Suite Data

Oracle STRONGLY RECOMMENDS that you never use SQL*Plus, Oracle Data Browser, database triggers, or any other tool to modify Oracle E-Business Suite data unless otherwise instructed.

Oracle provides powerful tools you can use to create, store, change, retrieve, and maintain information in an Oracle database. But if you use Oracle tools such as SQL*Plus to modify Oracle E-Business Suite data, you risk destroying the integrity of your data and you lose the ability to audit changes to your data.

Because Oracle E-Business Suite tables are interrelated, any change you make using an Oracle E-Business Suite form can update many tables at once. But when you modify Oracle E-Business Suite data using anything other than Oracle E-Business Suite, you

may change a row in one table without making corresponding changes in related tables. If your tables get out of synchronization with each other, you risk retrieving erroneous information and you risk unpredictable results throughout Oracle E-Business Suite.

When you use Oracle E-Business Suite to modify your data, Oracle E-Business Suite automatically checks that your changes are valid. Oracle E-Business Suite also keeps track of who changes information. If you enter information into database tables using database tools, you may store invalid information. You also lose the ability to track who has changed your information because SQL*Plus and other database tools do not keep a record of changes.

1

Understanding Script Author Commands

This chapter covers the following topics:

- Action Types and Commands
- Command Types

Action Types and Commands

You can add functionality to a graphical script at runtime by defining commands in the Script Author.

This section includes the following topics:

Where Commands are Defined in the Script Author

Commands can be defined for a graphical script in the Script Author in the following ways:

- Associated with Script Author objects using various action types
- Associated with "questions" (also referred to as answer definitions) using the data dictionary
- Associated with panel text using an embedded value
- Associated with a Shortcut button in the Shortcut button bar (for agent interface operations only)
- Associated as the return value or parameter of another command

Commands can also be defined for a wizard script by selecting options in the Define

Question Detail wizard window. All commands are associated at the question within a panel level. Commands available from the wizard script include the following:

- Constant command (referred to as the "Default Value" the question)
- Validation that an answer is provided at runtime (for example, was any input provided by the user)
- Validation for an integer provided as an answer at runtime (validation includes verifying that the answer provided is an integer only, or is an integer within a provided range, or is a valid date, or is a valid date not in the past)

Action Types

Action types provide a way to associate a Script Author command with a graphical object in the script (a panel, group, block, branch, or the global script itself). Commands are associated with Script Author graphical objects using various action types, based on the object.

Using an action type, you can associate an existing command from the command library, or create a new command. Action types include actions, delete actions, pre-actions, post-actions, and application programming interface (API) actions.

Actions execute at runtime when the object is reached. Actions are properties of branches only, and represent the only command occurring for the branch, other than routing the script to the next object according to properties of the branch type. Actions can include any command type except an API action.

Delete actions execute at runtime when the object containing the delete action is reached. The delete action causes a row to be deleted from the database, based on what is currently selected in the Scripting cursor. Nothing else in the script or transaction is deleted.

Pre-actions execute at runtime when the object containing the pre-action is reached and prior to:

- Displaying any panel objects (text, graphics, or questions);
- Collecting any answers, for a panel;
- Executing any commands associated with the group or block and contained in the object;
- Displaying subgraphs contained in groups or blocks.

Post-actions execute at runtime when the object containing the post-action is reached, and *after*:

- Answers are retrieved and written to the blackboard (for a panel), and prior to reaching or processing the next object in the script;

- Commands associated with a group or block and contained in the object are executed, and prior to reaching or processing the next object in the script;
- Subgraphs contained in groups or blocks are reached and processed by the script, and prior to returning flow to the calling parent graph.

Delete actions execute at runtime when the object containing the delete action is reached. The delete action causes a row to be deleted from the database, based on what is currently selected in the Scripting cursor. Nothing else in the script or transaction is deleted.

API actions are associated with blocks only and execute at runtime immediately following the execution of any pre-actions to the API block, and prior to the display of any panel content or execution of any objects contained within the API block.

The table below indicates which action types are associated with each object, and describes when that command is executed for each object.

Object	Action Type	Description
Default branch	Action	Command executes at runtime upon reaching this branch in the flow of the script.
Distinct branch	Action	Command executes at runtime upon reaching this branch in the flow of the script.
Conditional branch	Action	Command executes at runtime upon reaching this branch in the flow of the script.
Indeterminate branch	Action	Command executes at runtime upon reaching this branch in the flow of the script.
Global script	Pre-action	Command executes at runtime prior to displaying any objects in the user interface or evaluating any commands associated with objects in the script.
Global script	Post-action	Command executes at runtime immediately following the display of the last panel or object reached in the flow and prior to ending the interaction.
Panel	Pre-action	Command executes at runtime prior to displaying any panel content or answers contained in the panel.
Panel	Post-action	Command executes at runtime immediately following action by the user to continue to the next panel in the flow, and prior to evaluation of the next object.

Object	Action Type	Description
Group	Pre-action	Command executes at runtime prior to display of any panel content or execution of any object contained in the group. This is true even if reaching the group via a shortcut.
Group	Post-action	Command executes at runtime immediately following the display of any panel content or execution of any object contained in the group. and prior to evaluation of the next object in the flow of the script.
Block	Pre-action	Command executes at runtime prior to display of any panel content or execution of any database or API actions contained within the block.
Block	Post-action	Command executes at runtime immediately following the display of any panel content or execution of any database or API actions, and prior to evaluation of the next object in the flow of the script.
Block	API action	Command executes at runtime immediately following the execution of any pre-actions to the API block, and prior to the display of any panel content or execution of any objects contained within the API block.

Guidelines

- Actions can be performed on branches anywhere in a script, except between the Start node and the first object in the script.
- Delete actions can be included in a script as an action to a branch, or as a pre-action or post-action to any other object in a script.
- Pre-actions and post-actions can be performed on panels, blocks, groups, or the global script.
- API actions can be associated with a block of type API block.

Related Topics

For more information, see the Defining Commands section in the *Oracle Scripting User Guide.*

Best Practice Java Methods, page 2-9

Command Types

You can create the following types of commands for a graphical script in Script Author:

- Java Commands, page 1-5
- PL/SQL Commands, page 1-10
- Blackboard Commands, page 1-11
- Forms Commands, page 1-14
- Constant Commands, page 1-33
- Delete Actions, page 1-34

Java Commands

Using a Java command, you can execute any public method in an appropriately referenced Java class. The Java command is executed when the appropriate object is reached in the flow of the script at runtime.

Guidelines

- Commands are associated with objects using several approaches. For more information on where Java or other commands can be placed into a script, see Where Commands Are Defined in the Script Author, page 1-1.
- You can reference existing best practice Java methods (see Best Practice Java Methods, *Oracle Scripting Developer's Guide*), or create custom Java methods.
- Java commands that invoke Oracle Scripting APIs require the server proxy bean as a parameter. Check the requirements of a specific API.
- No provision is made to create or compile custom Java methods in the Script Author. All custom Java must be created by certified, trained and knowledgeable individuals using external applications or utilities.
- The Java class you reference must have the appropriate import statements to access Oracle Scripting-specific classes. For more information, see Section 1.2.1.5, "Importing Scripting Classes into Java Source Files".
- The method within the Java class that you reference in the Script Author must be fully qualified. For more information, see Section 1.2.1.1, "Referencing Java Methods in Script Author".
- For Java commands to execute as intended, the referenced Java class and method

must be appropriately compiled, packaged, and available to the class loader of the Java Virtual Machine (JVM) in the Scripting session where it will be executed.

- You can deploy custom Java archives to the applications database using the Scripting Administration console, where the code archive can be made globally available to all scripts, or mapped to one or more specific scripts, resulting in the referenced methods being available at script runtime.

- For specific instructions on defining a Java command in the Script Author, see Invoking a Public Method in a Java Class in Oracle Scripting User Guide.

Referencing Java Methods in Script Author

Java Class Path Specification

To specify a Java command in the Script Author, you must enter appropriate values in the Command Info area of the Command Properties window:

- In the Name field, enter the command name.

  This is the name Oracle Scripting uses to reference this command. If stored in the database as a reusable command, this is the name that identifies this specific Script Author command.

- In the Command field, enter the command path.

  This is a string that includes the Java class name for the command (including the fully qualified class path), followed by two colons, and the specific Java method of the class.

Fully Qualified Class Paths

If the class is written as part of a package, this path must be fully qualified. To fully qualify a class path, begin with the top level of the package, adding a period between each package level and between the last level of the package and the class name.

Example

For example, if the package is myprojects.myclasses, the class is TestClass, and the method name is testMethod, the values required to specify this command in the Command Info area of the Command Properties window are testMethod (for the Name field) and myprojects.myclasses.TestClass::testMethod (for the Command field).

Referencing Best Practice Java Methods in Script Author

Oracle Applications includes Java methods written specifically for Oracle Scripting to quickly enable frequently requested script runtime functionality. These best practice Java methods are organized into three classes: ScriptUtil, NodeDefault, and NodeValidation. Each is in a package (oracle.apps.ies.bestpractice), and must be fully qualified.

Example

For example, if the package is oracle.apps.ies.bestpractice, the class is NodeValidation, and the method name is checkNullValue, the values required to specify this command in the Command Info area of the Command Properties window are checkNullValue (for the Name field) and oracle.apps.ies.bestpractice.NodeValidation::checkNullValue (for the Command field).

To illustrate this concept, consider a custom Java method that checks for a null value. This Java method, checkNullValue, is useful to ensure that a value is returned for an executed PL/SQL command.

In the first example, this Java method is written in custom code with no package. The checkNullValue method is in the NodeValidation class file. The Command field to specify this command is NodeValidation::checkNullValue.

To use the same method from the best practice Java, the value of the Command field must be fully qualified to reference the Java method in its package. Thus, the Command field to specify this command using best practice Java is oracle.apps.ies.bestpractice.NodeValidation::checkNullValue.

This fully qualified class path ensures the Java class loader has access to the NodeValidation class and all its methods, including checkNullValue.

Thus, best practice Java methods enable an enterprise to substantially reduce or even eliminate writing, deploying and testing custom Java. This reduces development time and costs, providing a faster time to market and enabling code reuse, while additionally reducing the requirement for Apache Web server configuration file customization.

Guidelines

- Do not attempt to modify APPS.ZIP to include custom Java methods. This could impact the behavior of Oracle Scripting and other Oracle applications.
- Best practice Java methods can be used to support all Oracle Scripting graphical scripts, whether executed in the agent interface or as a Web-based survey.
- When using best practice Java methods to extend Oracle Scripting functionality, review the Java method signature, which includes:
  - Scope modifier
  - Return type
  - Java method name
  - Arguments or parameters (if any)

  You will also need to know what exceptions are thrown, if any. This information is available in the documentation for the best practices Java methods.
- Arguments needed by best practice methods can be passed as parameters to a

command in the Script Author. An example showing how to do this is described in Referencing Best Practice Java in a Command, page 2-20.

Passing the Proxy Bean as a Parameter to a Java Command

When developing or referencing custom Java code, many commands require you to pass the server proxy bean. The server proxy bean is the object that serves as a proxy on behalf of custom code, allowing the custom code to invoke Scripting APIs.

The proxy bean is instantiated within the same Java Virtual Machine (JVM) as the session object. This Java bean can be referenced by Java command objects created as part of a script (which are also instantiated within the JVM of the Session object). Thus, the proxy bean can be used by these Java command objects to call APIs on the Session object.

The proxy bean and its APIs and events serve as the sole interface between an external application and the Scripting Engine, which resides (in the Apache mid-tier architecture) in the Apache JServ.

You must include the proxy bean as a parameter for any Java command using Oracle Scripting APIs. Use this procedure to pass the server proxy bean as a parameter to a Java command in a graphical Script Author script.

Prerequisites

- Identify the name and location in a graphical script of the appropriate Java method.
- The Java method must require the proxy bean to invoke an Oracle Scripting API.

Responsibility

Scripting Administrator

Steps

1. Using the Script Author, open a new or existing graphical script.
2. Navigate to the object containing the appropriate Java command.
3. Open the Command window.
4. In the Parameters area, verify that this command does not already pass the server proxy bean. If you see **Command parameter: Proxy**, this procedure is not required.
5. From the Parameters area, click **Add**.

   The Parameters window appears.
6. Enter the appropriate information to invoke the server proxy bean.
7. In the Parameters window, in the Name field, type **Proxy** in exact case.

Note: You do not need to specify a class or value. However, the value you type into the Name field of the Parameters window is case-specific. Ensure the P is upper case and the remaining letters are in lower case.

8. In the Parameters window, click **OK**.

   The Parameters window closes.

9. In the Command window, click **OK**.

10. Save your work.

Guidelines

Some custom Java methods call or instantiate the server proxy bean as pb, other methods as Proxy. Regardless, you can pass this particular parameter by referencing only the bean name (Proxy).

Standard Java Naming and Usage Conventions

Oracle recommends that you adhere to standard Java naming and usage conventions. For example:

- Package names are all lower-case. To fully qualify a class path, begin with the top level of the package, adding a period between each package level and between the last level of the package and the class name.

- Class names begin with an upper-case letter. If the name is a concatenated string of words, the initial letter of each subsequent word is capitalized. The remaining characters are lower-case.

  Since a class is a set of related Java methods that share common characteristics, one manner of naming the class is to describe those characteristics. For projects with a single class, you may wish to name the class after the function of the script or the name of the campaign.

- Method names begin with a lower-case letter. If the name is a concatenated string of words, the initial letter of each subsequent word is capitalized. The remaining characters are lower-case.

  The method name should reflect what the method does. If a procedure, using a verb-plus-object name is standard (for example, setValue). If a function, the same is true, usually descriptive of the return value (for example, getName).

- Variables and return values, like method names, typically begin with a lower-case letter. If the name is a concatenated string of words, the initial letter of each subsequent word is capitalized. The remaining characters are lower-case.

Variable names and return value names are typically short and descriptive (for example, columnName or firstName). Variables may be listed with their data type.

Importing Scripting Classes into Java Source Files

Whether creating custom Java or using best practice Java provided by Oracle, you must have the appropriate import statements in your Java source file. When the compiled class file contains the appropriate references, the methods in the class have access to existing Java classes created to support Oracle Scripting. The methods and objects in the referenced files are imported and made available to the class loader as if the definitions were included in your Java file. Only import public Java classes; referencing private classes is not supported.

Following is a recommended minimum set of import statements to include in Java source files used for Oracle Scripting. This is typically placed in the source file after the package designation (if any) and before the designation for the class to begin.

```
import oracle.apps.ies.integration.common.*;
import oracle.apps.ies.integration.common.exception.*;
```

If creating custom Java beans to be used as replacements for panels, you can also include:

```
oracle.app.ies.integration.client
oracle.app.ies.integration.client.external
```

Add to your import statements any Java classes you need to import in addition, such as:

```
import java.text.*;
import java.util.*;
```

The above Java classes are an example only, and may not be required to support your code. Additionally, other classes may be required, as appropriate. Reference JDK documentation for information on appropriate Java classes and inheritance.

PL/SQL Commands

Using a PL/SQL command, you can execute any procedure or function stored in a database table. The PL/SQL command is executed when the appropriate object is reached in the flow of the script at runtime.

Guidelines

- You can interact with database tables using PL/SQL commands or by using a Script Author block object. Blocks can query, insert, or update database tables, in addition to providing a clearly identifiable location for inserting an API into a script. For information on defining blocks, see Using Oracle Scripting > Using the Script Author > Defining Blocks.

- PL/SQL commands in graphical scripts require connection information to be defined in the command. This tells Oracle Scripting how and where to connect to the appropriate database tables to access the stored procedure or function. For

information on connection information, see Using Oracle Scripting > Using the Script Author > Deploying the Script > Connecting to the Database.

- Commands are associated with objects using several approaches. For more information on where PL/SQL or other commands can be placed into a script, see Section 1.1.1, "Where Commands Are Defined in the Script Author".

- No provision is made in the Script Author to write, test, or deploy to the database stored PL/SQL procedures or functions. All PL/SQL code must be created by certified, trained and knowledgeable individuals using external applications or utilities.

- When defining a PL/SQL command in the Script Author, you must assign parameters (using the Parameters area of the Command Properties window) to match all parameters in the stored function or procedure, whether they are IN, OUT, or IN/OUT type. Both OUT and IN/OUT parameters will be passed back into the Scripting blackboard as key/value pairs. The blackboard key for each parameter is identical to the name used when the parameter is defined in the Script Author.

Blackboard Commands

The Scripting blackboard is a virtual memory space where information is stored in key/value pairs. This information is only retained for the duration of a single Oracle Scripting interaction.

Using a blackboard command, you can retrieve values from a specific script interaction, and display the value at runtime, or use the value for processing within the script.

Providing Values to the Blackboard

This section describes the three methods of providing values to the Scripting blackboard.

Providing an Answer at Runtime

When an answer is provided by the script end user at runtime, this information is written to the blackboard. The key to access this information is the "question" (or "answer definition") name entered into the Name field of the Answer Entry window in the Script Author when the panel answer was defined. The answer provided by the script end user at runtime is the value.

For example, if you create an answer definition in the Script Author named lastName, then at runtime, as soon as that answer is provided by the script end user (Mr. Lincoln) in the corresponding text field, the Scripting blackboard is populated with the key value pair of (lastName, "Lincoln"). This value persists until the script terminates, or the panel is refreshed and populated with a different value.

Setting a Value Using the setBlackBoard API

Using the Scripting API setBlackBoardAnswer in a Java command, information can also be written explicitly to the blackboard without interaction by the script end user. This

takes two parameters: the blackboard key (entered as a string) and the value you wish to set (a serializable value). The return type for this API is void.

Regardless of the method of populating information to the blackboard, a blackboard command for which you specify the key will return the requested value.

Launching a Script from an Integrated Application

Launching a script from either of two integrated business applications, Oracle TeleSales or Oracle Collections, results in that application invoking Scripting APIs to set a distinct set of values from the business application to the Scripting blackboard.

For more information, see the section Parameters Passed to the Scripting Blackboard in the *Oracle Scripting Implementation Guide*.

Uses of the Blackboard Command

Using an embedded value containing a blackboard command, you can display information known to the Scripting Engine for the current session to display that value in a panel.

You can also use a blackboard command as a parameter to another command. In this way, you can pass a required value stored in the blackboard for the current script session to its parent command to execute. For example, if you need to pass the name of the current customer as a parameter to a PL/SQL command, you can then retrieve specific information for that customer into the script.

Another example of using a blackboard command as a parameter to another command is to pass blackboard parameters to a Java command, and use the Java code to evaluate whether specific information has been collected. Based on whether the indicated keys are null, you can route the flow of the script (for example, to a group that collects the information and routes the script accordingly). This can provide dynamic features to the script at runtime based on the set of requirements indicated in Script Author commands.

Maintaining State in the Scripting Blackboard

The blackboard maintains state for the key/value pairs it stores, relative to the flow of the script. In this way, if there is a change in the value assigned to the key, this can be reflected based on the method or API you use to evaluate the key/value pair.

Take the example of a particular blackboard key in a given script (firstName). In this scenario, there is only one manner in which a value is provided: the script end user enters his first name (Juan) in a text field in the second panel.

If you display the key/value pair prior to the answer being provided, the result is "firstName", null. If you display the key/value pair after the answer is provided, the result is "firstName", "Juan". If later in the script the end user returns to this panel and changes the answer to "John," then the result of displaying this key/value pair will be "firstName", "John."

Guidelines

- Commands are associated with objects using several approaches. For more information on where blackboard or other commands can be placed into a script, see Section 1.1.1, "Where Commands Are Defined in the Script Author".

- Blackboard commands do not require the ServerProxyBean to be passed as a parameter. You simply need the blackboard key. (Note that this may not be true when using blackboard APIs that require the proxy bean to be in the Interaction state.)

- When defining a blackboard command, there are two values in the Command Info area of the Command Properties window: Name, and Command. The Command field is crucial, and must specify the blackboard key. The Name field is optional. Many script developers use the same value (the blackboard key) for this field. If that is not clear, you may use a different value. The Name field can also be left blank, but this is not recommended.

- If attempting to access a blackboard value that was entered by the script end user responding to a multiple-selection answer type (a checkbox group or multi-select list box), then the command you create in the Script Author must know that the response or responses are stored in a vector. Thus, this blackboard command would be nested in a parent command. For the nested blackboard command, select the option Add Value as Command? When the command executes at runtime, the Scripting Engine attempts to evaluate each parameter of a command. If the parameter is marked as an Iterating Parameter, the Scripting Engine verifies that the value of the parameter is a vector. Then it iterates through the vector, executing the command once for each element in the vector. Each element in the vector is passed to the command at the same place in the list of parameters as the iterating parameter.

How Long Is Blackboard Information Retained?

Information stored in the Scripting blackboard is only retained for the duration of a single Scripting interaction. This interaction begins with the instantiation or launch of a specific script, and ends either when an agent terminates the script manually or when the script has concluded. A script is concluded when the termination node on the root graph or top level of the canvas is reached in the flow of the script.

Using Scripting APIs

Using a variety of Scripting APIs, information can be written explicitly to the blackboard, or manipulated from the blackboard. To explicitly write values to the blackboard, you must use blackboard APIs as Java commands.

Using Scripting blackboard APIs, you can enable selection and dynamic presentation of appropriate questions based on answers previously provided by the script end user (or by evaluating blackboard information explicitly pulled into the script from database tables).

The table below lists Oracle Scripting blackboard APIs, including the method name, a description of the method, the return type, and any parameters passed.

Method Name	Description	Return Type	Parameters
getRelativeBlackBoardAnswer	Returns the answer for the specified blackboard key relative to the point in the script that the command is executed. Returns null if no value is associated with the key.	Serializable	Proxy String key
getLastBlackBoardAnswer	Returns the last answer saved to the blackboard for the specified key, regardless of the point in the script that the command is executed. Returns null if no value is associated with the key.	Serializable	Proxy String key
getAllBlackBoardAnswers	Returns a vector containing all the answers in the blackboard for the specified key. Returns null if no value is associated with the key.	Vector	Proxy String key
setBlackBoardAnswer	Sets the value for the specified blackboard key. This API is not recommended to be used to set a value identical to a key created by using an answer definition name in a script.	void	Proxy String key Serializable value
removeAllBlackBoardAnswers	Removes all answers in the blackboard for the specified key. Recommended only for removing values placed by using the setBlackBoardAnswer API in a Java method. This API is not recommended to remove values for a key created using an answer name.	void	Proxy String key Serializable value

Forms Commands

Using forms commands in graphical scripts created with the Script Author, you can accomplish the following:

- Get a value from forms-based Oracle applications to use in a script
- Set a value in forms-based Oracle applications from a script
- Launch other forms-based applications from a script
- Launch a specified URL in a Web browser from a script
- Invoke custom PL/SQL in response to a forms command
- Call business application-specific APIs customized for use with Oracle Scripting

The forms command is executed when the appropriate object is reached in the flow of the script at runtime.

Business Rules Apply

When developing a script using forms commands, you must understand and work within the business rules enforced by the forms-based application. Although the forms command automates a process that would otherwise be performed by an agent in the application, requirements for following those business rules stand. Ignoring these requirements can negatively impact either the data the script must retrieve or set, or the ability of the script to retrieve or set those values each time a script is executed.

For example, if trying to obtain a customer's last name from a forms-based customer profile, you must ensure that the customer profile is populated. As another example, if a forms-based application requires you to provide information in form A prior to accessing form B, results may be unpredictable unless your script provides the prerequisite values for form A prior to getting or setting data in form B.

In general, using forms commands in the Script Author, you can get or set values in any form directly accessible from the Navigator (a top-level form) with high confidence. Business rules still apply as if the agent were physically accessing the application. For example, the user must have the appropriate responsibility to open an item from the Navigator and may require specific roles, usages, or group memberships.

Second and subsequent level forms (nested forms accessed through the top-level form) are likely to have prerequisites or dependencies for data or agent interaction that may not be required at the top level. The further a form is nested (the more interaction required by the agent to reach the form in the application from the Navigator), the more likely it is to have such dependencies. When you automate the process in a script, you must ensure those requirements are met as surely as if the agent were clicking in the business application.

Thus, scripts getting or setting forms values should be thoroughly and rigorously tested to ensure expected results.

Guidelines

- Forms commands are used to integrate scripts with Oracle Forms-based applications. This requires appropriate installation and implementation of those

applications and a working Forms server.

- To set a value in a form, two conditions must prove true:

  1. The form must be able to be invoked by APP_NAVIGATE.EXECUTE or FND_FUNCTION.EXECUTE forms commands. All appropriate parameters for each of these specific commands must be provided.

  2. The Scripting form must be part of the same forms window or set as the form in which you want to set a value. Forms applications that are part of the Scripting Workbench object group own some of Scripting's objects, including bean_area, window, and canvas, and can more freely exchange information. At time of publication, of the three business applications integrated with Oracle Scripting, only the Customer Support module of Oracle TeleService currently meets this definition.

     Note: Based on the information above, you cannot get or set values in integrated Forms-based applications such as Oracle TeleSales or Oracle Collections, unless changes are made to those applications to be part of the Scripting Workbench object group.

- To get a value from a form (whether you wish to display the value in a script, use the value for evaluating logic to control the script flow, or write that value elsewhere) the form must be open. If the value of the form field is null, you must ensure the logic in your commands considers a null condition prior to attempting to display the value in the script, or the script will hang (pause and halt processing) and the agent will need to close the script manually, losing all information in the current Oracle Scripting session.

- When defining a forms command in a graphical script, in the Command Info area of the Command Properties window there are two fields: the Name field and the Command field.

  - While the name value is not critical to the execution of the command, you should provide a meaningful value in this field. Some developers choose to use the same precise string for both the Name and Command values. Others elect to use the name field to designate the purpose for the command. If designated, ensure the values you provide meet established naming conventions or guidelines.

  - The command value generally contains the function for the command. Thus, if using the APP_NAVIGATE.EXECUTE command, type this string in the Command field.

- Command parameters to a forms command must consist of the block name and the

form field. These are entered into the parameter window in the Value field using the syntax <block name>. <field name>. (The block is a block defined in the same form that contains the Scripting Window.) The Name field of the parameter is not critical. As a general rule, the same value can be used for both fields. However, you must ensure consistency with designated project guidelines, if any.

- Forms commands must contain a return value as a parameter if the intention of the command is to return a value. This is true regardless of whether the value is shown as an embedded value in the panel layout editor, or whether the value is saved to the Scripting blackboard.

Oracle Scripting Forms APIs

The following are Oracle Scripting APIs that can be called as Forms commands.

Command Name	Description	Parameters	Return Value
getValueFromForm	Returns the answer for the specified field in the containing form, as specified by the BLOCK.ITEM command parameter. Returns null if no value is associated with the specified form field. Requires a return value to be specified.	BLOCK.ITEM	Required
setValueInForm	Sets the contents of the Value command parameter from the Script Author to the specified field in the containing form, as specified by the BLOCK.ITEM command parameter. The Value parameter can be a constant value, or a value returned by executing another command.	BLOCK.ITEM Value	None
FND_ FUNCTION.EXECUTE	This function opens a new instance of a form. This should be used whenever you need to open a form programmatically.	Function that launches the specific form Any standard parameters associated with that function	None

Command Name	Description	Parameters	Return Value
APP_ NAVIGATE.EXECUTE	This function calls a form function to reuse an instance of the same form that has already been opened.	Function that launches the specific form Any standard parameters associated with that function	None
WEB.SHOW_DOCUMENT	This command allows a script to launch a URL as an action within the script.	Function that launches the specific form Any standard parameters associated with that function	None

Getting a Value from Forms-Based Oracle Applications

Use this procedure to get a value from forms-based Oracle applications to use in a script. The getValueFromForm application program interface (API) takes one parameter (the BLOCK.ITEM attribute of the value you wish to retrieve from the form) and one return value (the name you use in the Script Author to reference the value returned from the form).

The value returned by the command is not automatically set to the Scripting blackboard, but is available to the Scripting Engine (after the command is successfully executed) to be used in one of three ways:

1. To populate an answer control (using the default command specified in the question's data dictionary).

2. To display in a panel using an embedded value (using the name provided in the Value field of the return value parameter).

   Note: In order to use the retrieved value in any other manner or at any other location in the script, you must first set the return value to the blackboard using the setBlackBoardAnswer API (typically using a Java command), and retrieve it using the blackboard key/value pair.

3. To use the return value of the forms command as a parameter to another command.

In this case, the forms command would be nested in the top-level command, and the Add Value as Command? check box is selected, providing a second command properties window in which to define the forms command.

If retrieving the value to use later in the script with blackboard commands, getValueFromForm API can be associated anywhere a command can be specified.

Prerequisites

- The form from which the value is retrieved must be accessible by the responsibility of the current user.
- You must know the block and field name for the specific form value you want to retrieve. From an Oracle Forms window, with your cursor in the appropriate field, select Help > Diagnostics > Examine to view the block and field variable names.
- You must account for null values to ensure no errors are encountered. Attempting to retrieve null values can cause a substantial delay at script runtime until a timeout for the request is encountered.

Steps

1. If you want to display the value returned by this forms command in the current panel as an embedded value, then perform the following:
   - Navigate to the panel layout editor.
   - Enter the appropriate text you want to appear at runtime.
   - At the location in the panel text in which you want the return value from the forms command to appear, type any text as a placeholder.

     For example, type EV.
   - Highlight the placeholder text, and from the panel layout editor toolbar, click the EV button to indicate that you wish to create an embedded value.

     The Modify Command Properties button in the panel layout editor toolbar enables.
   - Click the Modify Command Properties button.

     The Command Properties window appears.
2. If you want to display the value returned by this forms command in the answer control of a new panel answer, then perform the following:
   - Navigate to the Answer Entry Dialog window of the designated panel answer.
   - In the Name field, type a name for this question (also known as an answer

definition).

For example, type contactPersonFirstName.

- From the UI Type area, define the question user interface control type.

  For example, to provide the value of the contact person's first name in a text field, select Text Field.

- Optionally, if you want the question control to contain a label in the script at runtime, in the Label for reporting field, type a value.

  For example, type Contact person first name:

- Click **Edit Data Dictionary**.

  The Edit Data Dictionary window appears. The last tab selected during this Script Author session is selected.

- Click the General tab.

- In the General area, for the Default Command field, click **Edit**.

  The Command Properties window appears.

3. From the Command Type area, select Forms Command.

   Tip: If defining several similar forms commands, you may want to define the first command in the command library. Subsequently, when accessing the Command Properties window, you can click Use Command Library, select the appropriate script and click OK, and then modify properties as required.

4. In the Command Info area, in the Name field, type a name for this command.

   Note: The value for the Name field is not critical, but the field must not be null. Some developers use the getValueFromForm API name here as well as the Command field; others enter a description of the parameter; others combine both, such as getValueCONT_PER_FIRST_NAME. Ensure you follow any designated project guidelines or naming conventions.

   For example, type getContactPersonFirstNameFromForm.

5. In the Command field, type getValueFromForm.

   Note: This value is case-sensitive.

6. In the Parameters area, click **Add**.

   The Parameters window appears.

   > **Note:** The parameters window for a command parameter appears the same as the parameters window for a return value. Ensure you enter the command parameter value in the correct field.

7. In the Parameters window for this command parameter, perform the following:

   - Leave the Class field blank.
   - In the Name field, enter a name for this command parameter.

     For example, for the contact person first name, enter CONTACT_HEADER_BLK.CONT_PER_FIRST_NAME.

     > **Note:** The value for the Name field is not critical, but the field must not be null. Some developers use the BLOCK.ITEM value here as well as in the following Value field; others enter a description of the command parameter. Ensure you follow any designated project guidelines or naming conventions.

   - In the Value field, enter the BLOCK.ITEM value for the form value you want to retrieve using this command parameter.

     For example, for the contact person first name, enter CONTACT_HEADER_BLK.CONT_PER_FIRST_NAME.

   - Click **OK**.

     The Parameters window for the command parameter closes.

     The Command Properties window displays the Name value of the command parameter in the Parameters area.

8. In the Return Value area, perform the following:

   - Click **Edit**.

     The Parameters window for this return value appears.

     > **Note:** The parameters window for a return value appears the same as the parameters window for a standard Script Author command parameter. Ensure you enter the return value parameter in the correct field.

- In the Parameters window, in the Name field, enter any meaningful value.

  For example, enter the BLOCK.ITEM value, or a simplified version such as ContPerFirstName or FirstName.

- Leave the Value field empty.

- Click **OK**.

  The Parameters window for the return value parameter closes.

  The Command Properties window displays the Name value of the parameter in the Return Value area.

9. Click **OK** to save the properties you defined and to close the Command Properties window.

10. Save your work.

11. When executed from the Customer Care component of Oracle TeleService, this script will retrieve the populated value of the selected field name to the script do display in the panel text or answer control, as appropriate.

Guidelines

- Values returned from a form can be displayed in a panel using an embedded value using the procedure described above.

- Values returned from a form can be displayed as the default command of an answer control using the procedure described above.

- Values returned from a form can be used as parameters in another command.

- Before values returned from a form can be used as parameters in any other Script Author command, you must save them to the blackboard using the setBlackBoardAnswer API as a Java command.

- This will not function with Oracle TeleSales or Oracle Collections.

Setting a Value in Forms-Based Applications

Use this procedure to set a value in forms-based Oracle applications from within a script. The setValueInForm API takes two command parameters (the BLOCK.ITEM attribute of the value you wish to set in the form, and the Value parameter which specifies the value to set). No return values are required.

Prerequisites

- In order to set a value in a form from a script, the form must be open and the field must be modifiable.

- The script user must have the appropriate responsibility to access the specified field.

Steps

1. From an appropriate location in a script, access the Command Properties window.
2. From the Command Type list, select **Forms Command**.

   Tip: If defining several similar forms commands, you may want to define the first command in the command library. Subsequently, when accessing the Command Properties window, you can click Use Command Library, select the appropriate script and click OK, and then modify properties as required.

3. In the Command Info area, in the Name field, type a name for this command.

   Note: The value for the Name field is not critical, but the field must not be null. Some developers use the setValueInForm API name here as well as the Command field; others enter a description of the parameter; others combine both, such as setValueCONT_PER_FIRST_NAME. Ensure you follow any designated project guidelines or naming conventions.

   For example, type setContactPersonFirstNameInForm.

4. In the Command field, type **setValueInForm**.

   Note: This value is case-sensitive.

5. In the Parameters area, click **Add**.

   The Parameters window appears.

   Note: The parameters window for a command parameter appears the same as the parameters window for a return value. Ensure you enter the command parameter values in the correct field.

6. In the Parameters window for this command parameter, perform the following:
   - Leave the Class field blank.
   - In the Name field, enter a name for this command parameter.

     For example, for the contact person first name, enter CONTACT_HEADER_

BLK.CONT_PER_FIRST_NAME.

> **Note:** The value for the Name field is not critical, but the field must not be null. Some developers use the BLOCK.ITEM value here as well as in the following Value field; others enter a description of the command parameter. Ensure you follow any designated project guidelines or naming conventions.

- In the Value field, enter the BLOCK.ITEM value for the form value you want to set using this command parameter.

  For example, for the contact person first name, enter CONTACT_HEADER_BLK.CONT_PER_FIRST_NAME.

- Click **OK**.

  The Parameters window for the command parameter closes.

  The Command Properties window displays the Name value of the command parameter in the Parameters area.

7. In the Parameters area, click **Add** to begin adding the second command parameter.

   The Parameters window appears.

8. In the Parameters window for this command parameter, perform the following:

   - Leave the Class field blank.
   - In the Name field, type Value.
   - In the Value field, type the value you want to pass to the form.

     For example, to enter a contact person first name of Norman, type Norman.

   - Click **OK**.

     The Parameters window for the command parameter closes.

     The Command Properties window displays the Name value of the command parameter in the Parameters area. Two parameters are listed.

9. Click **OK** to save the properties you defined and to close the Command Properties window.

10. Save your work.

11. When executed from the Customer Care component of Oracle TeleService, this script will set the specified value into the designated field as identified by the

BLOCK.ITEM parameters.

Guidelines

- Values can only be passed to a form in a modifiable field.
- The form must be open to successfully set the value.
- This will not function with Oracle TeleSales or Oracle Collections.

Defining a Shortcut Button to Launch a Form

You can launch a form from Oracle forms-based applications from within a script. Use this procedure to define a Shortcut button in the button bar to open a specific form. When the script is executed in the Scripting Engine agent interface at runtime, you can click a button in the shortcut button bar to open the specified form in a separate window.

Prerequisites

- The function you wish to associate with the shortcut button must be accessible to the Oracle Applications responsibility of the user that will execute the script.

  Note: It is not sufficient for the Oracle Applications user to be assigned the responsibility required to execute the function. The user executing the script must do so from the required responsibility for the form to open successfully.

- More than one window may be associated with the same form.

Steps

1. Using the Script Author, open a new or existing graphical script.
2. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.
3. In the script properties menu, select the **Shortcut Panel** attribute.

   Shortcut Panel properties appear.
4. Click **Add**.

   The Shortcut Info Entry window appears.
5. Enter Shortcut information as appropriate.
   - In the ID field, enter a unique identification name for this shortcut.

     For example, if launching the Find/Enter customers form, type LF/E.fmx.

- In the Label field, enter label that you want to appear on the shortcut button.

  For example, type Launch Find/Enter Customers Form.

- In the Tooltip field, enter the contents of the tooltip, which will appear in the Scripting Engine agent interface when the agent's cursor rests on the button.

  For example, type Find or enter customers in Customer Support.

- On the Command line, click **Edit**.

  The Command Properties window appears.

6. Enter the appropriate information to reference the command you want to execute at runtime when the Shortcut button is clicked.

   - From the Command Type list, select **Forms Command.**

   - In the Command Info area, in the Name field, type a name for this command.

     Note: The value for this field is not critical. Some developers use the same name as the Parameter Value field; others enter a description of the parameter. Ensure you follow any designated project guidelines or naming conventions.

     For example, type the form name, AR_ARXCUDCI_STD.

   - In the Command field, type the name for the function that launches the specified form.

     For example, type AR_ARXCUDCI_STD.

   - If the specified function required any parameters, then in the Parameters area, click Add, and enter the parameter. Click **OK** to save the parameter and close the window. Repeat this step as required.

     The Command Properties window displays the name value of the parameter.

   - Click **OK** to save the Command Properties and close the window.

     The Shortcut Info Entry Dialog window displays the name value of the command.

7. In the Shortcut Info Entry Dialog window, to enable the Shortcut button upon the launch of the script, select **Enabled**.

   A check appears, indicating the button is enabled.

8. If you want to save your work and continue, click **Apply**.

9. To add additional Shortcut buttons, click **Add** in the Shortcut Panel properties and repeat the required steps.

10. Click **OK** to save your work and exit the Shortcut Info Entry window.

Guidelines

- The responsibility of the logged-in user governs the functions accessible in Forms-based applications.
- When you click a shortcut button in the agent interface to open a form, any forms that are currently minimized (such as the Script Chooser) may also appear.
- For command values, use APP_NAVIGATE.EXECUTE only when you want to reuse an instance of a form that has already been opened. In all other instances, use FND_FUNCTION.EXECUTE as the command value.

Launching a Web Browser as a Script Action

Use this procedure to define a forms command in a graphical script to launch a specific URL in a Web browser as an action to a command.

Web browsers can also be opened to a specific URL using the setBrowserURL Oracle Scripting API. That API can be used to establish a hypertext link in panel text that the runtime user can select if desired. Using this API as a forms command to a specified action ensures that the browser will launch when the action is triggered in the script, removing the element of choice and obviating the requirement to display a hypertext link.

Prerequisites

The designated URL must be accessible by users of the script. For example, if security such as a firewall is in place, the user must be on the appropriate side of the firewall to access the site through a Web browser in order for the same action to occur successfully using this API from a script.

Steps

1. From an appropriate location in a script, access the Command Properties window.

2. From the Command Type list, select **Forms Command**.

   Tip: If defining several similar forms commands, you may want to define the first command in the command library. Subsequently, when accessing the Command Properties window, you can click Use Command Library, select the appropriate script and click OK, and then modify properties as required.

3. In the Command Info area, in the Name field, type a name for this command.

Note: The value for the Name field is not critical, but the field must not be null. Ensure you follow any designated project guidelines or naming conventions.

For example, type setURL.

4. In the Command field, type WEB.SHOW_DOCUMENT.

   Note: Case is not important for this value.

5. In the Parameters area, click **Add**.

   The Parameters window appears.

6. In the Parameters window for this command parameter, perform the following:

   - Leave the Class field blank.
   - In the Name field, enter a name for this command parameter.

     For example, for a command that launches Oracle's web site, type www.oracle.com.

     Note: The value for the Name field is not critical, but the field must not be null. Some developers describe the function of this parameter; others include the actual URL in this field. This value appears in the command properties window after defining the parameter. Ensure you follow any designated project guidelines or naming conventions.

   - In the Value field, enter the URL you want the web browser to open, including the protocol. For example, for a command that launches Oracle's web site, type www.oracle.com.
   - Click **OK**.

     The Parameters window for the command parameter closes.

     The Command Properties window displays the Name value of the command parameter in the Parameters area.

7. Click **OK** to save the properties you defined and to close the Command Properties window.

8. Save your work.

9. When this action is reached in the flow of the script, a new instance of the Web browser used to launch the user's Oracle Applications session will appear in a separate window, displaying the contents of the designated page.

Calling APIs Customized for Use with Oracle Scripting

Due to its Java architecture and its ability to communicate with the applications database, Oracle Scripting is a highly extensible application. Using application program interfaces customized for this purpose, applications can easily communicate with Oracle Scripting.

Oracle TeleService

Oracle TeleService is a forms-based business application that includes Oracle Customer Care and Oracle Customer Support. This application includes the ability to log and track service requests.

The Oracle Customer Care component of Oracle TeleService passes information regarding a selected customer record to the Scripting blackboard when you execute a Forms command in the script. A record must be selected in the Customer Care contact center in order for the information to successfully transmit to the Scripting blackboard. The values include information about the customer (customer ID, customer account ID, contact and contact point ID and type), and the interaction ID.

Use this procedure to define a forms command to obtain a value passed to the Oracle Scripting blackboard from the Customer Care component of Oracle TeleService. If the value retrieved is to be used as a parameter for another command, this procedure will be performed as a nested command.

Prerequisites

- You must know the TeleService/Customer Care API name.
- You must know the correct location in the script in which to define this command.

Steps

1. If you want to display the value returned by this forms command in the current panel as an embedded value, then perform the following:
   - Navigate to the panel layout editor.
   - Enter the appropriate text you want to appear at runtime.
   - At the location in the panel text in which you want the return value from the forms command to appear, type any text as a placeholder.

     For example, type EV.
   - Highlight the placeholder text, and from the panel layout editor toolbar, click

the EV button to indicate that you wish to create an embedded value.

The Modify Command Properties button in the panel layout editor toolbar enables.

- Click the Modify Command Properties button.

  The Command Properties window appears.

2. If you want to display the value returned by this forms command in the answer control of a new panel answer, then perform the following:
   - Navigate to the Answer Entry Dialog window of the designated panel answer.
   - In the Name field, type a name for this question (also known as an answer definition).

     For example, type contactPersonFirstName.
   - From the UI Type area, define the question user interface control type.

     For example, to provide the value of the contact person's first name in a text field, select Text Field.
   - Optionally, if you want the question control to contain a label in the script at runtime, in the Label for reporting field, type a value.

     For example, type Contact person first name:
   - Click **Edit Data Dictionary**.

     The Edit Data Dictionary window appears. The last tab selected during this Script Author session is selected.
   - Click the General tab.
   - In the General area, for the Default Command field, click **Edit**.

     The Command Properties window appears.
3. If you want to use the value returned by this forms command as a parameter to another Script Author command, then perform the following:
   - Navigate to the Command Properties window of the top-level command.
   - In the Parameters area, define the top-level parameter required for the top-level command.

     For example, if the top-level command is a PL/SQL command to write a record to the database, and you want to obtain the contact ID for the selected customer, then in the Name field of the Parameters window, provide an

appropriate name (for example, contact_id).

- In the top-level command parameter, in the Parameters window, select the Add Value as Command? check box.

  The Parameters window refreshes. The Value field now includes an Edit and a Remove button.

- In the Value field, click **Edit**.

  The Command Properties window for the nested command appears. It is in this window that you will define your forms command.

4. From the Command Type area, select Forms Command.

   Tip: If defining several similar forms commands, you may want to define the first command in the command library. Subsequently, when accessing the Command Properties window, you can click Use Command Library, select the appropriate script and click **OK**, and then modify properties as required.

5. In the Command Info area, in the Name field, type a name for this command.

   Note: The value for the Name field is not critical, but the field must not be null. Some developers use the identical API name here as in the Command field; others enter a description of the parameter. Ensure you follow any designated project guidelines or naming conventions.

   For example, type GetContactId.

6. In the Command field, type the exact name of the API. The Oracle TeleService APIs are documented in the Integrating Oracle Scripting section of the *Oracle Scripting Implementation Guide*.

   Note: This value is case-sensitive.

   For example, to get the contact ID for the selected customer, type GetContactId.

7. No command parameters are required.

8. In the Return Value area, perform the following:

   - Click **Edit**.

     The Parameters window for this return value appears.

- In the Parameters window, in the Name field, enter the name you want to use as the blackboard value.

  For example, type GetContactId.

- Leave the Value field empty.

- Click **OK**.

  The Parameters window for the return value parameter closes.

  The Command Properties window displays the Name value of the parameter in the Return Value area.

9. Click **OK** to save the properties you defined and to close the Command Properties window.

10. Save your work.

11. When executed from the Customer Care component of Oracle TeleService, this script will retrieve the appropriate value and set it to the Oracle Scripting blackboard using the return value name.

Guidelines

- Subsequent to execution of the forms command, you can reference the value in the blackboard using the return value name as the blackboard key to the key/value pair.

- This can be executed as a parameter to another command in order to provide the value obtained by the forms command as a command parameter to that top-level command.

Oracle TeleSales and Oracle Collections

While Oracle TeleSales and Oracle Collections do not allow Oracle Scripting to get or set values in forms, both business applications do provide pre-built integration. When a script is launched from either of these business applications (in their separate respective methods), over fifteen possible values are sent to the Oracle Scripting blackboard. These values can be used in the script to perform various functions, from branching to display of information and so on, using Oracle Scripting blackboard APIs.

The full list of values that are potentially sent to the blackboard are documented in the Integrating Oracle Scripting section of the Oracle Scripting Implementation Guide. Note that the values must be available to the business application in order to be sent to the blackboard; any values not available will be placed in the blackboard as a null value. Thus, ensure your scripts take possible null values into account when using these values. For example, you can test the value passed by the blackboard key CONTACT_ID and, if null, present the user with the ability to retrieve the contact ID from the applications database.

Related Topics

Oracle TeleService APIs are also documented in the Integrating Oracle Scripting section of the *Oracle Scripting Implementation Guide*.

Oracle TeleSales and Oracle Collections values sent to the Oracle Scripting blackboard are documented in the Integrating Oracle Scripting section of the *Oracle Scripting Implementation Guide*.

For more information on APIs developed by business applications to work with Oracle Scripting, see the appropriate product documentation.

Constant Commands

Using a constant command provides a constant, predetermined value at runtime. This is often used to provide a default selection for an answer with one or more possible values. You can also use a constant command to provide a default return value as a parameter to another command.

Guidelines

- Commands are associated with objects using several approaches. For more information on where constant or other commands can be placed into a script, see Where Commands Are Defined in the Script Author, page 1-1.

- Use a constant command to provide a default answer selection or lookup value to an open-ended question at runtime to avoid a null value without using a validation routine. For example, when provided with a text field or text area, you can provide a constant value of "not applicable" that the user can overwrite if applicable.

- When used to provide a default answer selection, the answer can be changed at runtime. This is true regardless of whether the answer user interface type is open-ended (text field, text area, password without validation, single or group check boxes, and the multi-select list box can be null) or whether it enforces a value (radio buttons, drop-down list or multi-select list boxes, or a button if more than one lookup is provided for button types.

- Since the purpose of the constant command is to return a constant value, remember to provide the desired constant string as a return value in the Command Properties window.

- Constant commands do not require the ServerProxyBean to be passed as a parameter.

- When defining a constant command, there are two values in the Command Info area of the Command Properties window: Name, and Command. Neither of these fields are crucial, provided you provide the constant value as a return value to this command. Many script developers use the same value (the actual constant value being provided) for both fields. Others use the answer definition name, or if the

answer is provided as a constant value for a blackboard key, the blackboard key name. The Name and Command fields can also be left blank, but this is not recommended.

Delete Actions

The delete action causes a row to be deleted from the database, based on what is currently selected in the Scripting cursor.

When you execute a database query from a script using a query block, and one or more rows that match your criteria are returned, the cursor holds the first row. All rows are retained in static memory until the next query is executed (or until the interaction ends). Using the delete action will cause that row to be deleted from the database.

The delete action is not strictly a command, but is provided in the same interface to make this functionality accessible to script developers in a manner consistent with the application's existing paradigms.

- Scripting APIs are available to advance the cursor, check if the cursor is valid, and so forth. These can be used in combination with the delete action during the development of a script to determine which row of information should be deleted.

- The delete action will delete the row of information in the database last retrieved by a query, regardless of when in the current script interaction the last query took place. Thus, before including a delete action in a script, it is recommended to ensure that the relevant query is successful (that it will always return rows). Ensuring the validity of a query prior to executing the delete action will preclude deletion of a row returned for the previous query if the following query returns no values in a particular set of circumstances.

- For ease of maintenance, it is recommended that you insert the delete action as close as possible in the flow of a script to its relevant query.

- The delete action takes a single parameter in the Name field. The purpose of this field is to identify the delete action within the Script Author. Regardless of the name, the database row corresponding to the row held in the cursor will be deleted. The Name field can also be left blank, but this is not recommended.

2

Customizing Oracle Scripting

This chapter covers the following topics:

- Customization and Support
- Determining Where to Define a Command
- Best Practice Java Methods
- Passing Parameters to the Web Interface
- Performing Advanced Graphical Script Tasks

Customization and Support

Each script created with Oracle Scripting is customized according to specific requirements to achieve a particular set of goals. Oracle Scripting provides a set of best practice Java methods, best practice survey scripts, and building blocks (pre-built graphical script fragments) to assist you in customizing a script to extend its functionality using existing components.

If Oracle Scripting does not meet documented functionality, you can seek the assistance of Oracle Support Services (OSS). However, Oracle does not support customization or custom code. Use of the Script Author component of Oracle Scripting requires the appropriate training, skill set, technology, and experience.

Formal and Informal Training

For interaction center agents using the Scripting Engine agent interface, Oracle recommends that you become familiar with any new script prior to using the script in production. Based on the complexity of the script, this process can take between five minutes and a few hours.

For script developers, implementation staff, scripting administration, and survey campaign administration, Oracle suggests that you attend one or more of the Oracle Scripting training classes available through Oracle University.

Training and certification in the requisite technologies is also a prerequisite.

Skill Sets and Experience Required for Oracle Scripting

Skill sets required for Oracle Scripting vary on components to be used at the enterprise and the degree of complexity required. Scripting may require script developers, script administrators, survey campaign administrators, integration with other Oracle applications, custom code development, JSP developers and HTML developers.

Oracle strongly recommends contracting Oracle Consulting Services (OCS) or certified, experienced consulting partners to develop the first several scripts to be used in production in an enterprise.

Script and Survey Development and Maintenance

The Script Author is intended for the functional user with some technical knowledge. Additionally, certification in and knowledge of the related technologies relevant to your scripting project is required. Each Oracle Scripting script is customized to meet specified business needs. These needs, defined as a set of business rules, are incorporated into a script by trained, knowledgeable script developers using the Script Author tool. Each script includes the script metadata, instructions regarding the one or more possible script flows to completion, and references to any required custom code.

To make the Script Author more accessible to non-technical users, Oracle has added a Script Wizard feature to the Script Author Java applet. Users can quickly and easily create simple scripts or surveys by providing script information in a series of windows known as a wizard. This feature can help reduce the time it takes to train a non-technical Script Author user to create and modify simple scripts. While hiding complexity from non-technical users, the Script Wizard feature makes it possible to re-use questions and answers, reduce keystrokes and data entry errors, and reduce some repetitious work needed to create and modify a script.

Nonetheless, wizard scripts require the script developer to have a clear and consistent understanding of the business rules of the script and the business process flow required for the overall interaction. A knowledge of Oracle Scripting's handling of data is absolutely essential to developing successful scripts.

Custom Code

Scripts typically require custom code to extend their functionality. Regardless of the execution method of the script, custom code required may include any one or a combination of supporting technologies. These include Java, PL/SQL, Oracle Forms, Constant commands, and Scripting Blackboard commands. Custom code can be associated only with graphical scripts.

JSP

Scripts to be executed as surveys also require custom JSP development in order to customize at least one set of survey resources. These are reusable, and an enterprise

may elect to use a single set of resources for all surveys implemented in the enterprise. Alternatively, requirements for each survey campaign may dictate a separate set of JSP resources for each. A set of resources includes three JSP/HTML page fragments or files: a header, a final page, and an error page. These are all simple JSP pages, and seeded examples are shipped in Oracle Applications. Thus, the requirement for JSP is low and may be a one-time setup requirement.

HTML

Knowledge of HTML may be required in order to customize the presentation of a panel in a script. When you build a script using the graphical tools of the Script Author, you specify display text (including questions), in the panel layout editor. You have control over font size, typeface, color and weight, using the graphic UI provided by the panel layout editor. You also select a user interface control for each answer, such as a radio button or check box. From these choices you select, HTML is automatically generated to control the display of the specified panel attributes. This HTML includes some dynamic content which is processed and displayed at runtime.

The Script Author allows import and export of panel content layout to fully control its appearance at runtime. If you want to customize the presentation of a panel in a script, you can export and modify panel HTML. You can also generate original HTML following Script Author HTML syntax rules. For example, you may wish to use tables to align a series of radio buttons or text fields, rather than accept the default HTML layout generated by the Script Author. You can then import the customized HTML using the Script Author for a customized look for panel display. This model is appropriate for executing scripts either in the agent interface or as a survey in the Web interface.

HTML can also be modified for wizard scripts, by first graphing the script (creating a copy of the wizard script accessible by the Script Author graphical tools). Note, however, that once a script is graphed, any changes made to it cannot be accessed from the Script Wizard.

Script Campaign Management

Interaction centers using Oracle Scripting in the agent interface may have specific script campaign managers. These individuals must understand the goals and objectives of the campaign and the business rules which must be incorporated into the script to achieve those goals. Typically, these individuals will work closely with script developers to determine detailed script requirements.

Scripting Administrators

Oracle Scripting includes a Scripting Administration console. Users of this JSP/HTML-based user interface include script developers, who launch the Script Author applet from the Home tab, and administer deployed scripts and custom Java archive files from the Administration tab. Interaction center campaign administrators or system administrators (as well as script developers) will also typically access this console to run

panel footprint reports to help tune a script's structure, increase agent performance and reduce average talk time. To access the Scripting Administration console user interface from Oracle HTML-based applications, these users must have the Scripting Administrator responsibility.

Since script developers can deploy, delete, and otherwise affect scripts in production, and can manipulate information in the applications or other enterprise database, Oracle strongly recommends that only trusted users be provided with the Scripting Administrator responsibility.

Survey Campaign Management

Enterprises using Oracle Scripting to conduct survey campaigns must have survey campaign administrators. These individuals must understand the goals and objectives of the survey campaign in order to achieve those goals. These survey campaign administrators must:

- Identify the sample (target population of individuals whom they want to poll).
- Create or work with individuals to create a list of possible survey respondents consisting of the sample.
- Work closely with script developers to determine detailed script requirements and enforce the business rules which must be incorporated into the script to achieve those goals.
- Work with JSP developers to create appropriate survey resources.
- Determine how many cycles are required to conduct each survey and schedule calendar days for execution.
- Use the Survey Administration console to input business requirements for the overall survey campaign, specific cycles and deployments, define survey resources, and monitor the progress of an ongoing campaign.

Oracle Scripting includes a Survey Administration console. Users of the JSP/HTML-based survey campaign administrative user interface are non-technical users with access to detailed project requirements. These individuals are typically interaction center survey campaign administrators or system administrators.

To access the Survey Campaign administrative interface from Oracle Personal Homepage (PHP) login, these users must have the Survey Administrator responsibility.

Using Best Practices and Building Blocks

Best practice scripts and Java code and building block components are provided by Oracle on an "as is" basis. Pre-built scripts flows are provided to save you time and serve as examples of how to customize scripts. Building blocks typically use Application Program Interfaces (APIs) to read or write data to other Oracle

Applications or access the Oracle database, and can be incorporated into your custom scripts to provide them with this functionality.

When used properly, each pre-built component is tested and fully functional. However, you must fully understand how Oracle Scripting works in order to use, modify and maintain each properly.

For example, some customization components (such as best practice surveys) may be used as stand-alone scripts. Nonetheless, surveys cannot be executed until the Survey component is fully implemented, and a survey campaign is created and deployed successfully. Other customization components (such as the building blocks) are to be used as components to incorporate into your custom scripts. However, a building block may access an application that is not configured or not in use in the enterprise.

The functionality of best practice and building block components is therefore only supported to the degree that they will function in a certified test environment. Customizations are not supported. Use of these components is at your own risk.

APIs Are Public But Not Supported

APIs used with Oracle Scripting are public, but not supported. Use of any components that access APIs is at your own risk. Oracle development teams endeavor to continue supporting the APIs, best practice scripts and code, and building blocks through subsequent releases. However, Oracle cannot guarantee that APIs which worked in a previous release will continue to function in future releases in all cases.

You are therefore responsible for the functionality of any customized scripts, including use and customization of any best practice or building block components.

Determining Where to Define a Command

Commands are used in Oracle Scripting graphical scripts to execute actions at runtime. The Command Properties window is context-sensitive. When you select a command type, the window expands, or fields are disabled, as appropriate.

Use this procedure to determine where in a graphical script to begin defining a command to accomplish specific purposes.

Prerequisites

None

Steps

1. Using the Script Author, open a new or existing graphical script.
2. If you want to associate the command as an action to a branch:
    1. In the branch properties menu, click the **Actions** attribute.

2. Click **Add**.

3. If you want to associate a command as a pre-action or post-action to an object:
   1. Navigate to the object for which you wish to associate a command.
   2. Double-click on the object to access its properties.
      - To access properties of the global script object, double-click on an empty area of the canvas, or select **Script Properties** from the File menu.
      - You can also right-click any object or branch and select **Edit Blob Properties** or **Edit Branch Properties** from the resulting pop-up menu.
   3. Expand the **Actions** tree in the properties menu to display action attributes.
   4. Select **PreAction** or **PostAction** attribute from the Actions tree.
   5. Click **Add**.

      The Command Properties window appears.

4. If you want to associate the command as an API action for a block object:
   1. Double-click on the block object to access its properties.
   2. In the block properties menu, click the **Types** attribute.
   3. Select **API Block** from the list.
   4. Optionally, click **Apply**.
   5. In the block properties menu, click the **Object Dictionary** attribute.

      The Object Dictionary appears.
   6. Click **Add**.

      The Command Properties window appears.

5. If you want to associate the command to a text or timer object in the script information area:
   1. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.
   2. In the script properties menu, select the **Static Panel** attribute.

      Static Panel properties appear. The Static Panel is also referred to as the script information area.

3. In the Static Panel window, define the text or timer by selecting a position, clicking a static information area object type (**Text** or**Timer**), and adding appropriate ID and Label information.

4. On the Command line, click **Edit**.

   The Command Properties window appears.

6. If you want to associate the command to a Shortcut button in the Shortcut Panel:

   1. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.

   2. In the script properties menu, select the **Shortcut Panel** attribute.

      Shortcut Panel properties appear.

   3. Click **Add**.

      The Shortcut Info Entry window appears.

   4. Enter Shortcut information as appropriate.

      In the **ID** field, enter a unique identification name for this shortcut.

      In the **Label** field, enter label that you want to appear on the shortcut button.

      In the **Tooltip** field, enter the contents of the tooltip, which will appear in the Scripting Engine agent interface when the agent's cursor rests on the button.

      On the Command line, click **Edit**. The Command Properties window object appears.

7. If you want to associate a command as a parameter to a parent command:

   1. Define a command.

   1. Under Parameters, click **Add**.

      The Parameters window appears.

   2. In the Name field, type the name of the command parameter.

   3. In the Value field, type the value for the parameter.

      If this parameter is required by a Java method, this is identical to the value referenced in the method signature (case-sensitive).

      - If a string, type the literal string value.

      - If the value is a boolean, type true or false.

4. Select A**dd Value as Command**.

   The Parameters window will expand.

5. From the Value field, click **Edit**.

   A child **Command Properties** window appears. The command you define in this window will be executed prior to the parent command.

6. Define the command type, and type the name and command values. Define any parameters or return values as appropriate.

   Click **OK** to save these parameters and close the window. You will be returned to the parameters window for the parent command.

8. If you want to associate a command to a parent command in order to provide the parent command with a return value:

   1. Define a command.

   2. Under Parameter, click **Add**.

      The Parameters window appears.

   3. In the Name field, type the name of the command parameter.

   4. In the Value field, type the value for the parameter.

      If this parameter is required by a Java method, this is identical to the value referenced in the method signature (case-sensitive).

      - If a string, type the literal string value.
      - If the value is a boolean, type **true** or **false**.

   5. Select **Add Value as Command**.

      The Parameters window will expand.

   6. From the Value field, click **Edit**.

      A child Command Properties window appears. The command you define in this window will be executed prior to the parent command.

   7. Define the command type, and type the name and command values. Define any parameters as appropriate.

   8. In the Return Value area, click **Edit**.

      The Parameters window appears. These are return value parameters.

9. Enter return value parameters as required.

   In the Name field, enter any name. This will identify the constant command.

   In the Value field, enter the value you wish to be passed as the return value.

   Click OK to save these parameters and close the window. You will be returned to the parameters window for the parent command.

9. In the Command Properties window, enter the appropriate properties for the desired command type. Refer to instructions for a specific command type for detailed information.

10. Save your work.

Best Practice Java Methods

Oracle Applications includes a library of best practice Java methods written specifically for Oracle Scripting. Best practice Java methods provide the ability to incorporate frequently requested functionality into Script Author graphical scripts without writing new Java code.

Using best practice Java methods, you can:

- Set, enable, or disable shortcut buttons in the Shortcut button bar (agent interface)
- Start or stop a timer in the script information area (agent interface)
- Validate a user's input or response using established validation routines (Web and agent interfaces)
- Check for a null value to determine whether to perform a query or request input from user (Web and agent interfaces)
- Obtain the time of day to dynamically generate greetings or route script flow to specified paths (Web and agent interfaces)

These are only a few examples. For details, refer to the description of each method below, grouped by class.

The answer validation commands automatically associated with a question within a panel by selecting options in the Define Question Detail wizard window also access best practice Java methods (from the NodeValidation class).

Overview of Best Practice Java Methods

These Java methods are organized into three separate Java classes, grouped by function. These include:

- ScriptUtil Java Methods, page 2-11
- NodeDefault Java Methods, page 2-16
- NodeValidation Java Methods, page 2-17

The best practice Java methods are included as compiled Java class files in APPS.ZIP, a compressed archive containing Java classes for Oracle Applications. For enterprises certified to execute Oracle Scripting in the caching architecture, these classes are also included in IESSERVR.JAR.

To use best practice Java methods, simply create a Script Author command in a script, referencing the best practice Java method and any required parameters or return values.

Oracle does not make source code available. If you wish to understand the parameters used in these Java methods, refer to the descriptive tables for each of the three best practice Java classes below.

As discussed in Standard Java Naming and Usage Conventions, page 1-9, some of the best practice methods listed herein do not adhere to standard Java naming and usage conventions. These will be updated in the near future. When they are, the older best practice Java methods will be decremented. To ensure compatibility, replaced methods will continue to be included in shipped code for a period of time.

Apache Mid-Tier Architecture

Oracle Scripting implementations using the Apache mid-tier architecture can use best practice Java methods more easily than custom Java. In this architecture, the JVM on the Apache Web server references the JSERV.PROPERTIES file on the Web server to locate and identify all Java classes required. Since APPS.ZIP is referenced by this file, all best practice Java commands can be accessed without the requirement to create, compile, or upload custom Java. This speeds development and maintenance, provides for code reuse and eliminates potential Java coding errors for commonly required Scripting Engine functionality.

Caching Architecture

Enterprises upgrading previous Oracle Scripting implementations using the caching architecture may also use best practice Java methods to support Script Author scripts. The Oracle Scripting best practice Java methods are included in IESSERVR.JAR, one of the three base Java archive files that comprise the Oracle Scripting product. The inclusion of the three best practice classes in IESSERVR.JAR ensures accessibility for enterprises using the caching architecture of Oracle Scripting.

Deciding Between Best Practice and Custom Java

Typically, when writing custom Java methods to support a script, you must compile your custom Java source code, combine one or more compiled classes into a JAR file, and store the JAR file on the server.

Users of the Scripting Administration console and the Script Author Java applet can

deploy custom JAR files to the applications database. This greatly simplifies the use of custom Java code.

If using the older, stand-alone Script Author Java application instead of the integrated applet, then custom JAR files cannot be stored in the database. Based on the architecture in which you are executing scripting, you must then reference the class path of your custom Java in a configuration file (in the JSERV.PROPERTIES file on the Apache Web server for Apache Mid-Tier Architecture operations, or in the APPSWEB.CFG file on the applications server if using the Caching Architecture). Finally, using the Script Author, you must reference each specific Java method in the command field of a Java command.

In contrast, the best practice Java methods are included in APPS.ZIP. The class path for APPS.ZIP is already referenced in the appropriate configuration file. If using best practice Java methods, you do not need to generate, test, or compile Java code, nor do you need to deploy best practice Java code to the apps server or reference the class path in a configuration file. You need only reference each specific Java method in the command field of a Java command using the Script Author.

> **Caution:** *Do not, under any circumstances, add custom code to* APPS.ZIP. Customization of APPS.ZIP is not supported, and could result in loss of function not only for Scripting but for other Oracle Applications.

Use Best Practice Java Methods with or Instead of Custom Code

If existing best practice Java methods meet all your needs for extending a script, you can use them exclusively. Doing so will eliminate the need to deploy custom code to the database and map the JAR files to a script. Restricting custom Java to the use of best practice Java methods for users of the older stand-alone Script Author application will eliminate the need to write, test, compile and deploy custom Java and to modify JSERV.PROPERTIES or APPSWEB.CFG configuration files to identify the class path of the JAR files on the applications server. You can also use custom code in combination with best practice Java methods in a single script. As long as each command is appropriately referenced, any combination is supported.

ScriptUtil Java Methods

The table below describes available Java methods and their parameters.

Method Name	Description	Return Type	Parameters
jumpToShortcut	Jumps user to group containing the shortcut property shortcutName. Prerequisite: This method requires a group object on the Script Author canvas containing a shortcut value. This group is the destination of the jump. Replaces method JumpToShortcut. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter ShortcutName changed to shortcutName to follow standard parameter naming conventions.	void	Proxy String shortcutName
JumpToShortcut	*Decremented*. Jumps user to group containing the shortcut property ShortcutName. Prerequisite: This method requires a group object on the Script Author canvas containing a shortcut value. This group is the destination of the jump.	void	Proxy String shortcutName

Method Name	Description	Return Type	Parameters
startTimer	Starts an interaction timer (displays in Agent interface only). Use this method to start an interaction timer in the script information area at runtime. If timing the full interaction, this method must be called prior to the first panel. **Note:** Cannot be used as pre-action to the global script. Prerequisite: Requires a timer to be defined as a data element in the script information area (static panel). Replaces method StartTimer. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter TimerName changed to timerName to follow standard parameter naming conventions.	void	Proxy String shortcutName
StartTimer	*Decremented*. Starts an interaction timer (displays in Agent interface only). Use this method to start an interaction timer in the script information area at runtime. If timing the full interaction, this method must be called prior to the first panel. **Note:** Cannot be used as pre-action to the global script. Prerequisite: Requires a timer to be defined as a data element in the script information area (static panel).	void	Proxy String shortcutName

Method Name	Description	Return Type	Parameters
stopTimer	Stops count on interaction timer (displays in Agent interface only). Use this method to stop the count on an active interaction timer in the script information area at runtime. This will have no negative effect if invoked when a timer is not active. Prerequisite: Requires a timer to be defined as a data element in the script information area (static panel). Replaces method StopTimer. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter TimerName changed to timerName to follow standard parameter naming conventions.	void	Proxy String shortcutName
StopTimer	*Decremented*. Stops count on interaction timer (displays in Agent interface only). Use this method to stop the count on an active interaction timer in the script information area at runtime. This will have no negative effect if invoked when a timer is not active. Prerequisite: Requires a timer to be defined as a data element in the script information area (static panel).	void	Proxy String TimerName

Method Name	Description	Return Type	Parameters
resetTimer	Resets to 0 seconds the count on an active interaction timer in the Agent interface. Has no effect if invoked when a timer is not active. Prerequisite: Requires a timer to be defined as a data element in the script information area (static panel). Note lower case initial capitalization of parameter timerName, following standard parameter naming conventions.	void	Proxy String timerName
getUser	Returns the user name of the individual running the script in a string called User	String	Proxy
checkReturnStatus	Boolean expression that checks to see if the status of a command returns a value of S indicating success. If S, returns True; otherwise, returns False.	boolean	Proxy
getCursorColumn	Returns the corresponding value of the column in the cursor, based on the index that is passed in the i parameter for an integer.	String	Proxy integer "i"
findAndSetCurrentRecord	Finds the current record in the scripting cursor and sets the pointer to that record, based on what was selected in the question. By default, any access to the scripting cursor points to the first record. After calling this method, any future access to the cursor record will return values from the new row where the pointer is located. Parameter rowid changed to rowID to follow standard parameter naming conventions	void	Proxy String rowId String columnName
setCursorColumnToBB	Sets the corresponding value of the cursor column specified in the columnName parameter to the Blackboard variable specified in the parameter key.	void	Proxy String key String columnName

Method Name	Description	Return Type	Parameters
getIndexforColumn	This method returns the numeric index (0,1, 2....) in the cursor for the columnName parameter passed.	int	Proxy String columnName

NodeDefault Java Methods

The table below describes available Java methods and their parameters.

Method Name	Description	Return Type	Parameters
getTimeOfDayForGreeting	Returns the string *morning*, *afternoon*, or *evening* derived from the current time (based on SYSDATE) so that you can set up a greeting as an embedded value.	String	
getTimeOfDayForClosing	Returns the string *day*, *afternoon*, or *night* derived from the current time (based on SYSDATE) so that you can set up a closing using an embedded value.	String	
currentDate	Returns current date in format MM/dd/yyyy.	String	
currentTime	Returns the time in format hh:mm:ss am/pm.	String	

Method Name	Description	Return Type	Parameters
setStaticInfo	Allows you to set values in the script information area (displays in Agent interface only). Requires you to pass the fieldID and its value value as parameters. Replaces method SetStaticInfo. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter FieldID changed to fieldID and parameter Value changed to value to follow standard parameter naming conventions.	void	Proxy String fieldID String value
SetStaticInfo	*Decremented*. Allows you to set values in the Script Information panel (displays in Agent interface only). Requires you to pass the FieldID and its value as parameters.	void	Proxy String FieldID String value

NodeValidation Java Methods

The table below describes available Java methods and their parameters.

Method Name	Description	Return Value	Parameters
validateForDateFormat	Boolean expression that checks for a date in format MM/dd/yyyy. If a different format is found, returns an error message asking the user to enter the date in the correct format. Private static parameter dateFormat changed to DATE_FORMAT to follow standard parameter naming conventions.	boolean	String answer

Method Name	Description	Return Value	Parameters
validateForTimeFormat	Boolean expression that checks for time in format hh:mm:ss am/pm. Returns an error if not in the correct format. Private static parameter timeFormat changed to TIME_FORMAT to follow standard parameter naming conventions.	boolean	String answer
validateDateNotInPast	Boolean expression that checks to see that the date entered (in format MM/dd/yyyy) is in the future. If false, provides an error message to the user with a prompt to enter a valid date not in the past. Private static parameter dateFormat changed to DATE_FORMAT to follow standard parameter naming conventions. Changed error message from Please enter a valid date/time... to Please enter a valid date....	boolean	String answer
validateIntegerEntered	Boolean expression that checks to see if a blackboard key has an invalid value. Returns True indicating an invalid value if blackboard value is null, blank, contains only spaces, or is a string of -1 indicating an exception. Otherwise returns False indicating a valid value.	boolean	String answer String label
validateIntegerInRange	Boolean expression that allows you to enter a minimum and maximum range, and subsequently checks that the user's entry is within that range.	boolean	String answer String minStr String maxStr String label
validateRequiredField	Boolean expression that checks to see if a value is entered for the designated field. If null or empty, provides an error message to the user with a prompt to enter a value for that field.	boolean	String answer String label

Method Name	Description	Return Value	Parameters
checkNullValue	Boolean expression that checks to see if a blackboard key has a null value. Returns True if null; otherwise returns False.	boolean	String answer
checkInvalidValue	Boolean expression that checks to see if a blackboard key has an invalid value. Returns True indicating an invalid value if blackboard value is null, blank, a space, or is a string of -1 indicating NotInInteractionException or InvalidCursorException. Otherwise returns False.	boolean	String answer String bbkey
enableButton	Enables a button on the shortcut bar (displays in Agent interface only). Replaces method EnableButton. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter ButtonID changed to buttonID to follow standard parameter naming conventions.	void	String answer buttonID
EnableButton	*Decremented*. Enables a button on the shortcut bar (displays in Agent interface only).	void	String answer ButtonID

Method Name	Description	Return Value	Parameters
disableButton	Disables a button on the shortcut bar (displays in Agent interface only). Replaces method DisableButton. Changes include: • Method name changed to begin with lower case to follow standard Java method naming conventions. • Parameter ButtonID changed to buttonID to follow standard parameter naming conventions.	void	String answer buttonID
DisableButton	*Decremented*. Disables a button on the shortcut bar (displays in Agent interface only).	void	String answer ButtonID

Referencing Best Practice Java in a Command

In Script Author graphical scripts, commands are associated with a branch or with an object (panel, group, block, or the entire script object itself). Commands are defined in the Command Properties window and execute the specified action at runtime, affecting the objects to which they are associated.

Use this procedure to define a Java command in the Script Author that references best practice Java methods included with Oracle Scripting.

Prerequisites

- Script Author release 11.5.6 or later.
- Oracle Applications release 11.5.6 or later.
- Identify the name and location of the appropriate Java method you wish to reference in the command.

Steps

1. Using the Script Author, open a new or existing graphical script.
2. Navigate to the object for which you wish to associate a command.
3. Open a Command window.

4. From the Command Type list, select Java Command.

5. In the Name field, enter a unique name for this Java command.

6. In the Command field, enter all of the following, concatenated and with no spaces:

    1. The fully qualified Java class name, including package path.

    2. Two colons.

    3. The specific Java method you are referencing for this command.

    Thus, if the class you are referencing is in the package oracle.apps.ies.bestpractice, the class is named SampleClass, and the method is SampleMethod, the full value to enter in the Command field is:

    ```
    oracle.apps.ies.bestpractice.SampleClass::SampleMethod
    ```

7. If the signature for the desired Java method indicates that parameters are required, do the following:

    1. Under Parameters, click **Add**.

    2. In the Name field, type the name of the command parameter.

        If the method includes the ServerProxyBean as a parameter, type "Proxy" (case-sensitive) in the Name field. No other values are required for this parameter.

    3. In the Value field, type the case-sensitive value for the parameter referenced in the Java signature.

        If a string, type the literal string value.

        If the value is a boolean, type True or False.

    4. If you want to add the value as the return value of an executed command:

        Select **Add Value as Command**.

        From the Value field, click **Edit**.

        Enter the parameters for the new command, save, and click **OK** to save the command.

    5. Click **OK** to save the values entered as parameters.

8. If you want to add a return value to the top-level command:

    1. Click **Edit** in the Return Value area.

        The Parameters window appears.

2. Enter appropriate return value parameters, save, and exit.

9. Click **OK** to save the values entered in this command.

10. Save your work. When deployed, the specified Java method will be called at runtime.

Passing Parameters to the Web Interface

When executing scripts in the Scripting Engine Web interface, you can pass parameters into an Oracle Scripting session from external applications. These parameters are stored in the Scripting blackboard, and are available for use throughout the script session. This method is appropriate regardless of whether you run scripts as surveys, or from a self-service Web application such as Oracle iSupport.

Using methods appropriate to your source application, pass each parameter to the Scripting blackboard by concatenating each key and value to the runtime survey URL required to initiate a script session.

After the last parameter required as part of the URL to initiate the script transaction, pass in values from external applications by concatenating an ampersand (&), the key name, an equal sign (=), and the key value to the URL required to initiate the survey.

Thus, if you want to pass an Oracle iSupport transaction ID (for example, 999) to a Web script launched from iSupport, you can append to the survey URL the key value pair transaction_id=999. This will then be accessible as a blackboard value from the script with and the key/value pair (transaction_id, 999).

Initial URL May Differ

The initial runtime survey URL used in your environment may differ based on method of execution for your environment. For example:

- The initial JSP page called may be IESSVYMENUBASED.JSP if hosted on a Web-based application.
- The initial JSP page called may be IESSVYMAIN.JSP if executed as a survey.
- Each Scripting Engine transaction using the Web interface requires at least one parameter, dID (deployment ID). List-based surveys also require a unique rID (respondent ID) parameter.

Syntax and Examples

The appropriate syntax follows. This includes two additional parameters (Keyname 1 and Keyname 2):

```
http://<hostname>:<port>/OA_HTML/<hosted or stand-alone survey
jsp>?<Deployment
ID>&<Respondent ID>&<Keyname1=<Value1>&<Keyname2>=<Value2>
```

An example of a list-based stand-alone survey URL passing a customer ID of 12345 and a customer name of Smith appears as follows:

```
http://server1.company.com:7777/OA_HTML/iessvymain.jsp?dID=263&CUSTOMER_
ID=12345&CUSTOMER_NAME=Smith
```

An example of a hosted survey URL passing a Party ID of 1000 and a customer name of Chan appears as follows:

```
http://server2.company.com:8888/OA_HTML/iessvymenubased.jsp?dID=374&PART
Y_ID=1000&CUSTOMER_NAME=Chan
```

The limit to parameters that can be passed is restricted by the Web browser's supported limit of URL characters.

Performing Advanced Graphical Script Tasks

This section describes various tasks you may wish to perform to provide additional functionality to a script at runtime. Some tasks involve manipulating aspects of the Scripting Engine agent interface. As such, these tasks are not applicable to scripts developed for execution in the Web interface. If a task is applicable to the agent interface only, this information is clearly described.

Defining a Shortcut Button

Shortcut buttons display in the agent interface at runtime, and present as a horizontal row of buttons immediately above any panels rendered. In order for a Shortcut button to function, the script must have a group with the appropriate Shortcut property.

Clicking on a shortcut button executes any Script Author command associated with the button. For example, you can jump in the script to a destination group anywhere in the script (identified by the Shortcut property of a group). A button can also call any other Script Author command For example, a button can be programmed to execute the Forms Goto URL command to access a web page in a new browser window.

Defining a Shortcut Button to Execute a Command

Use this procedure to define a Shortcut button in the button bar to execute any Script Author command.

Steps

1. Using the Script Author, open a new or existing graphical script.
2. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.
3. In the script properties menu, select the **Shortcut Panel** attribute. Shortcut Panel properties appear.
4. Click **Add**. The Shortcut Info Entry window appears.

5. Enter Shortcut information as appropriate.
    1. In the **ID** field, enter a unique identification name for this shortcut.
    2. In the **Label** field, enter label that you want to appear on the shortcut button.
    3. In the **Tooltip** field, enter the contents of the tooltip, which will appear in the Scripting Engine agent interface when the agent's cursor rests on the button.
    4. On the Command line, click **Edit**.

       The Command Properties window appears.
6. Enter the appropriate information to reference the command you want to execute at runtime when the Shortcut button is clicked.
7. Click **OK** to save the Command Properties and close the window.
8. To enable the Shortcut button upon the launch of the script, select **Enabled**.

   A check appears, indicating the button is enabled.

   > **Note:** You may also use Java command to enable a Shortcut button at a later time during the flow of the script, or to disable a Shortcut button at a specified point in the flow.
9. If you want to save your work and continue, click **Apply**.
10. To add additional Shortcut buttons, click **Add** in the Shortcut Panel properties and repeat the required steps.
11. Click **OK** to save your work and exit the Shortcut Info Entry window.

Defining a Shortcut Button to Jump to a Group

Use this procedure to define the Java code and a shortcut button to jump to a group. This can be accomplished using the JumpToShortcut Java method in the best practice ScriptUtil class, or by writing a custom Java method using the jumpToShortcut Scripting API.

Sample Code

The following sample Java method establishes a destination group by its Shortcut property.

```
public void shortcut1(ServerProxyBean pb, String DestinationGroup)
{
try
{
pb.jumpToShortcut(DestinationGroup);
}
catch (Exception e)
{
e.printStackTrace();
}
}
```

In this example, Shortcut 1 is the method to associate with a Shortcut button. When shortcut1 is evaluated, the flow of the script will jump to the group with the Shortcut name DestinationGroup. The first panel in that group will display at runtime.

Prerequisites

- Create the group intended as the destination of the jump. Ensure you provide a Shortcut property. The group must be syntactically correct:
  - It must contain a Start node, Termination node and appropriate branching between all objects.
  - Any panels in the group must have answers defined. If distinct branching is used, one answer definition in that panel must be set to trigger as the default for distinct branching.
- Write the code for the Java method, following the guidance above. The method must name the destination group by its Shortcut name.

Steps

1. Using the Script Author, open a new or existing graphical script.
2. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.
3. In the script properties menu, select the **Shortcut Panel** attribute. Shortcut Panel properties appear.
4. Click **Add**.

   The Shortcut Info Entry window appears.
5. Enter Shortcut information as appropriate.
   1. In the **ID** field, enter a unique identification name for this shortcut.
   2. In the **Label** field, enter label that you want to appear on the shortcut button.
   3. In the **Tooltip** field, enter the contents of the tooltip, which will appear in the

Scripting Engine agent interface when the agent's cursor rests on the button.

4. On the Command line, click **Edit**.

   The Command Properties window appears.

6. From the Command Type list, select **Java Command**.

7. In the **Name** field, enter a unique name for this Java command.

8. To define a shortcut button using the JumpToShortcut method in the best practices Java:

   1. In the Command field, type

      oracle.apps.ies.bestpractice.ScriptUtil::JumpToShortcut

   2. Under Parameters, click **Add**.

      The Parameters window appears.

   3. In the Name field, type **Proxy**.

      No other values are required for the ProxyServerBean parameter.

   4. Click **OK** to save the Proxy parameter and close the window.

   5. Under Parameters, click **Add**.

      The Parameters window appears.

   6. In the Name field, type the name of the command parameter.

      Type the value **ShortcutName**. This is the string used as a parameter by the JumpToShortcut method.

   7. In the Value field, type the value of the shortcut property associated with the group you wish this button to jump to.

      For example, to jump to a group with the shortcut property Group1, type Group1 in the Value field.

      > **Caution:** This field should contain the value of the Shortcut property, not the name of the group.

   8. Click **OK** to save the Proxy parameter and close the window.

9. To define a shortcut button using a custom Java method invoking the jumpToShortcut API as in the sample code above, then do the following:

1. In the Command field, type the fully qualified name of the custom Java class, two colons, and the Java method, concatenated and without spaces.

   For example, assuming the method in the example above is in the class CustomCode and is appropriately deployed to the database in the path <JAVA_TOP>/ies_custom, type the following in the command field.

   ```
   <JAVA_TOP>.ies_ custom.CustomCode::shortcut1
   ```

2. Under Parameters, click **Add**. The Parameters window appears.

3. In the Name field, type **Proxy**. No other values are required for the ProxyServerBean parameter.

4. Click **OK** to save the Proxy parameter and close the window.

   Since the value of the shortcut group is determined by the custom Java method, no additional parameters are required unless required by the custom method.

10. Click **OK** to save the Command Properties and close the window.

11. To enable the shortcut button upon the launch of the script, select **Enabled**.

    A check appears, indicating the button is enabled.

    > **Note:** You may also use Java command to enable a Shortcut button at a later time during the flow of the script, or to disable a Shortcut button at a specified point in the flow.

12. If you want to save your work and continue, click **Apply**.

13. To add additional Shortcut buttons, click **Add** in the Shortcut Panel properties and repeat the required steps.

14. Click **OK** to save your work and exit the Shortcut Info Entry window.

Defining Shortcuts

A Shortcut is the property of a Group in the Script Author. Using Java commands, the flow of a script in runtime can jump to a "target" group containing a Shortcut of a specified name. The group contains the panels that are the destination that result when clicking the Shortcut button. Three typical uses of a Shortcut are to associate a target for a Shortcut button in the Shortcut button bar (for the agent interface only), to enable the Disconnect button (for the agent interface only), and to associate a target for a group containing specific functionality which must be accessed after meeting a specified condition in a script (for either runtime interface).

Use this procedure to define a group as the target of a jump.

Prerequisites

Insert a group.

Steps

1. Using the Script Author, open a new or existing graphical script.
2. In the tool palette, click the **Toggle Select Mode** tool.
3. On the canvas, double-click a group.

   The Properties window appears.
4. In the Group tree, select **Shortcut**.
5. In the Shortcut field, type the name of the shortcut.

   For example, to define a group as the target after clicking **Disconnect** in the agent interface, type WrapUpShortcut.
6. Click **Apply** to save your work and continue, or click **OK** to save your work and exit the Properties window for the group.

   If you want to jump to the Shortcut in runtime from a Shortcut button, define the Shortcut button.

If you want to jump to the Shortcut in runtime based on values or other criteria, define that criteria and associate the criteria with one or more commands in the script.

If you created this group to enable the Disconnect button in the agent interface, you must ensure the subgraph created by the group is appropriately terminated, with (at minimum) a default branch from the Start node to a Terminal node.

Enabling the Agent Interface Disconnect Button

The Disconnect button appears at the bottom right of the Scripting Engine agent interface, in the outermost frame of the window. The purpose of this button is to allow agents using Oracle Scripting to bring the flow of the script to a close from any point in the script. When this button is clicked at runtime, the agent is "jumped" in the flow of the script to a specified destination group.

For scripts created using the Script Wizard, the Disconnect button is created automatically, by placing a group named Disconnect on the canvas. This subgraph represented by this group contains default branching to a Termination node, and does not contain any panels. The Shortcut property contains the WrapUpShortcut designation.

For graphical scripts, the Disconnect button is not enabled until you create a destination group (or designate an existing group as the destination group), provide the appropriate content for that group (optional objects and mandatory termination), and provide the appropriate Shortcut property designation.

Note: Unlike other instances of jumping the end user to a specified group, enabling the Disconnect button does not require any Java to be created.

The destination group routed to from the Disconnect button must:

- Be placed on the root graph of the script.
- Contain the shortcut name WrapUpShortcut (this is case-sensitive).
- Be properly terminated on the root graph, with a default branch leading to a Termination node on the root graph.
- Be properly terminated in the child graph within the destination group.

To immediately end the script session, include no panels in the destination group. Simply add a Termination node and default branching. Alternatively, to capture information required for every session, include panels, blocks, or child groups within the destination group. For example, for an inbound script, you can include a panel that thanks the user for calling, and inquiring how the caller heard about this particular promotion. For outbound scripts, you may wish to establish callback information in the group.

Based on the approach taken (including wrap-up information, or immediately ending the script session), this destination group is generally referred to either as the WrapUp group or the Disconnect group. Use this procedure to define a group as the destination target for when an agent clicks Disconnect in runtime.

Prerequisites

None

Steps

1. Using the Script Author, open a new or existing graphical script.
2. If you want to create a new group as the target destination of the Disconnect button, then do the following:
    1. In the tool palette, click the **Group Insertion Mode** tool.
    2. On the root graph of the canvas, click where you want to insert the group.
    3. In the tool palette, click the **Toggle Select Mode** tool.
    4. Double-click the group you just created.

       The Properties window appears.
3. If you want to designate an existing group as the target of the Disconnect button,

then do the following:

1. In the tool palette, click the **Toggle Select Mode** tool.
2. On the root graph of the canvas, double-click a group.

   The Properties window appears.

4. In the Group tree, select **Properties**.

   Properties field for the group display.

5. In the Name field, type a meaningful name.

   For example: if clicking **Disconnect** at runtime must immediately end the current script session, type Disconnect; if clicking **Disconnect** at runtime must result in displaying panels wrapping up the transaction, type **WrapUpGroup**.

6. Click **Apply** to save your work and continue.
7. In the Group tree, select **Shortcut**.
8. In the Shortcut field, type **WrapUpShortcut**.

   Note: This value is case-sensitive and must be entered precisely.

9. Click **OK** to save your work and exit the Properties window for the group.
10. Click **OK** to save your work and exit the Properties window for the group.
11. Click **OK** to save your work and exit the Properties window for the group.
12. If the destination group on the root graph is not yet connected via default branching to a Termination node, do the following:
    1. In the tool palette, click the **Termination Point Insertion Mode** tool.
    2. On the root graph of the canvas, click where you want to insert the Termination node.
    3. In the tool palette, click the **Default Branch Mode** tool.

       When the pointer is over the canvas, the pointer is a cross hair (+).

    4. On the canvas, drag the pointer from the destination group to the Termination node.

13. If clicking **Disconnect** at runtime must immediately end the current script session, do the following:

1. In the tool palette, click the **Toggle Select Mode** tool.
2. On the root graph of the canvas, click once to select the group.

   The Properties window appears.
3. In the tool palette, click the **Go down into child graph** tool.

   The subgraph representing the target group appears.
4. In the tool palette, click the **Termination Point Insertion Mode** tool.
5. On the canvas, click where you want to insert the Termination node.
6. In the tool palette, click the **Default Branch Mode** tool.

   When the pointer is over the canvas, the pointer is a cross hair (+).
7. On the canvas, drag the pointer from the Start node to the Termination node.
8. In the tool palette, click the **Go up to parent graph** tool.

   The canvas refreshes, displaying the root graph.

   The subgraph representing the target group appears.

14. If clicking **Disconnect** at runtime must result in displaying panels wrapping up the transaction, and if this functionality does not already exist in the WrapUp group, do the following:
    1. In the tool palette, click the **Toggle Select Mode** tool.
    2. On the root graph of the canvas, click once to select the group.

       The Properties window appears.
    3. In the tool palette, click the **Go down into child graph** tool.

       The subgraph representing the target group appears.
    4. Add panels, groups, or blocks appropriately to provide the desired functionality for the WrapUp group at runtime.
    5. In the tool palette, under the last appropriate object, click the **Termination Point Insertion Mode** tool.
    6. On the canvas, click where you want to insert the Termination node.
    7. In the tool palette, click the **Default Branch Mode** tool.

       When the pointer is over the canvas, the pointer is a cross hair (+).

8. On the canvas, drag the pointer from the last appropriate object in your WrapUp group to the Termination node.

9. In the tool palette, click the **Go up to parent graph** tool.

   The canvas refreshes, displaying the root graph.

   The subgraph representing the target group appears.

15. Save your work.

Enabling or Disabling a Shortcut Button

Upon defining a Shorctut button, you can choose whether to enable it immediately or at a later point in the script using a Java command. In the same manner, at any point in the script, you can use a Java command to disable a Shortcut button.

Use the following procedure to enable or disable a shortcut button using the best practice Java methods enableShortcutButton and disableShortcutButton in the class ScriptUtil.

Prerequisites

- For this procedure to have any effect, a Shortcut button must be programmed to display in the Shortcut button bar in the agent interface at runtime.
- You must know the ID of the Shortcut button.

Steps

1. At the appropriate location in a graphical script, navigate to a Command Properties window.
2. In the Command Type area, select Java Command.
3. In the Command Info area, type in the Name field a unique name for this Java command.
4. In the Command field, type the fully qualified Java command of the appropriate Java method.

   If you want to enable a Shortcut button, enter

   ```
   oracle.apps.ies.bestpractice.ScriptUtil::enableShortcutButton.
   ```

   If you want to disable a Shortcut button, enter

   ```
   oracle.apps.ies.bestpractice.ScriptUtil::disableShortcutButton.
   ```

5. In the Parameters area of the Command Properties window, click **Add**.

   The Parameters window appears.

6. In the Class field, leave this value blank.

7. In the Name field, type **Proxy** in exact case.

8. In the Value field, leave this value blank.

9. In the In/Out list, make no selection. This field is not configurable at this time.

10. In the Add Value as Command checkbox, leave this unselected.

11. Click **OK**.

    The Parameters window closes and the new command parameter is listed as a parameter to this command.

12. In the Parameters area of the Command Properties window, click **Add**.

    The Parameters window appears.

13. In the Class field, leave this value blank.

14. In the Name field, type shortcutPanelID in exact case.

15. In the Value field, enter the ID of the Shortcut button.

16. In the In/Out list, make no selection. This field is not configurable at this time.

17. In the Add Value as Command checkbox, leave this unselected.

18. Click **OK**.

    The Parameters window closes and the new command parameter is listed as a parameter to this command.

19. Click **Apply** to save your work and continue, or click **OK** to save your work and exit the Command Properties window.

Defining the Script Information Area

If explicitly defined in a script, the script information area appears immediately below the Oracle Applications toolbar in the Oracle Scripting agent user interface.

The script information area is also known as the Static Information Panel or the Static Panel, although information displayed may contain dynamic information. The script information area is a global script property of a graphical script, and displays for all sessions of a script in which it has been defined. There are no facilities for creating a script information area in a wizard script.

A script executed in the agent interface at runtime can display up to three rows of

information. Each row has a left, middle, and right position, for a total of nine possible data element locations. Optionally, each data element may contain a label. Data element types include text or timer.

Text Data Elements

For text data elements, the value can be passed to the script information area at runtime in two ways:

1. You can provide a constant value using a Constant command.

2. You can dynamically pass a value from the Blackboard. This allows you to hard-code a value or insert a value retrieved using a PL/SQL query.

Timer Data Elements

Timers defined in the script information area will need to be activated elsewhere using a Java command. If using the best practice Java methods, use the StartTimer method in the ScriptUtil class. This command must not be a pre-condition to the script. You can start the timer in one of two ways:

1. You can start the timer by referencing the appropriate Java method prior to the first panel, in order to display the timer beginning with the first panel displayed.

   Any commands associated as an action with the outgoing branch of the Start node will also be ignored. Any other location where a command can be defined is appropriate. For example, you can reference the method as a precondition to the first panel, or in a Block referencing an API as the action.

2. You can start the timer by referencing the appropriate Java method at a later point in the script, in order to time a specific portion of the script.

More than one timer can be used in the Script Information panel. The best practices methods include a StopTimer method which can be associated as a command to a button on the Shortcut button bar or as an action anywhere in the script. To use best practice Java to start the timer, use the following command

```
oracle.apps.ies.bestpractice.ScriptUtil::StartTimer
```

Use this procedure to define the script information area with text, timers, and labels.

Prerequisites

None

Steps

1. Using the Script Author, open a new or existing graphical script.

2. Double-click on an empty area of the canvas, or select **Script Properties** from the File menu, to access properties of the global script object.

3. In the script properties menu, select the **Static Panel** attribute.

Static Panel properties appear as a matrix (up to three rows, each with a left, center, and right position). Each cell represents the position where a defined set of data elements appears at runtime.

> **Note:** To reduce the amount of screen space consumed by the script information area on an agent's desktop, define your data elements in rows. Unused rows will not display in runtime, providing more viewing area for interaction center agents.

4. Click in any cell to define a data element in that position at runtime.

   The Text and Timer buttons are enabled.

5. Click **Text** or **Timer**, as appropriate.

6. The matrix refreshes, indicating the selected region and data type. The data fields below the matrix are enabled.

7. Enter parameters for this data element as appropriate.

   1. In the ID field, enter a unique identification value.

      The Oracle Scripting API requires this ID.

   2. In the Label field, enter the name you wish to appear as the label for this data element, if any.

   3. If you want to pass a command to this data element, in the Command field, click **Edit**.

      The Command Properties window appears.

      > **Note:** Do not associate a Java command with a timer data element. The Java command required to start the timer must not be defined as a global script property.

   4. Define the command as appropriate. When complete, click **OK** to save the command and return to the Static Panel properties.

   5. Click **OK** to save changes to the Command Properties window.

      The matrix in the Static Panel properties displays the data element.

8. To save your work and define additional script information area attributes, click **Apply** in the script properties window.

9. To save your work and exit the script properties window, click **OK**.

Related Topics

For information on passing a constant value to the script information area in the agent interface at runtime, see Setting a Constant in the *Oracle Scripting User Guide*.

For information on using a Java command to start the timer, see Invoking a Public Method in a Java Class in the *Oracle Scripting User Guide*.

Defining a Constant Command

Follow this procedure to define a constant command in a graphical script.

> **Note:** Providing an entry in the Default Value field of the Define Question Detail window of a wizard script will automatically provide a constant command for a panel question at runtime.

Prerequisites

None

Steps

1. Using the Script Author, open a new or existing graphical script.
2. At the location where you wish to provide a constant value as a command, access the Command Properties window. For more information, see Determining Where to Define a Command, page 2-5.

   The Command Properties window appears.
3. From the Command Type list, select **Constant Command**.
4. In the Command Info area, type the name of the command in the Name field.
5. In the Return Value area, click **Edit**.

   The Parameters window appears.
6. In the Class field, leave this value blank.
7. In the Name field, leave this value blank.
8. In the Value field, enter the value you wish to pass as a constant.

   For example, type **Customer Service Follow-Up**.
9. In the In/Out list, make no selection. This field is not configurable at this time.
10. In the Add Value as Command check box, leave this deselected.

11. Click **OK**.

    The Parameters window closes and the new command parameter is listed as a return value to this command.

12. Click **OK** to save changes to the Command Properties window.

    The value entered as a return value to this command will be passed at runtime. For example, if you passed the value of Customer Service Follow-Up as a label in the script information area, this value appears in the script information area of the agent user interface at runtime.

Using the Indeterminate Branch

Using the indeterminate branch and an associated Java method, you can route the flow of a graphical script at runtime based on conditions tested for in your custom code. You can *jump* to a sibling object on the graph, or to a group on any graph in the script based on its shortcut property.

Like all commands, the expression associated with the indeterminate branch is evaluated at runtime when the corresponding Script Author object is reached in the flow of the script. When a condition tests true, the target destination provided by the return statement becomes the next object processed in that script interaction.

Jumping to a Sibling Object

A sibling object is any Script Author object appearing on the same graph. If you have only one graph in a script (the root graph), every object is a sibling object. With the indeterminate branch, you can use any configurable object (a panel, group, or block) as the destination, using the name of the object. Only panels contain elements that are displayed at runtime, but if a group or block is the destination, the processing associated with that group or block then occurs. If a panel is the destination object, that panel's display objects (any panel layout content, and at least one answer control) appear to the script end user.

Jumping to a Group

To jump to a group on a different graph using the indeterminate branch, you must designate a shortcut within the target group, and return the shortcut name in the associated expression.

Oracle recommends using this feature sparingly, because it breaks rules that all other branches (default, distinct, and conditional) follow, and introduces great complexity in debugging script flow issues. Using the indeterminate branch to jump to a group in a different graph of the script is similar in approach and result to using a GoTo statement in other programming languages. Oracle Scripting script development follows a strong paradigm of flow based on expected conditions, and any other instance of script flow is easily traceable, whereas using this approach is not.

Indeterminate Branch Exceptions

Graphs that contain an Indeterminate branch introduce two exceptions to general Script Author syntax rules:

1. A graph with an Indeterminate branch need not adhere to the requirement for a Termination node in order to pass a syntax check. This assumes the Indeterminate branch is the last object that will be evaluated on that graph.

2. When using an Indeterminate branch, you may use panels or groups that have no incoming branches, assuming these objects are to be reached no other way than as the destination of a jump.

3. If an object breaks incoming branch requirements because it is the destination of an indeterminate branch jump, standard syntax and branching rules for all subsequent objects still apply (beginning with the outgoing branch from the destination object).

For more information on understanding graphs, see the *Oracle Scripting User Guide*.

Sample Expressions Using Return Statement

The following are sample expressions using a return statement. The jumpToDestination Java method evaluates the contents of a string populated in the argument "answer" and routes the script accordingly.

```
public String jumpToDestination(ServerProxyBean pb, String answer)
{
if (answer.equals("condition_A"))
{
return "destination_A";
}
if (answer.equals("condition_b"))
{
return "destination_B";";
}
else
{
return "destination_C";
}
}
```

- If the first condition is found (if the value of the string answer is equal to condition_A), the flow of the script will continue with the processing of the destination_A object.

- If the second condition is found (if the value of the string answer is equal to condition_A), the flow of the script will continue with the processing of the destination_B object.

- If the third condition is found (if the value of the string answer is equal to any value other than those tested in the previous conditions), the flow of the script will continue with the processing of the destination_C object.

- If a listed destination is a sibling object (is on the same graph as the Indeterminate branch), it can be a panel, group, or block.

- If a listed destination is on any other graph in the script, it must be the Shortcut name of a group in order to evaluate correctly at runtime.
- The above method is just a sample. While this sample tests the contents of a variable, any condition can be tested for and evaluated.

Prerequisites

Any panel, group or block object must exist on the canvas in order to use the indeterminate branch.

Write the code for the expression (typically a Java method), following the guidance above. The expression must name the destination object or objects, based on each condition tested.

Steps

1. Using the Script Author, open an existing graphical script.
2. Navigate to an object for which you want to define an Indeterminate branch.

   This object must currently have no outgoing branches
3. In the tool palette, click the **Indeterminate Branch Mode** tool.
4. On the canvas, click on the source object and drag outwards about two inches in any direction.

   The Indeterminate branch appears.
5. Double-click on the branch object or right-click the branch and select **Edit Branch Properties**.

   The properties window for the Indeterminate branch appears.
6. Optionally, in the Indeterminate tree, select **Properties**.

   In the Properties pane, two fields appear.

   1. In the Name field, enter a meaningful name for this branch.
   2. In the comments field, enter any descriptive comments if desired.
   3. Click **Apply** to save your changes.
7. In the Indeterminate tree, select **Expression**.

   In the Expression pane, click Edit. The Command window appears.
8. Define the indeterminate expression as a command.

   The command must return the name of a sibling object (a panel, block or group on the same graph as the source object) and the shortcut name associated with a group

on any hierarchical level of the script.

9. Click **OK** to exit the Command window.

10. Click **Apply** to apply your work or click **OK** to save your work and exit the Properties window.

Using Iterating Parameters

The Command Parameter window for graphical scripts has an Iterating Parameter check box. Selecting this parameter enables the Scripting Engine to retrieve answers from a vector. This is required when retrieving answers from one of the new multiple-selection answer UI types (check box group or multi-select list box) and passing the answers to a predefined API.

Business Rules Governing Iterating Parameters

The business rules governing iterating parameters include:

1. Each command can contain at most only a single iterating parameter.

2. The iterating parameter can only be used with Script Author commands that take arguments (Java, SQL, and Forms commands).

3. The single iterating parameter can be a parameter to the top-level command or to a nested command (a command passed as the parameter to another command).

4. A Parameter can only be marked as an iterating parameter if both of the following conditions are true:

   - The parameter is a nested command (Add Value as Command? is selected).

   - The nested command can return a vector.

   The following command types are supported: Blackboard, Java, SQL (with iterating parameter), and Forms commands (with iterating parameter).

These rules are enforced in the Script Author, either via the Command and Parameter windows or by the Syntax Checker.

Iterating Parameters in the Scripting Engine

The following describes the behavior of the iterating parameter in the Runtime Scripting Engine:

During script runtime, the Scripting Engine (regardless of execution interface) attempts to evaluate each parameter of a Script Author command. If the parameter is marked as an iterating parameter, the Engine verifies that the value of the parameter is a vector. Upon confirmation, the Engine iterates through the vector, executing the command

once for each element in the Vector. Each element in the vector is passed to the command at the same place in the list of parameters as the iterating parameter.

If the iterating parameter's nested command does not return a vector at runtime, an exception is generated.

If the iterating parameter's nested command returns an empty vector at runtime, the command is not executed at all, and no error or exception is generated.

Just as the answer to a multi-select node is stored in the blackboard as a vector of individual selected items, setting the default value of a multi-select node must be done with a vector. Each element of the vector is matched with the value of the lookup choices of the node. If a match is found, that lookup choice is selected.

Querying the Database and Displaying Results

Using a block designated as a query, you can obtain data stored in any database tables accessible at runtime and display the information in a script.

After any successful query, values returned are stored temporarily in the scripting cursor. This data does not persist beyond the next query, or the end of the interaction, whichever comes first. Prior to the next query, you can retrieve data from the cursor to display in a script, or write that data to the Scripting blackboard.

You can access data retrieved by a database query in the following ways.

- Displaying returned values as panel text
- Displaying returned values as panel answer lookup values
- Writing returned values to the scripting blackboard
- Displaying the single record referenced by the cursor as panel text
- Displaying all records retrieved by the query as lookup values to an answer
- Writing retrieved records to the scripting blackboard

Some of these ways are documented below.

Displaying Returned Values as Panel Text

Information stored in the cursor as the result of a query can be displayed in a panel as an embedded value using a blackboard command. If a database query contains one or more constraints designed to ensure a single row is returned, you need only reference the column name in the Command field of a blackboard command in order to display the value. If more than one row is returned by a query you may be required to advance through the rows, testing the values until you ascertain the one which you wish to display. This requires custom Java commands executing cursor APIs. For more information on storing and referencing information returned by a query, see

Understanding the Scripting Cursor, page 2-49.

Displaying Returned Values as Panel Answer Lookup Values

Information stored in the cursor as the result of a query can be used to populate panel answers.

Determine in Advance the Amount of Values to Display

To populate a single-value UI type, you must ensure that your query is structured to return only a single row, because regardless of how many records are returned as a result of a query, the cursor will only point to the first row. This applies to the password field or standard boolean checkbox UI types, as well as open-ended UI types such as the text field or text area.

> **Caution:** Do not use this procedure to populate the button answer UI type.

To populate multiple-value UI types such as radio buttons, drop-down list boxes, multi-select list boxes, or check box groups, your query should be structured to return one or more values. Typically, users expect at least two values to be provided for these choices. If a query at runtime is expected to populate an answer control, and the query returns no values, the script user will not be able to pass that panel and may not be able to complete the script.

Using Validation

To avoid creating a panel for which no answers can populate, and to preclude other problems, creating validation for each query is strongly recommended.

You can write any validation routine you desire, and use commands in the Script Author to execute validation. For example, you can use the Oracle Scripting API isCursorValid in a Java command to check if records were returned to the Scripting cursor. This API will pass a boolean value of True if the cursor contains records. If used in a conditional branch, the branch should flow to continued execution of the script upon returning True. Whenever a conditional branch is used, a default branch should also be used from the source object to branch the script should the condition evaluated return a value of False.

Working with a Stateful Transaction

When a query for a column name returns a value, that value is stored in the cursor as a key/value pair, with the key equivalent to the column name. Prior to the next query (if any), that value can be displayed or evaluated in the script using a blackboard command. The key to access that value is equivalent to the column name. If the same key is subsequently used as an answer name elsewhere in the same script, then you must exercise caution when testing the blackboard value to know where in the flow of the script you request the value. Since the blackboard maintains state, testing the blackboard value after the query will provide one answer, and testing the blackboard

after the panel has been reached and answered will return a second answer.

In some scripts, due to limited queries or rigid flow of the script, it is easy to make this distinction. Other scripts involve substantial "jumping" or cycling through various portions of the script. For example, a service script may ask after each customer concern is addressed if there is anything else the agent can assist the customer with. This would result in the same set of panels (typically collected in a group object) being accessed multiple times in a script. In such cases, one possible approach is to explicitly set a second blackboard answer that can be used as a control.

Use the example of the value CUSTOMER_NAME, which in a given script can potentially be populated both from a query and also from a panel answer. You can create a Java command as a post-action to the relevant panel only, that sets blackboard values (for example, the key/value pair of setCUSTOMER_ NAMEFromPanel/Y).

Any time you do a query in the script, you want to check both blackboard values (CUSTOMER_NAME and setCUSTOMER_NAMEFromPanel. If the second blackboard value is null, you know the value of CUSTOMER_NAME came from the initial query. If the second blackboard value is Y, you know the value was populated from the panel.

Prerequisites

- You must know the appropriate tables and columns to query and have appropriate database connection information and privileges.
- Queries can return zero, one, or more than one rows of information. If using a single-display UI type, you must use a query that will return a single value.
- If using a multiple-display UI type, you must use a query that will return at least a single value. It is recommended to use a query that will return at least two values.

Steps

1. In the Script Author tool palette, click the **Block Insertion Mode** tool.

   When the pointer is over the canvas, the pointer is a pointing finger.

2. On the canvas, click where you want to insert the block.

   The block appears. If the Popup on Blob Creation option is selected, then the Properties window for the object appears.

3. If the Sticky Mode option is selected, then in the tool palette, click the **Toggle Select Mode** tool.

4. On the canvas, double-click the block.

   The Properties window appears.

5. In the Block tree, select **Properties**.

6. In the Name field, type the name of the block.

This name should indicate the purpose for the query. For example, type getActiveScripts.

7. In the Comments field, type a comment if desired.

   Comments are optional.

8. Click **Apply** to save your work and continue.

9. In the Block tree, select **Types**.

   This is where you designate whether you want to insert, query, or update the database, or whether you wish to access an API or other command.

10. From the Types list, select **Query**.

11. Click **Apply** to save your work and continue.

12. In the Block tree, select **Object Dictionary**.

13. Enter appropriate information in the Connection/Tables tab:

    1. In the Connection area, define how Oracle Scripting connects to the database at runtime. To reuse an existing connection:

       - Select **Reuse an existing connection**.
       - From the Existing Connections list, select a connection.

       To establish new connection information:

       - If necessary, clear **Reuse an existing connection**.
       - Type the connection parameters.

    2. In the Tables area, for each table you wish to connect to:

       - Click **Add Table**.
       - The Table window appears. The text area will accommodate any length required. Alternatively, you can resize this window if desired.
       - In the Table Name field, type the name of the table.

         For example, type IES_DEPLOYED_SCRIPTS.

         Although this field is not case sensitive, it is standard to type table names in all upper-case letters.

       - Click **OK**.

3. Click **Apply** to save your work and continue.

14. If you want to access tables that require joins, then, in the Join Condition tab, do the following:

    - Click **Add**.

      The Join Condition window appears.

    - Identify primary key and foreign key information in the Master PK Table, Detail PK Table, and Detail FK Table tab.

    - Click **OK**.

15. In the Query Data tab, do the following:

    1. Click the Query Columns tab.

    2. For each table in your query, identify the column as indicated:

       - Click **Add Column**.

       - In the Enter a Key/Value window, type the name of the column.

         For example, type DSCRIPT_NAME to query the name of a script in table IES_DEPLOYED_SCRIPTS, and ACTIVE_STATUS to query the active status of the resulting script.

       - From the Data Type list, select the data type of the selected column.

         For example, for DSCRIPT_NAME (which has a VARCHAR2 data type), select VARCHAR, and for ACTIVE_STATUS (which has a number data type) select BIGINT.

       - Click **OK**.

       - Repeat as necessary for other columns

    3. If your query has any constraints, then, in the Query Constraints tab, click Add Constraint.

    4. Enter your constraint just as you would using a SQL statement.

       - Include all information that would follow the WHERE clause.

       - Include literal strings within single quotes.

         For example, type ANONYMOUS='YES' to include the constraint of "where the anonymous flag is set to yes." (Note the single quotes for the literal

string 'Yes'.)

As another example, type ID=1 to include the constraint of "where the specified identification number is equal to one." (Note the lack of single quotes.)

- Click **OK**.
- Repeat as necessary for other constraints.
- Each constraint added in the constraint window is automatically concantenated together during runtime with AND and prepended with WHERE. Optionally, the user can manually concatenate constraints into a single constraint by including a literal string such as "OR", "AND", or other accepted SQL conventions.

5. Users can reference blackboard values as query constraints using the syntax <BLACKBOARD KEY>. For example, type customer_id = :PARTY_ID to include the constraint of "where the customer_id is equal to the PARTY_ ID value set in the blackboard for this session."

   Note: Due to a limitation in the way blackboard key names are specified, there must be at least one space following each blackboard key name to ensure that your SQL functions. Thus, in the example above, type a space after :PARTY_ID before exiting the constraints field.

   Click **OK** to save properties for this block object.

   The block properties window closes. The configured block appears on the canvas.

16. Define objects (if required), termination point, and appropriate branching within the block.

    Panels within a block can be used to specify parameters to be used in the actual SQL procedure. In this example, no parameters are required. However, since a block introduces a child graph, you must provide appropriate branching and termination for the script to be syntactically correct.

    - If necessary, in the tool palette, click the **Toggle Select Mode** tool.
    - On the canvas, click the block you have defined, and then click the Go down into child graph button on the canvas tool bar.

      The graph for the block appears.

    - In the tool palette, click the **Termination Point** tool.

- On the canvas, click where you want to insert the Termination node.
- In the tool palette, click the **Default Branch Mode** tool.

  When the pointer is over the canvas, the pointer is a cross hair (+).
- On the canvas, drag the pointer from the Start node to the Termination node.

  When you release the pointer, the objects are connected.
- In the canvas tool bar, click **Go up to parent graph**.

17. In the tool palette, click the **Panel Insertion Mode Point** tool.
18. On the canvas, click where you want to insert the panel.

    This panel will use the results of a successful query (defined above) to populate an answer type.
19. On the canvas, double-click the panel.

    The Properties window appears.
20. In the Panel tree, select **Properties**.
21. In the Name field, type the name of the panel.
22. Optionally, in the Comments field, enter any comments.
23. In the Label field, type the label of the panel.

    The panel label is the value used to identify specific panels in the Progress Area of the Scripting Engine agent interface at runtime. This does not display in the Web interface.
24. Click **Apply** to save your work.
25. In the Panel tree, select **Answers**.
26. In the Answers pane, click **Add**.

    The Answer Entry window appears. Do the following:

    1. In the Answers pane, click **Add**.
    2. Select the Default for Distinct Branching option.
    3. In the Name field, provide an answer name.

       For example, type showDSCRIPT_NAME.

4. From the UI type list, select an appropriate answer UI type:
   - For queries returning one value, use Text Field, Text Area, Check Box, or Password, as appropriate.
   - For queries returning two or more values, use Radio Button, Drop Down, Check Box group, or Multi-Select List Box, as appropriate.

     For example, to display a list of active scripts, select Drop Down.
5. In the Label for reporting area, enter a label, if required.
6. Click **Edit Data Dictionary**.
7. In the data dictionary, click the Lookups tab.
8. From the Lookups area, select the Cursor Lookup option.
9. In the Cursor lookup display value field, enter any name.
10. In the Cursor lookup value field, enter the name of the database column that you want to display in this answer UI type.

    For example, type DSCRIPT_NAME.
11. Click **OK** to close the data dictionary for this specific answer.
12. Click **OK** to close the Answer Entry window.

    The Panel Properties window appears, listing the answer you have configured.

27. Define additional answers if desired.
28. Click **OK** to close the Panel Properties window.
29. Enter other objects into the script if desired.
30. In the tool palette, click the **Termination Point** tool.
31. On the canvas, click where you want to insert the Termination node.
32. In the tool palette, click the **Default Branch Mode** tool.

    When the pointer is over the canvas, the pointer is a cross hair (+).
33. On the canvas, connect the block object to the panel object, and the panel object to the Termination node. Connect other objects as appropriate.
34. Save your work.

Understanding the Scripting Cursor

The Scripting Cursor is a temporary memory space accessible only for the duration of a single scripting interaction. After any successful query, values returned (if any) are stored in the cursor.

Values retrieved are stored using the column names under which the data was stored in the database table.

Data Persistence

Any values stored in the cursor will be overwritten with values retrieved by any successive query in a single scripting interaction. Additionally, data does not persist in the Scripting cursor beyond a single scripting interaction. Thus, after performing a query, you must either use the required data prior to the next query, or write it to the Scripting blackboard. Writing data to the blackboard will enable you to use the data after any successive queries in that interaction. Note, however, that blackboard data also does not persist beyond the current scripting interaction. You always have the option of writing retrieved data to database tables for later access, if required.

Multiple Values, Single Reference

The full results of a query are stored in the cursor, whether one row is returned or many. By default, the cursor points to the first row in a set of returned values. The cursor can be advanced to the second or subsequent rows using specific cursor APIs.

Scripting Engine Cursor APIs

The Scripting Engine cursor is associated with the results of a SQL query executed on a database. The cursor maintains a current record whose contents can be retrieved and which can be advanced through each record in the result set returned by the query, using cursor APIs.

Method Name	Description	Return Type	Parameters
isCursorValid	Returns the boolean value True if the cursor contains results from a query; otherwise, returns False. Throws NotInInteractionException.	boolean	Proxy

Method Name	Description	Return Type	Parameters
getNumberOfCursorColumns	Returns the number of columns contained by each record in the cursor. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	integer	Proxy
getCursorColumnNames	Returns a vector of strings containing the names of all columns in the cursor. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	Vector	Proxy
getCursorColumnTypes	Returns a vector of Class objects containing the types of all columns in the cursor. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	Vector	Proxy
getNumberOfCursorRecords	Returns the number of records contained in the cursor. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	integer	Proxy
isLastCursorRecord	Returns the boolean value of True if the current record in the cursor is the last record of the result set returned by the query or if no records were returned by the query. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	boolean	Proxy

Method Name	Description	Return Type	Parameters
nextCursorRecord	Advances the cursor so that the next record in the result set becomes the current record. Throws NotInInteractionException. Throws Last Record Reached Exception if the current record is also the last record. If no results were returned to the cursor, throws InvalidCursorException.	void	Proxy
getCurrentRecord	Returns a vector of objects containing the contents of the current record in the cursor. Throws NotInInteractionException. If no results were returned to the cursor, throws InvalidCursorException.	Vector	Proxy

Replacing a Panel with a Java Bean

Use this procedure to replace a graphical script panel with a custom Java bean. The use of Java beans to replace panels in a script is only supported for the Scripting Engine agent interface.

Prerequisites

- Create a java bean class that meets the requirements of an Oracle Scripting replaceable java bean.
- Package the class containing the Java bean into a Java archive (JAR) file using methods supported by Oracle Scripting (use the JAR utility from the JDK specifying no compression, or create a ZIP file and rename the file to a JAR file format).
- Using the Scripting Administration console, upload the Java bean to the applications database. For instructions, see Uploading New Java Archive Files in the *Oracle Scripting Implementation Guide*.
- Designate a panel in the custom script to be replaced with the Java bean.

To load a jar file into the database, you need to use the Scripting Administration console. The Jar file tab allows you to specify a logical name for the jar file (this needs to

match the name given in step 2c). Then specify the file location of the Jar file and upload the file. Note that a Jar/script mapping does not have to be created if the Jar file is only to be used for its Java Beans (and not for other Java Commands).

Steps

1. Using the Script Author, open a new or existing graphical script.
2. For a new script, create a panel by doing the following:
   - In the tool palette, click the **Panel Insertion Mode** tool.
   - On the canvas, click where you want to insert the panel.
   - In the tool palette, click the **Toggle Select Mode** tool.
   - Double-click on the appropriate panel.

     The Panel Properties window appears.
   - From the panel properties tree, select Properties.
   - In the Properties pane, type a name for this panel.
   - Optionally, in the Properties pane, type comments and a label, if required.
3. For an existing script, locate the panel you want to replace, and access the panel properties by doing the following:
   - In the tool palette, click the **Toggle Select Mode** tool.
   - Double-click on the appropriate panel.

     The Panel Properties window appears.
   - From the panel properties tree, select **Properties**.
4. In the Properties pane, select the **Replace with a Java Bean** box.

   The Bean Name and Jar File Name fields enable.
5. In the Bean Name field, type the fully qualified class name of the Java Bean to use, including package name.
6. In the Jar File Name field, type the name of the JAR file containing the appropriate Java bean.

   This is the logical name used to associate the Java archive file in the database. If no value was provided in the Name field when uploading the Java bean, this will be identical to the file name.

When uploading a Java archive into the applications database, this value is referenced.

7. In the Panel Properties window, click **OK**.

   The panel properties are saved, and the Panel Properties window closes.

8. Save your work.

Guidelines

Replacing panels in a script with custom Java beans is customization. Custom Java beans must be developed by knowledgeable individuals certified in Java and any other appropriate technologies. Problems with customized script are not supported. Ensure you understand return values, methods, and properties used in custom Java bean components.

3

Predefined Script Author Commands

This chapter covers the following topics:

- Overview of Seeded Script Author Commands
- Support Statement
- Seeded Command Detail

Overview of Seeded Script Author Commands

Reusable commands are stored Script Author commands that allow developers to define parameters to be passed to PL/SQL procedures, Forms commands, Java commands, Scripting blackboard commands, and constant commands.

Using Oracle Scripting several reusable commands are installed in the applications database and are available out of the box for use from the Script Author. These seeded commands provide script developers with a set of tools to integrate with Oracle business applications, and perform a number of functions from a graphical script that integrate with the Trading Community Architecture (TCA) of Oracle Applications, as well as specific schema items in Oracle business applications such as Oracle TeleSales, Oracle TeleService, and Oracle Collections. For example, using the commands currently available, you can create a lead in Oracle TeleSales, register a customer for an event, or create contacts, party IDs or locations in the TCA. Currently, all of the seeded commands available are for business-to-business applications. These commands are all associated with a contact within an organization in the TCA architecture. These commands create, update, or retrieve information with a series of custom APIs that serve as wrappers around the actual business objects exposed by the business applications. These seeded commands are not applicable to business-to-consumer applications.

The table below lists the full set of seeded commands currently available:

Command Function	Command Name	Description
Create Lead	ies_create_opp_for_lead	Creates an opportunity for an existing Oracle Telesales sales lead
Create Lead Opportunity	ies_create_opp_for_lead	Creates an opportunity for an existing Oracle Telesales sales lead
Send Collateral	ies_send_collateral	Creates a collateral request for One to One Fulfillment
Register for an Event	ies_create_event_reg	Registers a customer for an event
Register Interest for Organization	ies_create_interest_org	Registers a product interest at the Organization level
Register Interest for Contact	ies_create_interest_contact	Registers a product interest at the Contact (Person) level
Create Organization Contact	ies_create_org_contact	Creates a new customer in the CRM Trading Community Architecture (TCA) schema
Create Party Org Type	ies_create_party_org_type	Creates a Party organization in the TCA schema
Create Party Site	ies_create_party_site	Creates a Party Site in the TCA schema
Create Person Party	ies_create_person_party	Creates a Person Party in the TCA schema
Create Location	ies_create_location	Creates a location/address in the TCA schema
Update Organization Contact	ies_update_org_contact	Updates a contact in the TCA schema
Update Party Site	update_party_site	Updates a location/address in the TCA schema

Each of the seeded commands listed above contains a number of parameters, as

described in this section. Using Script Author, you must supply values for each of the parameters in order for the commands to be processed correctly in a script. These scripts can be executed either in the agent interface, or as a survey taken over the Web interface of Oracle Scripting.

For example, if you want to create a script that sends a collateral item to a customer, you would need to supply information such as the recipient's e-mail address, collateral ID, e-mail subject heading, and the content of the e-mail message that is sent to the customer with the collateral attachment. The values from these parameters can be derived from questions prompted for within the script (by supplying a list of values of collateral items, which the user selects at runtime), or they can be "hard coded" as constants in the Send Collateral reusable command.

The command parameters have been set up so they reference blackboard key names for many of these commands. This provides you with flexibility in using the commands by letting you either set the value of the blackboard key name to a constant, or requiring you to use each blackboard key name as the name of a node (also referred to as an "answer definition" or "question") which collects the appropriate value from the user at runtime. Thus, to use one of the seeded commands successfully, you must fully understand the set of parameters required for the command to execute, and facilitate the passing of those parameters in the script.

For example, consider how the Send Collateral command might be used: Imagine a scenario where you want to prompt the recipient for his or her e-mail address and the specific piece of collateral desired. However, you want to "hard code" the values to be used for the subject heading and e-mail message content. You can create a script which uses the pre-defined blackboard key names as the questions which are displayed at runtime. By collecting the runtime user's responses into these specific blackboard key names, you can simply invoke the Send Collateral reusable command without having to modify the command once it is pulled into the script.

Support Statement

Use of seeded commands in a script is considered customization, which is not supported. In order to use the seeded commands successfully, you must have a detailed understanding regarding the use of Oracle Scripting. Specifically, you must understand how to create and modify commands for graphical scripts in the Script Author, how to import, edit, and modify reusable commands, how to pass parameters in a command, how to use blackboard, constant, and Java commands, and more. You must understand and be certified in the relevant technologies (Java, PL/SQL, SQL) and be familiar with Oracle Applications administration. If using Java, you must be conversant with the appropriate methods of writing, testing, compiling, and deploying the custom code, how to reference the code from the Script Author, and where to administer Oracle Applications in the appropriate architecture so that your compiled Java is accessible. This may involve some Apache Web server administration or customization of other configuration files.

Seeded reusable commands are therefore documented to provide knowledgeable and

trained developers with the appropriate information to use the commands.

All reusable commands function as expected and as documented at the time of this writing. Oracle warrants that these commands as documented function at the time of publication. Since many of the commands integrate with other Oracle Applications and with the applications database, no guarantee is provided by Oracle that these commands will continue to work in future releases.

Nor is the use of seeded commands supported from a training perspective. If you do not possess the appropriate skill sets as addressed above, Oracle recommends you do not attempt to use the seeded commands. Since use of seeded commands is customization, assistance is not available through Oracle Support Services. These commands are provided as an added service to our customers on an as-is basis. Use these components at your own risk.

Seeded Command Detail

This section provides details for all seeded commands, including each seeded command name, relevant package name, related building block script, and each command parameter, blackboard value, and a brief description of each.

Create Lead

Description

This command creates a lead in Oracle Telesales.

Prerequisites

In order to successfully execute the Create Lead API, you must ensure that the user has the appropriate responsibilities and profile settings that provide the ability to create a lead through Oracle Telesales.

Package Name

IES_TELESALES_BP_PKG.create_sales_leads

Associated Building Block Script Name

ieslead.scr (Create Lead building block script)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.

Parameter	Blackboard Key Name	Description
p_budget_amount	P_BUDGET_AMOUNT	The amount of money that is being budgeted. This parameter is not required and can be set to a null value. If using the Create Lead building block script, the P_BUDGET_AMOUNT value is set by an answer in a panel within the Gather_Required_ Lead_Data group.
p_budget_status_code	P_BUDGET_STATUS_CODE	The status of the budget. This must be a valid BUDGET_STATUS in the AMS_LOOKUPS table. If using the Create Lead building block script, the P_ BUDGET_AMOUNT value is set by an answer in a panel within the Gather_Required_Lead_Data group.
p_channel_code	Constant	A channel code associated with the lead. This parameter is assigned a value in the reusable command and should not be changed.
p_contact_id	CONTACT_ID	The CONTACT_ID which identifies the person within the organization for whom the lead is being created. This must be a valid contact, and it is associated with the Party_ID. If the script using the reusable command has been called by OracleTelesales or Collections, this value is passed into the script with a Blackboard Name of CONTACT_ID. If the script has been called by Oracle Teleservice, the value of the CONTACT_ID can be retrieved by using the Forms command GetContactId and assigning its value as a constant for this parameter. If you are using the Create Lead building block script, the script checks to see if a CONTACT_ID has been passed by the calling application. If the value is null, it looks up the CONTACT_ID based on the PARTY_ID passed into the script. Otherwise, it provides a query allowing you to select the Contact person. Though it references the blackboard key CONTACT_ID, it is passing Null since CONTACT_ ID is never set.
p_currency_code	Constant - USD	A currency code. This value is pre-set and does not need to be changed.

Parameter	Blackboard Key Name	Description
p_customer_id	PARTY_ID	The Party_ID which identifies the organization with which this lead is associated. This value is required, and it must reference a valid organization. When a script is invoked by Oracle Telesales or Collections, this value is supplied as the PARTY_ID Blackboard Name. If the script has been called by Oracle Teleservice, the value of the PARTY_ID can be retrieved by using the Forms command GetCustomerId and assigning its value as a constant for this parameter.
p_decision_time frame_ code	P_DECISION_TI MEFRAME_ CODE	The time frame in which the customer/prospect expects to purchase the product for which the lead is being created. This value must be a valid DECISION_TIMEFRAME in the AS_LOOKUPS table. If using the Create Lead building block script, the P_DECISION_TIMEFRAME_CODE value is set by a an answer in a panel within the Gather_ Required_Lead_Data group.
p_description	P_DESCRIPTION	A description of the project for which the lead is being created. The value for this is not validated and may be null. If using the Create Lead building block script, the P_DESCRIPTION value is set by an answer in a panel within the Gather_Required_ Lead_Data group.
p_interest_type_ id	P_INTEREST_TY PE_ID	Identifies the Interest Type for which the lead is being created. This is a required value, and it must be a valid INTEREST_TYPE_ID from the AS_ INTEREST_TYPES_V table. If using the Create Lead building block script, the P_INTEREST_TYPE_ID is set by a query that allows the user to select an interest type from a drop down menu in a panel in the Gather_Required_Lead_Data group.
p_primary_inter est_ code_id	P_PRIMARY_IN TEREST_ CODE_ID	Identifies the specific item, or interest code, for which the lead is being created. This is a required value, and it must be a valid INTEREST_CODE_ID from the AS_INTEREST_CODES_V table. If using the Create Lead building block script, the P_PRIMARY_INTEREST_CODE_ID is set by a query that allows the user to select an interest code from a drop down menu in a panel in the Gather_ Required_Lead_Data group.

Parameter	Blackboard Key Name	Description
p_project_name	P_PROJECT_NAME	Identifies the project name the customer uses to describe the initiative. This value is not validated and can be null. If using the Create Lead building block script, the P_PROJECT_NAME is set by an answer in a panel in the Gather_Required_Lead_ Data group.
p_secondary_interest_ code_id	P_SECONDARY_ INTEREST_ CODE_ID	Not required and may be null. Identifies the Interest Type for which the lead is being created - and this is referenced a secondary product interest. If this parameter is set to a non-null value, it must be a valid INTEREST_TYPE_ID from the AS_INTEREST_TYPES_V table. If using the Create Lead building block script, the P_SECONDARY_ INTEREST_CODE_ID is set by a query that allows the user to select an interest type from a drop down menu in a panel in the Gather_Required_Lead_Data group.
p_source_promotion_id	P_SOURCE_PROMOTION_ ID	The source code for this lead. This value must be a valid SOURCE_CODE_ID in the AMS_SOURCE_ CODES table. If using the Create Lead building block script, the P_SOURCE_PROMOTION_ID value is set by a an answer in a panel within the Gather_Required_Lead_Data group.
p_status_code	Constant	Constant; leave the value already defined by the script.
p_user_name	IES_FND_USER_ NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_return_status	x_return_status	S - Success E - Error U - Unexpected error
x_sales_lead_id	x_sales_lead_id	The sales lead ID returned for a successful execution of the API.

Create Lead Opportunity

Description

This command creates an opportunity for an Oracle Telesales lead.

Prerequisites

An Oracle Telesales lead must already exist in order to create an opportunity for a lead.

Package Name

IES_TELESALES_BP_PKG.create_opp_for_lead

Associated Building Block Script Name

ieslead.scr (Create Lead building block script)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_sales_lead_id	N/A	N/A
p_budget_status_code	P_BUDGET_STATUS_CODE	The status of the budget. This must be a valid BUDGET_STATUS in the AMS_LOOKUPS table. If using the Create Lead building block script, the P_ BUDGET_AMOUNT value is set by an answer in a panel within the Gather_Required_Lead_Data group.

Parameter	Blackboard Key Name	Description
p_channel_code	Constant	A channel code associated with the lead. This parameter is assigned a value in the reusable command and should not be changed.
p_contact_id	CONTACT_ID	The CONTACT_ID which identifies the person within the organization for whom the lead is being created. This must be a valid contact, and it is associated with the Party_ID. If the script using the reusable command has been called by OracleTelesales or Collections, this value is passed into the script with a Blackboard Name of CONTACT_ID. If the script has been called by Oracle Teleservice, the value of the CONTACT_ID can be retrieved by using the Forms command GetContactId and assigning its value as a constant for this parameter. If you are using the Create Lead building block script, the script checks to see if a CONTACT_ID has been passed by the calling application. If the value is null, it looks up the CONTACT_ID based on the PARTY_ID passed into the script. Otherwise, it provides a query allowing you to select the Contact person. Though it references the blackboard key CONTACT_ID, it is passing Null since CONTACT_ ID is never set.
p_currency_code	Constant - USD	A currency code. This value is pre-set and does not need to be changed.
p_customer_id	PARTY_ID	The Party_ID which identifies the organization with which this lead is associated. This value is required, and it must reference a valid organization. When a script is invoked by Oracle Telesales or Collections, this value is supplied as the PARTY_ID Blackboard Name. If the script has been called by Oracle Teleservice, the value of the PARTY_ID can be retrieved by using the Forms command GetCustomerId and assigning its value as a constant for this parameter.

Parameter	Blackboard Key Name	Description
p_decision_timefra me_ code	P_DECISION_TIMEFR AME_ CODE	The time frame in which the customer/prospect expects to purchase the product for which the lead is being created. This value must be a valid DECISION_TIMEFRAME in the AS_LOOKUPS table. If using the Create Lead building block script, the P_DECISION_TIMEFRAME_CODE value is set by a an answer in a panel within the Gather_ Required_Lead_Data group.
p_description	P_DESCRIPTION	A description of the project for which the lead is being created. The value for this is not validated and may be null. If using the Create Lead building block script, the P_DESCRIPTION value is set by an answer in a panel within the Gather_Required_ Lead_Data group.
p_interest_type_id	P_INTEREST_TYPE_I D	Identifies the Interest Type for which the lead is being created. This is a required value, and it must be a valid INTEREST_TYPE_ID from the AS_ INTEREST_TYPES_V table. If using the Create Lead building block script, the P_INTEREST_TYPE_ID is set by a query that allows the user to select an interest type from a drop down menu in a panel in the Gather_Required_Lead_Data group.
p_primary_interest _ code_id	P_PRIMARY_INTERE ST_ CODE_ID	Identifies the specific item, or interest code, for which the lead is being created. This is a required value, and it must be a valid INTEREST_CODE_ID from the AS_INTEREST_CODES_V table. If using the Create Lead building block script, the P_PRIMARY_INTEREST_CODE_ID is set by a query that allows the user to select an interest code from a drop down menu in a panel in the Gather_ Required_Lead_Data group.
p_project_name	P_PROJECT_NAME	Identifies the project name the customer uses to describe the initiative. This value is not validated and can be null. If using the Create Lead building block script, the P_PROJECT_NAME is set by an answer in a panel in the Gather_Required_Lead_ Data group.

Parameter	Blackboard Key Name	Description
p_secondary_interest_ code_id	P_SECONDARY_INTEREST_ CODE_ID	Not required and may be null. Identifies the Interest Type for which the lead is being created - and this is referenced a secondary product interest. If this parameter is set to a non-null value, it must be a valid INTEREST_TYPE_ID from the AS_INTEREST_TYPES_V table. If using the Create Lead building block script, the P_SECONDARY_ INTEREST_CODE_ID is set by a query that allows the user to select an interest type from a drop down menu in a panel in the Gather_Required_Lead_Data group.
p_source_promotion_id	P_SOURCE_PROMOTION_ ID	The source code for this lead. This value must be a valid SOURCE_CODE_ID in the AMS_SOURCE_ CODES table. If using the Create Lead building block script, the P_SOURCE_PROMOTION_ID value is set by a an answer in a panel within the Gather_Required_Lead_Data group.
p_status_code	Constant	Constant; leave the value already defined by the script.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.

Parameter	Blackboard Key Name	Description
x_return_status	x_return_status	S - Success E - Error U - Unexpected error
x_sales_lead_id	x_sales_lead_id	The sales lead ID returned for a successful execution of the API.

Send Collateral

Description

This command creates a Fulfillment Request for Oracle One-to-One Fulfillment, identifying the collateral that should be sent, the e-mail address to which it must be sent, and the subject of the message. Upon executing this API, One-to-One Fulfillment returns a Fulfillment Request ID, which can be used to track the status of the fulfillment request. One-to-One Fulfillment is actually responsible for processing the collateral request.

Prerequisites

None

Package Name

IES_TELESALES_BP_PKG.SUBMIT_COLLATERAL_TO_FM

Associated Building Block Script Name

iescltrl.scr (Create Collateral building block)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.

Parameter	Blackboard Key Name	Description
p_deliverable_id	P_DELIVERABLE_ID	P_DELIVERABLE_ID Provides a valid collateral identifier, which must be a valid DELIVERABLE_ID from the AMS_ DELIVERABLES_VL table. If you are using the Create Collateral building block script, the value for this blackboard key name is obtained by allowing the user to select from a drop down menu that displays a list of AMS_DELIVERABLE_NAME values. When the user selects the Deliverable based on its name, the script provides the P_ DELIVERABLE_ID.
p_email	P_EMAIL	Provides the e-mail address for the recipient. There must be a valid e-mail address in this field. If you are using the Create Collateral building block script, the P_EMAIL blackboard key name is assigned a default value of the e-mail address stored in HZ_ CONTACT_POINT.EMAIL_ADDRESS. If no e-mail address is stored in this table, then the building block displays a blank value.
p_party_id	CONTACT_ID	The CONTACT_ID which identifies the person within the organization for whom the lead is being created. This must be a valid contact, and it is associated with the Party_ID. If the script using the reusable command has been called by OracleTelesales or Collections, this value is passed into the script with a Blackboard Name of CONTACT_ID. If the script has been called by Oracle Teleservice, the value of the CONTACT_ID can be retrieved by using the Forms command GetContactId and assigning its value as a constant for this parameter. If you are using the Create Lead building block script, the script checks to see if a CONTACT_ID has been passed by the calling application. If the value is null, it looks up the CONTACT_ID based on the PARTY_ID passed into the script. Otherwise, it provides a query allowing you to select the Contact person.
p_subject	P_SUBJECT	Subject line of the e-mail message sent with the collateral attachment. This is a required field. If the Create Collateral building block script is used, this value is prompted for within the script.

Parameter	Blackboard Key Name	Description
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).
p_user_note	P_USER_NOTE	The text of a note that is appended within the e-mail message sent with the collateral attachment. This is not required and the value may be null. If the Create Collateral building block script is used, this value is prompted for within the script.

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_request_id	x_request_id	Fulfillment request ID that is returned from One to One Fulfillment. You can use this request ID to monitor the status of the request from the Fulfillment Admin Console
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Register for an Event

Description

Registers a customer or prospect for an Oracle Marketing Event. When the API is successfully executed, it returns the Event Confirmation number and an Event ID.

Prerequisites

None

Package Name

IES_TELESALES_BP_PKG.register_for_event

Associated Building Block Script Name

iesevent.scr (Event Registration building block)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_application_id	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_attendant_contact_id	PARTY_CONTACT_ID	Required field. The ORG CONTACT ID off the contact in the HZ_CONTACT_ORG_CONTACTS table. When using the Event Registration building block, this value is automatically retrieved.
p_attendant_party_id	CONTACT_ID	Required field. The CONTACT_ID which identifies the person within the organization for whom the lead is being created. This must be a valid contact, and it is associated with the Party_ID. If the script using the reusable command has been called by OracleTelesales or Collections, this value is passed into the script with a Blackboard Name of CONTACT_ID. If the script has been called by Oracle Teleservice, the value of the CONTACT_ID can be retrieved by using the Forms command GetContactId and assigning its value as a constant for this parameter. If you are using the Create Lead building block script, the script checks to see if a CONTACT_ID has been passed by the calling application. If the value is null, it looks up the CONTACT_ID based on the PARTY_ID passed into the script. Otherwise, it provides a query allowing you to select the Contact person.

Parameter	Blackboard Key Name	Description
p_event_offer_id	P_EVENT_OFFER_ID	Required field. A valid EVENT_OFFER_ID from the AMS_EVENT_OFFERS_VL table. If using the Event Registration Building block, the value for P_ EVENT_OFFER_ID is retrieved from a query which displays a drop down menu of all EVENT_OFFER_ NAMES. When the user select the event, the P_ EVENT_OFFER_ID is set.
p_registrant_contact_id	PARTY_CONTACT_ID	Required field. It is set in the manner described for the PARTY_CONTACT_ID blackboard key name above.
p_registrant_party_id	p_registrant_party_id CONTACT_ID	Required field. It is set in the manner described for the CONTACT_ID blackboard key name above.
p_source_code	P_SOURCE_CODE	Required field. A valid source code from the AMS_ SOURCE_CODES table. If using the Event Registration building block, the value for P_ SOURCE_CODE is retrieved by a query of all SOURCE_CODES in the AMS_SOURCE_CODES table, and the user is able to select the value from the table.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Parameter	Blackboard Key Name	Description
x_system_status_code	x_system_status_code	A system status code.

Register Interest for Organization

Description

Creates an interest at an Organization (party) level. Can provide up to two interest codes.

Prerequisites

None

Package Name

None

Associated Building Block Script Name

iesinto.scr (Register Interest for Organization building block)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_interest_type_id	P_INTEREST_TYPE_ID	Required field. A valid INTEREST_TYPE_ID from the AS_INTEREST_TYPES_V table. If using the Register Interest for Organization building block script, the value for P_INTEREST_TYPE_ID is provided by a query of this table. The user selects a INTEREST_TYPE, and the script provides the associated INTEREST_TYPE_ID as P_INTEREST_TYPE_ID.

Parameter	Blackboard Key Name	Description
p_party_id	PARTY_ID	Required field. This is the PARTY_ID corresponding to the Organization from which the contact was selected. This value is required, and it must reference a valid organization. When a script is invoked by Oracle Telesales or Collections, this value is supplied as the PARTY_ID Blackboard Name. If the script has been called by Oracle Teleservice, the value of the PARTY_ID can be retrieved by using the Forms command GetCustomerId and assigning its value as a constant for this parameter.
p_party_site_id	P_PARTY_SITE_ID	Required field. The party site associated with the contact. The P_PARTY_SITE_ID is automatically populated by the Register Interest for Organization building block script.
p_party_type	Constant	The type of party for which the interest is being created. This value is pre-filled and should not be changed.
p_primary_interest_ code_id	P_PRIMARY_INTEREST_ CODE_ID	Required field. A valid INTEREST_CODE_ID from the AS_INTEREST_TYPES_V table. If using the Register Interest for Organization building block script, the value for P_INTEREST_TYPE_ID is provided by a query of this table. The user selects an INTEREST_CODE, and the script provides the associated P_PRIMARY_INTEREST_CODE_ID as P_PRIMARY_INTEREST_CODE_ID.
p_secondary_interest_ code_id	P_SECONDARY_INTEREST_ CODE_ID	Not required and may be null. If this parameter is set to a non-null value, it must have valid INTEREST_CODE_ID from the AS_INTEREST_ TYPES_V table. If using the Register Interest for Organization building block script, the value for P_ INTEREST_TYPE_ID is provided by a query of this table. The user selects an INTEREST_CODE, and the script provides the associated P_SECONDARY_ INTEREST_CODE_ID as P_SECONDARY_ INTEREST_CODE_ID.

Parameter	Blackboard Key Name	Description
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_interest_id	x_interest_id	A confirmation number when the API executes successfully.
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error
x_system_status_code	x_system_status_code	A system status code.

Register Interest for Contact

Description

Registers an interest for a contact.

Prerequisites

None

Package Name

ies_create_interest_contact

Associated Building Block Script Name

iesintc.scr (Register Interest for Contact building block)

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_interest_type_id	P_INTEREST_TYPE_ID	Required field. A valid INTEREST_TYPE_ID from the AS_INTEREST_TYPES_V table. If using the Register Interest for Organization building block script, the value for P_INTEREST_TYPE_ID is provided by a query of this table. The user selects a INTEREST_TYPE, and the script provides the associated INTEREST_TYPE_ID as P_INTEREST_TYPE_ID.
p_org_contact_id	PARTY_CONTACT_ID	Required field. The ORG CONTACT ID off the contact in the HZ_CONTACT_ORG_CONTACTS table. When using the Event Registration building block, this value is automatically retrieved.
p_party_id	PARTY_ID	Required field. This is the PARTY_ID corresponding to the Organization from which the contact was selected. This value is required, and it must reference a valid organization. When a script is invoked by Oracle Telesales or Collections, this value is supplied as the PARTY_ID Blackboard Name. If the script has been called by Oracle Teleservice, the value of the PARTY_ID can be retrieved by using the Forms command GetCustomerId and assigning its value as a constant for this parameter.
p_party_site_id	P_PARTY_SITE_ID	Required field. The party site associated with the contact. The P_PARTY_SITE_ID is automatically populated by the Register Interest for Organization building block script.

Parameter	Blackboard Key Name	Description
p_party_type	Constant	The type of party for which the interest is being created. This value is pre-filled and should not be changed.
p_primary_interest_ code_id	P_PRIMARY_INTEREST_ CODE_ID	Required field. A valid INTEREST_CODE_ID from the AS_INTEREST_TYPES_V table. If using the Register Interest for Organization building block script, the value for P_INTEREST_TYPE_ID is provided by a query of this table. The user selects an INTEREST_CODE, and the script provides the associated P_PRIMARY_INTEREST_CODE_ID as P_PRIMARY_INTEREST_CODE_ID.
p_secondary_interest_ code_id	P_SECONDARY_INTEREST_ CODE_ID	Not required and may be null. If this parameter is set to a non-null value, it must have valid INTEREST_CODE_ID from the AS_INTEREST_ TYPES_V table. If using the Register Interest for Organization building block script, the value for P_ INTEREST_TYPE_ID is provided by a query of this table. The user selects an INTEREST_CODE, and the script provides the associated P_SECONDARY_ INTEREST_CODE_ID as P_SECONDARY_ INTEREST_CODE_ID.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_interest_id	x_interest_id	A confirmation number when the API executes successfully.

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Create Organization Contact

Description

Creates a contact for an organization.

Prerequisites

None

Package Name

IES_TCA_BP_PKG.create_org_contact

Associated Building Block Script Name

iescustr.scr (Retrieve Existing Contact Within an Organization building block. This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.

Parameter	Blackboard Key Name	Description
p_interest_type_id	P_INTEREST_TYPE_ID	Required field. A valid INTEREST_TYPE_ID from the AS_INTEREST_TYPES_V table. If using the Register Interest for Organization building block script, the value for P_INTEREST_TYPE_ID is provided by a query of this table. The user selects a INTEREST_TYPE, and the script provides the associated INTEREST_TYPE_ID as P_INTEREST_ TYPE_ID.
p_content_source_type	Constant	Required field. If using the building block script iescustr.scr, this is defined as a Constant that is set to an original value of USER_ENTERED.
p_org_party_id	x_party_id	Required field. The Party_ID of the organization to whom this contact is attached.If using the building block script iescustr.scr, this is set to the blackboard value for the blackboard key name of x_party_id. The blackbody key of x_party_id is set as a Post Action in the panel called Collect Org Details, as the return parameter on the reusable command ies_ create_party_org_type.
p_party_relationship_ type	Constant	Required field. If using the building block script iescustr.scr, this is defined as a Constant with a value of CONTACT_OF.
p_party_site_id	x_party_site_id	Required field. The party site associated with the contact.If using the building block script iescustr.scr, this is set to the value of the blackboard key of x_ party_site_id. The blackboard key for x_party_site_ id is set by the Post Action on the group called Create Party Site, which invokes the reusable command ies_create_party_site.

Parameter	Blackboard Key Name	Description
p_person_party_id	x_person_party_id	Required field. If using the building block scripts iescustr.scr, this is set to the Blackboard value for the blackboard key name x_person_party_id. The blackboard key of _person_party_id is set by the panel called Collect Person Info in the Post Action which calls the reusable command ies_create_ person_party.
p_user_name	IES_FND_USER_NAM E	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_org_contact_i d	x_org_contact_id	The Organization Contact ID of the organization created by successful completion of this reusable command.
x_party_rel_id	x_party_rel_id	The Party Relationship ID of the party created by successful completion of this reusable command.
x_rel_party_id	x_rel_party_id	The Relationship Party ID created by successful completion of this reusable command.
x_rel_party_nu mber	x_rel_party_number	The Relationship Party Number created by successful completion of this reusable command.

Parameter	Blackboard Key Name	Description
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Create Party Organization Type

Description

Creates a party organization type.

Prerequisites

None

Package Name

IES_TCA_BP_PKG.create_organization

Associated Building Block Script Name

iescustr.scr (Retrieve Existing Contact Within an Organization building block. This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_content_source_type	Constant	Required field. If using the building block script iescustr.scr, this is defined as a Constant that is set to an original value of USER_ENTERED.
p_org_name	P_ORG_NAME	Required field. The name of the Organization being created.If using the building block script iescustr.scr, this blackboard key value is defined as the answer to a question in the panel called Collect Org Details.

Parameter	Blackboard Key Name	Description
p_party_relationship_ type	Constant	Required field. If using the building block script iescustr.scr, this is defined as a Constant with a value of CONTACT_OF.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_party_id	x_party_id	The Party ID for the record created upon successful completion of this reusable command.
x_party_number	x_party_number	The Party Number for the record created upon successful completion of this reusable command.
x_profile_id	x_profile_id	The Profile ID for the record created upon successful completion of this reusable command.

Create Party Site

Description

Creates a party site.

Prerequisites

None

Package Name

IES_TCA_BP_PKG.create_party_site

Associated Building Block Script Name

iescustr.scr (Retrieve Existing Contact Within an Organization building block. This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_location_id	x_location_id	Required field. If using the building block script iescustr.scr, this is set to the Blackboard value for the blackboard key name of x_location_id. The blackboard key of x_location_id is set as a Post Action in the Group called Create Location, upon successful completion of the reusable command ies_ create_location.
p_party_id	x_party_id	Required field. The Party_ID of the organization to whom this contact is attached. If using the building block script iescustr.scr, this is set to the blackboard value for the blackboard key name of x_party_id. The blackboard key of x_party_id is set as a Post Action in the panel called Collect Org Details, as the return parameter on the reusable command ies_ create_party_org_type.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_party_site_id	x_party_site_id	The Party Site ID for the record created upon successful completion of the reusable command
x_party_site_number	x_party_site_number	The Party Site Number for the record created upon successful completion of this reusable command.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Create Person Party

Description

Creates a person/contact for an organization, setting the name and time, and retrieving the ID for the contact.

Prerequisites

None

Package Name

IES_TCA_BP_PKG.create_person

Associated Building Block Script Name

iescustr.scr (Retrieve Existing Contact Within an Organization building block. This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_content_source_type	Constant	Required field.If using the building block script iescustr.scr, this is defined as a Constant that is set to an original value of USER_ENTERED.
p_first_name	P_FIRST_NAME	Required field. The first name of the individual. If using the building block script iescustr.scr, this blackboard key is defined as the answer to a question in the panel called Collect Person Info.
p_last_name	P_LAST_NAME	Required field. The last name of the individual. If using the building block script iescustr.scr, this blackboard key is defined as the answer to a question in the panel called Collect Person Info.
p_middle_name	P_MIDDLE_NAME	Required field. The middle name of the individual. If using the building block script iescustr.scr, this blackboard key is defined as the answer to a question in the panel called Collect Person Info.
p_title	P_TITLE	Required field. The title (Mr., Mrs., Ms, etc.) of the individual. If using the building block script iescustr.scr, this blackboard key is defined as the answer to a question in the panel called Collect Person Info.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_party_number	x_party_number	The Party Number for the record created upon successful completion of this reusable command.
x_party_site_number	x_party_site_number	The Party Site Number for the record created upon successful completion of this reusable command.
x_profile_id	x_profile_id	The profile ID for the record created upon successful completion of this reusable command.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Create Location

Description

Creates a location for a contact, setting the contact's address information, and retrieving the ID for the location.

Prerequisites

None

Package Name

IES_TCA_BP_PKG.create_location

Associated Building Block Script Name

iescustr.scr (Retrieve Existing Contact Within an Organization building block. This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_address1	P_ADDRESS1	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_address2	P_ADDRESS2	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_address3	P_ADDRESS3	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_address4	P_ADDRESS4	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_city	P_CITY	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_postal_code	P_POSTAL_CODE	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_province	P_PROVINCE	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_state	P_STATE	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.
p_country	P_COUNTRY	Required field. If using the building block script iescustr.scr, this is defined as a blackboard key in the panel called Collect Person Info.

Parameter	Blackboard Key Name	Description
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_location_id	x_location_id	The location ID for the record created upon successful completion of this reusable command.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Update Organization Contact

Description

Updates the contact information for a contact in an organization.

Prerequisites

Contact information for an organization must already exist.

Package Name

IES_TCA_BP_PKG.update_org_contact

Associated Building Block Script Name

iesupdct.scr (Update Contact Information building block). This script retrieves an

existing contact within an organization, then allows the user to update the contact name and address information.

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_org_contact_id	P_ORG_CONTACT_ID	Required field. If using the building block script iesupdct.scr, this is set to the Blackboard value for the blackboard key name of PARTY_CONTACT_ID. The blackboard key of PARTY_CONTACT_ID retrieved in the Group Get_Party_And_Contact_ Information.
p_party_site_id	x_party_site_id	Required field. The party site associated with the contact.If using the building block script iescustr.scr, this is set to the value of the blackboard key of x_ party_site_id. The blackboard key for x_party_site_ id is set by the Post Action on the group called Create Party Site, which invokes the reusable command ies_create_party_site.
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.

Parameter	Blackboard Key Name	Description
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

Update Party Site

Description

Updates the contact information for a contact in an organization.

Prerequisites

A contact in an organization must already exist. Package Name IES_TCA_BP_PKG.update_party_site

Associated Building Block Script Name

iesupdct.scr (Update Existing Contact Within an Organization building block). This script retrieves an existing contact within an organization, then allows the user to update the contact name and address information

Input Parameters

The parameters passed by this seeded command are as follows:

Parameter	Blackboard Key Name	Description
p_api_version	Constant	The version number for the API. This value is pre-set and does not need to be changed.
p_party_site_id	P_PARTY_SITE_ID	Required field. The Party_Site_ID of the organization to whom this contact is attached. If using the building block script iesupdct.scr, this is set to the blackboard value for the blackboard key name of P_PARTY_SITE_ID. The blackboard key of P_PARTY_SITE_ID is set in the group Get_Party_ And_Contact_Information.

Parameter	Blackboard Key Name	Description
p_user_name	IES_FND_USER_NAME	Agent login name for the individual running the script or survey. This is a required field, but it does not need to be changed, as the IES_FND_USER_ NAME is automatically set. If the script is executed via a survey, this is the Guest User login (JTF Guest User, as described in the *Oracle Scripting Implementation Guide*).

Output Parameters

The parameters retrieved are as follows:

Parameter	Blackboard Key Name	Description
x_msg_count	x_msg_count	Tells the number of error messages in the message stack.
x_msg_data	x_msg_data	A concatenated error message.
x_return_status	x_return_status	S - Success E - Error U - Unexpected error

4

Oracle Scripting Building Blocks

This chapter covers the following topics:

- Overview of Building Blocks
- How to Use Building Blocks
- Summary Information on the Building Block Scripts
- The Retrieve Customer Building Block Script

Overview of Building Blocks

A building block is a portion of a Script Author script that contains functionality integrating with other Oracle Applications. Building blocks typically contain one or more API and typically perform a task-based operation such as creating a service request, registering for an event, and so forth.

The first building block listed includes flows to create a customer in the database, including contact, party, and organization information typically required in the TCA schema.

The next building block retrieves an existing customer from the database. The remaining building blocks also retrieve an existing customer from the database as part of their flow. It is necessary to retrieve a customer in order to perform a task-based operation. Thus, understanding of the Retrieve Customer building block component is crucial to using and customizing the remaining building blocks. The details of the retrieve customer building block are therefore detailed below.

Building blocks are not intended to be used as stand-alone scripts, but are expected to be imported into a graphical script and modified as appropriate. These are provided on an as-is basis. Use of building blocks requires customization, which is not supported.

You can set up a script or survey so that it either "hard codes" all of the parameter values for a reusable command, or so that it interactively prompts the end user for the required values at runtime. If you choose to prompt for values for the command parameters in a script or survey (for instance, by allowing a customer or prospect to

choose a specific fulfillment item from a drop-down menu of fulfillment items), you must set up the script so that it performs a query against the appropriate database tables and then displays a list of valid choices to the user at runtime. Likewise, when creating a lead, you need to set the values of a number of parameters, and these values must be valid. In other words, you can't create a lead for a customer or prospect who does not yet exist. Nor can you create a lead for a product that is not stored in the product tables of the applications database.

In order to make it easier for you to perform necessary queries against data in CRM tables, we have supplied several sample "building block" scripts that can be imported into your custom scripts. There is a building block script associated with each of the seeded reusable commands. The goal of each building block script is to provide a sample framework which performs the following:

- Works in conjunction with Oracle Telesales and Collections to retrieve customer information such as PARTY_ID from Oracle Telesales and Collections
- Provides query blocks to retrieve lists of values from the CRM schema for command parameters that must reference valid values in CRM tables
- Provides panels, questions, and answers that use the Blackboard Command values that are used as parameters in the reusable command.

Building Blocks Versus Seeded Reusable Commands

The differences between a building block script and a reusable command is that the building block script is designed as a dialog (script) that interactively guides a user through questions used to gather information for a particular function. A seeded reusable command is simply a command that provides you a parameter list, for which you must supply values. The building blocks and seeded reusable commands work together in the sense that the building block scripts use the Blackboard Key Name values as the questions for the information asked during runtime; once all the questions have been answered, the building block script invokes the appropriate seeded reusable command and all of the necessary parameters have been provided in order for it to execute successfully.

In the section on each reusable command, the associated building block which you can use in conjunction with the command is indicated. Again, please note that you can choose to "hard code" the values for all of the command parameters for each seeded reusable command, or you can use a combination of hard coding and interactively obtaining user input for the required parameters.

General Structure of a Building Block Script

The building block sample scripts are designed to work on their own, so you can open them in the Script Author, deploy them to the Scripting database, and then run them from an account that has the Oracle Telesales Agent responsibility, or by creating

survey campaigns that reference each building block script and deploying the building block scripts as surveys. These scripts can also be imported into any script or survey that needs to incorporate the building block functionality.

The general structure of a building block script is the following:

- An introductory panel which has panel text that describes the purpose of the script and lists the reusable commands used within the script.

- A group that retrieves an existing party (organization) and person (contact) from the customer schema. This group consists of several panels that first check to see if a calling application (such as Oracle Telesales) has already set the value for the party and contact; if not, it will query the user for this information.

  Note: This group is not required if the building block is included as part of another script that already performs this sort of functionality.

- A panel or group that collects the information that a customer/prospect needs to supply in order to invoke the reusable command and perform the business task. For example, the Create Collateral building block script contains a group that looks up the customer's e-mail address, provides a list of available collateral by looking up the information in the One to One Fulfillment table, then allows the user to define the subject and content for the e-mail message.

- Likewise, the Lead Creation building block script provides a series of queries of values in the tables and prompts the user to provide budget status, budget time frame, select a primary product interest, etc.

- A panel or group that performs that actually invokes the reusable command and checks the status of that action. For example, in the Lead Creation building block, this group invokes the Lead Creation API, checks the status, and reports the Lead ID to the user. (If the Lead API fails for any reason, it also displays an error message stating the problem).

Location of Building Block Scripts

The building block sample scripts are placed on the 11i APPL_TOP in the directory $IES_TOP/scripts.

How to Use Building Blocks

1. Open and deploy a building block script.

2. Run the building block in the Scripting Engine agent interface to ensure its

functionality.

3. Open the building block in the Script Author and examine it, familiarizing yourself with aspects such as:
   - From where commands are executed (pre-actions, post-actions, etc.)
   - What existing validation is used, if any, or if validation is required to be added
   - Whether you can see the existing text and graphics in panels, and if existing panel content meets your requirements
4. Incorporate the building block into an otherwise tested and fully functional script.
5. Test the script to ensure that it functions as imported.
6. Modify the building block within the context of the script as required.

Summary Information on the Building Block Scripts

There are eight different building block scripts available. They can each be deployed and run alone, or incorporated within other scripts. Each of these scripts invokes several of the reusable commands: they all either retrieve or create a customer and organization, then perform a specific business task by using the reusable commands associated with that task.

They are:

Building Block Script	Description
iescustr.scr	This script creates a contact (name and address information), a party, a party location, and an organization type in the TCA schema.
iesrcust.scr	This script retrieves an existing contact within an organization. This script is structured in a way that allows it to retrieve all of the information about the contact and organization in order to perform any of the other reusable commands. More detail on this building block is described below.
iescltrl.scr	This script retrieves an existing contact within an organization and then allows the user to select from a list of available collateral. It allows the user to specify the e-mail address, subject line, and additional notes, then invokes the One to One Fulfillment API to create collateral.

Building Block Script	Description
iesevent.scr	This script retrieves an existing contact within an organization, then allows the user to register that contact for an event.
iesintc.scr	This script retrieves an existing contact within an organization, then creates an interest at the contact level.
iesinto.scr	This script retrieves an existing organization, then creates an interest at the organization level.
ieslead.scr	This script retrieves an existing contact within an organization, then creates a lead.
iesupdct.scr	This script retrieves an existing contact within an organization, then allows the user to update the contact name and address information.

The Retrieve Customer Building Block Script

The Retrieve Customer building block script (iesrcust.scr) should be considered for use in any script that requires customer contact information or which requires using any CRM APIs. This building block sets four important Blackboard Key Names:

Blackboard Key	Description
PARTY_ID	The ID for an organization. This is set by Oracle Telesales eBusiness Center form when you query by Party or Party_ Organization. If the value is not set when the script is launched, the Retrieve Customer building block will collect this ID from the user.
CONTACT_ID	The Contact_ID for a contact (person) in an organization. The script prompts the user to select a contact from a list of values based on a query of all contacts for the organization identified by the PARTY_ID.
PARTY_CONTACT_ID	Once a contact has been selected, this value is automatically set based on the selected contact.
P_PARTY_SITE_ID	Once a contact has been selected, this value is automatically set based on the selected contact.

5

Best Practice Surveys

This chapter covers the following topics:

- Best Practice Survey Script Considerations
- Location of Best Practice Survey Scripts
- Description of Best Practice Survey Scripts

Best Practice Survey Script Considerations

Oracle Scripting provides predefined survey questionnaire scripts flows to serve as examples of how to customize graphical scripts specifically intended for use as survey questionnaires. Aspects of these scripts can also be used in graphical scripts to be executed in the agent interface.

Best practice survey scripts can be used as-is. Note, however, that some words in panel text such as PRODUCT or COMPANY are used. You may want to customize these before deploying.

For some survey scripts, certain questions are broken out into multiple panels. The purpose for this approach is to make it easier for you to remove unwanted questions or add additional branching logic for a given question based on your survey campaign's specific requirements.

Additionally, best practice surveys are built to ask the most important questions first. If your survey campaign priorities differ, you may wish to rearrange the script flow to put panels asking the most crucial data first.

How you ask a question is critical to the quality of data received. In order to receive meaningful data, consider the end results. How will the reporting data be used? Are you providing appropriate choices for a survey respondent to allow you to collate data appropriately, and clearly differentiate between a positive, negative, or neutral opinion?

Best practice survey scripts are provided on an as-is basis. Be aware that best practice surveys, once customized, are not supported.

Location of Best Practice Survey Scripts

The best practice survey scripts are placed on the 11i APPL_TOP in the directory $IES_TOP/scripts.

Description of Best Practice Survey Scripts

The table below lists the file name of each best practice survey script, its global script name (the name referenced when deployed to the database), and a description of the purpose for each survey script.

File Name	Global Script Name	Description
iespship.scr	BP Product Shipment Followup	Set of typical survey questions to ask after a product is shipped to determine customer satisfaction. If customer is dissatisfied, gathers customer reasons.
iesesrfb.scr	BP Electronic SR Follow Up Survey	Set of questions to determine customer satisfaction following a service request. If customer is dissatisfied, prompts customer for callback information.
ieswebin.scr	BP Web Content Feedback	Survey that can be placed on a Web site to get customer feedback regarding the Web site
iescwsat.scr	BP Courseware Evaluation	Survey evaluating satisfaction after taking a course or training program
ieshitch.scr	BP HiTech Support Customer Sat	Detailed survey to determine customer satisfaction on customer support for high-tech industries

File Name	Global Script Name	Description
iesevtfb.scr	BP Event Followup	Survey to measure satisfaction with an event, including specific flows to determine effectiveness of delivery channels such as teleconference call, Web seminar, or live contact
iesnewcr.scr	BP New Customer Followup	New customer survey requesting information on a customer's reason for purchasing, competitive advantages to the product purchased, and overall product satisfaction
iescgsat.scr	BP Generic Customer Satisfaction	Generic customer satisfaction survey, not associated with any particular event

6

Customizing Panel Layouts

This chapter covers the following topics:

- Overview of Customizing Panel Layouts
- Purpose and Limitations of the Panel Layout Editor
- HTML Restrictions in the Scripting Engine
- Oracle Scripting Custom HTML Syntax
- Procedures for Customizing Panel Layouts

Overview of Customizing Panel Layouts

The Script Author provides three ways to customize panel layouts for graphical scripts:

1. Using the panel layout editor. This is an integrated component which provides script developers with a convenient method for creating and editing simple formatted panel content without writing HTML code.

2. Using third-party HTML generation and editing tools. For complex or feature-rich panel layouts, HTML content can be created externally and imported into a script. Using Oracle Scripting-specific HTML tags, the entire panel content, including all answer definitions and answer user interface controls, can be generated in custom HTML and imported into the script.

3. Using the panel layout editor in combination with custom HTML editing tools. If desired, answers can be defined from the Answer Entry Dialog window, and content can be created in the panel layout editor, and then exported as HTML, enhanced externally, and imported back into the script. Any previous answer definitions and all panel content and formatting is replaced by the imported panel content.

Knowledge of custom Oracle Scripting HTML syntax for is required.

Purpose and Limitations of the Panel Layout Editor

The Script Author includes a panel layout editor, which provides script developers with a convenient method for creating and editing simple panel content without writing HTML code. This simple graphic interface provides users with the ability to enter and format simple panel content, and manipulate typeface, font size and color, create bulleted and ordered lists, and more. In addition, it serves as the way to produce syntactically correct HTML content for Scripting-specific features (such as embedded values) without having to learn custom Oracle Scripting HTML tags and syntax. Answers and text can be interspersed and formatted as desired. The panel layout editor includes one-click formatting of text into two distinct styles, instructional text and spoken text. This provides script developers a consistent, rapid method to format text differently in the panel layout editor to indicate text an agent would speak (or a customer would read) that relates directly to the matter at hand, and to format text providing the script end user with specific instructions.

Nonetheless, for creating any complex layout, and using features of HTML such as tables, it is expected that script developers will use full-function third-party HTML editors (or code HTML by hand) and simply import the HTML content into the Script Author.

This combination ultimately delivers a solution that combines ease of use for simpler implementations with a high level of flexibility for implementations that have unusual or complex requirements. To support this model, this section includes Scripting-specific HTML tags and syntax.

HTML Restrictions in the Scripting Engine

All panel content, whether created in the panel layout editor or externally, renders as Hypertext Markup Language (HTML) in the Scripting Engine runtime interface. Each interface is subject to its own set of restrictions, as well as restrictions that apply to both interfaces.

Restrictions for Web and Agent Interfaces

All panel content, whether created in the panel layout editor or externally, renders as Hypertext Markup Language (HTML) in the Scripting Engine runtime interface. Each interface is subject to its own set of restrictions, as well as restrictions that apply to both interfaces.

Web Interface Restrictions

When developing scripts to execute in the Scripting Engine Web interface, you should be aware of the following restrictions, described in detail below:

- HTML interpretation differs by browser

- JavaScript and JScript Support
- Agent UI components not included in Web interface

HTML Interpretation Differs by Browser

In the Web interface, script HTML (executed as a survey questionnaire) is interpreted by the HTML engines afforded by the particular Web browser used. Thus, survey questionnaires may appear differently when rendered in Microsoft Internet Explorer than they do in Netscape Navigator. Differences may also appear between various releases of the same browser software.

JavaScript and JScript Support

JavaScript is a scripting language intended to execute within any JavaScript-compliant Web browser. All current Web browsers certified for use with Oracle Applications are JavaScript-compliant.

> **Note:** Microsoft Corporation has created a competing technology to JavaScript known as JScript. Due to browser-specific differences in internal script interpretation, scripts that execute perfectly in one Web browser may not function the same way in a different Web browser (including a different version of the same browser software). JavaScript and JScript developers must be certified in the appropriate relevant technologies and should take browser incompatibilities into account. Further, if scripts are intended for execution in the Web interface, developers must take care to ensure any JavaScripts or JScripts used will support both Netscape Navigator and Microsoft Internet Explorer.

- For the purposes of this document, scripts created in JavaScript or JScript are referred to as JavaScripts.
- Despite the name, JavaScript and the Java programming language are separate and distinct. JavaScripts can be included in custom panel text for a panel object in the Script Author, if that script will be executed in a Web browser.

> **Caution:** JavaScript is not supported for scripts executed in the agent interface and should not be used.

There are two approaches to including JavaScripts into panel HTML.

1. Create a complete and syntactically correct panel in the Script Author, and then export the panel HTML. Edit the HTML to include the JavaScript as appropriate.
2. Identify all requirements for the panel which requires JavaScript, and using a compliant HTML editor or text editor, write the HTML and JavaScript code for the

panel. Then, from the Script Author, import the code into the appropriate panel.

For more information, see Exporting Panel Text, Adding JavaScript, and Importing, page 6-14.

Agent UI Components Not Included in Web Interface

When scripts execute in the agent interface, the panels appear in a large work area. Above the panels, if programmed into the script's global properties, the script information area appears, and below this (if defined), shortcut buttons are included. A progress panel shows the active script's history to the right of the agent work area. All of these components are surrounded by a frame. At the bottom, right-hand side of the frame, a Disconnect button appears.

For scripts executed in the Web interface, only panels are rendered in the browser. Regardless of whether a script includes defined shortcuts, or timers or text programmed to appear in the script information area, these will not appear in scripts executing in a browser, and will not be accessible at runtime. The progress area will also not appear; to back up in a script executed in the Web interface, use the browser's Back button. Finally, the frame enclosing the script in the Web interface is the actual Web browser window. Thus, there is also no Disconnect button. The only way a wrap-up group can be accessed is by proceeding through the appropriate flow or flows in the script.

Generally, scripts created for one runtime interface will execute in the other. Nonetheless, the differences noted above must be considered to ensure appropriate flow in scripts executed in the Web interface. Scripts that meet an enterprise's requirements in the agent interface will also be lacking script information area headers, shortcut buttons, the progress area and the disconnect button, and may therefore not meet the same set of requirements when executed in the Web interface. This fact should be noted by sales, support, technical and functional consultants, and those interaction center staff members responsible for ensuring project requirements are fulfilled.

Agent Interface Restrictions

Swing and HTML Incompatibilities

In the Java-based agent interface, script HTML interpretation and rendering is provided by the Swing Java classes compatible with JavaSoft's Java Development Kit (JDK) 1.1.8.

While this entire area of Swing has been rewritten and is much improved in JDK 1.3, Scripting is unable to upgrade its version of Swing until all Oracle Applications upgrades to JDK 1.3 as well. Consequently, scripts executed in the agent interface inherit the HTML incompatibilities created by using the older version of swing. The following are known issues with this version of Swing for HTML rendering:

- Horizontal alignment of containers. While Swing does support alignment of individual elements, it cannot handle alignment of tables and logical containers (as specified by the ... tag).

- Table width attribute. The "width=<pixel value or percentage>" attribute of the <table>...</table> tag is ignored by Swing.

- Avoid use of the
 (break) tag. Swing has known issues with HTML that uses the
 tag. One option is to use paragraph tags instead. For example, use:

  ```
  <p> </p>
  ```

- Use well-formed HTML only. Swing is known to have problems with HTML that is not well-formed; that is, all start tags should have corresponding end tags (e.g. should always have a corresponding).

HTML 3.2 Compatibility

The HTML specification used for this version is HTML 3.2. Tags from later HTML specifications, including HTML 4.0, may not appear as expected when executed in the agent interface. Consequently, rigorous testing should be performed for all custom scripts including custom HTML code.

JavaScript and JScript

JavaScript and JScript are not supported for scripts executed in the agent interface and should not be used.

Oracle Scripting Custom HTML Syntax

When defining an answer in the Script Author, the following answer user interface types are supported: Text fields, text areas, radio buttons, check boxes, buttons, drop-down lists, password fields, check box groups and multi-select list boxes. Each of these answer UI control types has its own corresponding properties (discussed in detail in the Understanding Answers section of *Oracle Scripting User Guide*). Additionally, several of these answer UI control types have unique HTML tags, attributes, and syntax. Custom syntax for these answer UI types are summarized in the table below.

Answer UI Type	HTML Tag Set	Control Definition Syntax
radio button	<rbgroup>...</rbgroup>	<input name="answer definition name" type="radio">
check box group	<cbgroup>...</cbgroup>	<input name="answer definition name" type="checkbox">
button group	<pbgroup>...</pbgroup>	<input value="pbLabel" name="answer definition name" type="submit">

Individuals editing or creating panel text external to the script author must be familiar with and follow the appropriate syntax for any panel in order for custom HTML to

function within a script. Additionally, when editing or creating radio buttons, buttons, or checkbox groups, answer UI-specific custom syntax must be included.

Panel-Level HTML Requirements and Syntax

Panel HTML Requirements

In panel HTML created by the panel layout editor, an HTML table is automatically generated in the body of the HTML document. This table starts just above the first node (also known as a question or an answer definition).

In the HTML tags defining the table, the table has a name property, which must contain the value "IES_ControlManifest". The tag, when generated by the panel layout editor, appears as follows:

Answer UI-Level HTML Requirements and Syntax

For answer UI types that take lookup values, the number of buttons to be displayed and their labels are determined at runtime. Thus, despite the introduction of the What-You-See-Is-What-You-Get (WYSIWYG) paradigm, there is no way to have a WYSIWYG display of button, radio button, drop-down or multi-selection list, or checkbox group controls in the panel layout editor. When these answer UI types are viewed in the panel layout editor, a single control of the appropriate type is displayed, with no values. Instead of the lookup value, generic labels appear next to the appropriate control: rbLabel for radio buttons, cbLabel for check box groups, pbLabel for buttons. Labels (as populated in the Label for reporting field of the Answer Entry Dialog window) will render in the panel layout editor as well as at runtime.

In the panel HTML, a single set of custom tags defines the answer control. There is specific syntax associated with the button, radio button, and checkbox group answer UI controls. This syntax, which is built automatically with the panel layout editor, must be maintained in order for the panel to function at runtime.

Using the appropriate stands, answer UI controls can also be defined externally using a text editor or HTML generation and editing tool.

> **Note:** Code samples for HTML may exclude spaces or special characters. For example, the ASCII symbol for a space ( ) appears in panel layout editor-generated HTML but is generally excluded in this document.

Radio Button Answer UI Panel Syntax

This section describes syntax for the custom Oracle Scripting radio button group HTML tag set:

```
<rbgroup> ... </rbgroup>
```

Within the open and close radio button group tag set, there are two required tags, which can appear in any order, and can be surrounded by any valid HTML content.

One required tag is the radio button definition:

```
<input name="answer definition name" type="radio">
```

The name attribute includes the answer definition name, surrounded by quotes. This attribute associates the HTML control with its panel node or answer definition.

The other required tag is the radio button label definition tag set. The tags include:

```
<rblabel>rbLabel</rblabel>
```

Since the actual labels of the individual radio buttons will be determined at runtime, this tag set, including the single rbLabel, appears once in panel HTML (when reviewed before runtime), regardless of how many lookups you define. This serves as a placeholder into which the Scripting Engine at runtime will dynamically insert code for each defined lookup, one per radio button value. For example, consider a radio button group with three lookup values defined, one for each available flavor of an ice cream sundae. Each lookup value includes a display value and the actual value passed by the application (in this example, chocolate/c, vanilla/v and strawberry/s).

The panel HTML for this example appears as follows:

```
<rbgroup>
<input name="flavors" type="radio">
<rblabel>LookupValue1</rblabel>
</rbgroup>
```

At runtime, all custom tags above, starting with the rbgroup tag set, are replaced dynamically. For each lookup value, an instance of the input tag is generated, coded with the value of that lookup. Thus, continuing the above example, the HTML code that actually displays might appear as follows:

```
<p><input name="flavors" type="radio" value="c"><b>chocolate</b></p>
<p><input name="flavors" type="radio" value="v"><b>vanilla</b></p>
<p><input name="flavors" type="radio" value="s"><b>strawberry</b></p>
```

Check box Group Answer UI Panel Syntax

This section describes syntax for the custom Oracle Scripting checkbox group HTML tag set:

```
<cbgroup> ... </cbgroup>
```

Within the open and close checkbox group tag set, there are two required tags, which can appear in any order, and can be surrounded by any valid HTML content.

One required tag is the checkbox group definition:

```
<input name="answer definition name" type="checkbox">
```

The name attribute includes the answer definition name, surrounded by quotes. This attribute associates the HTML control with its panel node or answer definition.

The other required tag is the checkbox label definition tag set. The tags include:

```
<cblabel>cbLabel</cblabel>
```

Since the actual labels of the individual checkboxes in this checkbox group will be determined at runtime, this tag set, including the single cbLabel, appears once in panel HTML (when viewed before runtime) regardless of how many lookups you define. This serves as a placeholder into which the Scripting Engine at runtime will dynamically insert code for each defined lookup, one per checkbox value. For example, consider a checkbox group with three lookup values defined, one for each topping on an ice cream sundae. Each lookup value includes a display value and the actual value passed by the application (in this example, sprinkles/s, chocolate syrup/cs, and chopped nuts/cn).

The panel HTML for this example appears as follows:

```
<cbgroup>
<input name="topics" type="checkbox">
<cblabel>LookupValue1</cblabel>
</cbgroup>
```

At runtime, all custom tags above, starting with the cbgroup tag set, are replaced dynamically. For each lookup value, an instance of the input tag is generated, coded with the value of that lookup. Thus, continuing the above example, the HTML code that actually displays might appear as follows:

```
<p><input name="topics" type="checkbox" value="s"><b>sprinkles</b></p>

<p><input name="topics" type="checkbox" value="cs"><b>chocolate
syrup</b></p>

<p><input name="topics" type="checkbox" value="cn"><b>chopped
nuts</b></p>
```

Button Answer UI Panel Syntax

This section describes syntax for the custom Oracle Scripting button group HTML tag set:

```
<pbgroup> ... </pbgroup>
```

The button answer UI type can only be used when it is the single answer control defined for a panel. The most typical use of the button answer UI type is to provide a single value (such as "Continue") to advance the flow of the script. This answer UI type also supports multiple lookup values for the single answer definition (thus the designation of this tag set as a group). The result at runtime is that, for each lookup defined, the display value appears as the text label of a button at the bottom of the active panel. In panel layout editor-generated HTML, the series of buttons appear horizontally, left-aligned, in succession. If necessary, second or subsequent rows of buttons are right-justified.

Within the open and close button group tag set, there is a single required tag, the button definition, which can be surrounded by any valid HTML content:

```
<input value="pbLabel" name="answer definition name" type="submit">
```

The value attribute identifies this HTML string as a button. Unlike the other custom Oracle Scripting tags, there is no separate label tag.

The name attribute includes the answer definition name. This attribute associates the

HTML control with its panel node or answer definition.

The type attribute identifies this control as a submit button.

The label tag is not supported for the button answer UI type. Therefore, any value entered into the answer definition in the Label for reporting field is ignored.

For example, consider a button group with three lookup values defined, one for each container type for an ice cream sundae. Each lookup value includes a display value and the actual value passed by the application (in this example, small cup/sc, large cup/lc and banana split/bs). However, only the display value for each button (derived at runtime from lookup values defined) is included in the button's HTML content at runtime. Therefore, display values for multiple lookup choices for any one button answer definition must be unique.

The panel HTML for this example appears as follows:

```
<pbgroup>
<input value="pbLabel" name="containers" type="submit">
</pbgroup>
```

At runtime, all custom tags above, starting with the rbgroup tag set, are replaced dynamically. For each lookup value, an instance of the input tag is generated, coded with the value of that lookup. Thus, continuing the above example, the HTML code that actually displays might appear as follows:

```
<p><input value="small cup" name="containers"
type="submit">  </p>

<p><input value="large cup" name="containers"
type="submit">  </p>

<p><input value="banana split" name="containers"
type="submit">  </p>
```

Continue Button Panel Syntax

For panels containing answer UI types other than button, HTML code is required at the bottom of the panel to display a submit button at runtime. The function of this button is to progress the flow of the script from the current panel to the next object in the flow (dynamically determined at runtime). This is referred to as the Continue button.

Panels containing the button answer UI type use that control to progress the flow of the script. As described in Section 6.3.2.3, "Button Answer UI Panel Syntax", that answer control has its own distinct syntax requirements and rules.

When using the panel layout editor, HTML code is automatically generated for the Continue button. If you are building panel content external to the Script Author, this tag is required of all panels not containing the button answer UI type.

The attributes for this required tag can appear in any order, and can be surrounded by any valid HTML content. The syntax for this control is:

```
<input value="Continue" name="IES_CONTINUE" type="submit">
```

The value attribute is automatically provided by the panel layout editor, and is recommended for consistency. The contents of this attribute control the button label at

runtime. This attribute alone may be modified for this required tag.

Unlike the custom button answer UI type, the Continue button tag supports only a single button.

Embedded Value Syntax in Panel Text

When embedded values are added to a panel layout, the Script Author stores values toward the end of the HTML panel text in an embedded value string.

At the time of this writing, these values are stored after the close HTML tag. In order to fix incompatibilities with some HTML editors, the location in panel HTML where this string is stored will likely change in the future. Nonetheless, the presence of the iesembeddedvalue string, and its attributes, are required in order for the commands contained in embedded values to function.

The structure for the embedded value string is as follows:

```
<!--iesembeddedvalue iesuid="[unique string]"
iescommandtype="[embedded value command type]"
iescommand="[command name]" iescommanditerate="[true or false]"-->
```

The iesuid value is a unique identifier string.

The iescommandtype value identifies the command type (e.g., "blackboard" for a blackboard command).

The iescommand value is the name of the Script Author command.

The iescommanditerate value is a literal string of true or false, indicating whether the parameter must iterate.

Procedures for Customizing Panel Layouts

This section describes how to customize the content in a panel.

Requirements for Answer Lookup Values

Of the nine supported answer user interface control types, four do not require lookup values to be entered. These include the three text entry UI types (text field, text area, and password field), and the single checkbox. Text entry answer UI types take keyboard inputs or null values at runtime. The single checkbox control, which takes a click or null value, displays its checkbox label from the Label for reporting field.

All other answer UI types require lookups. These can be explicitly specified (hard-coded) lookup values, or table, cursor, or command lookups.

For all answer UI types except button, a "Continue" submit button will automatically be generated at the bottom of the panel, regardless of whether the panel contains a single answer definition or many.

For the button answer UI type, you must define a lookup value.

Caution: Although the Script Author application does not prohibit you from leaving the value and display value fields empty for a button (resulting in a narrow button with no label at runtime), this course of action is not recommended by Oracle and is not supported.

Requirements for Panel Content

All panel content, regardless of whether it is generated from the Script Author only, from custom HTML only, or a combination, requires at minimum:

- One or more answer definitions.
- The default for distinct branching attribute selected for one answer definition, unless the panel contains a multiple-answer UI type such as checkbox group or multi-select list box.
- HTML code for a submit button, which allows the script to continue to the next object in the flow at runtime.

Answer Definitions

- Each panel in a script requires one or more answers to be defined.
- If the answer user interface type is a button, there can only be a single answer for that panel.

Default for Distinct Branching

The default for distinct branching attribute must be selected for one answer definition per panel.

The only exception to this rule is a panel containing an answer definition of UI type check box group or multi-select list box. Only one answer can direct the flow of the script, whereas these answer UI types support multiple potential answers to the same question/answer definition. Thus, panels containing one or more of these answer UI types are absolved from this requirement.

- Regardless of whether a distinct branch is used, each panel requires at least one answer definition to be set as the trigger for distinct branching. This trigger is set by checking the Default for distinct branching option in an answer definition's Answer Entry Dialog window.
- The Default for distinct branching option must be selected for panels with a button answer UI.
- If you have a panel with only one answer defined, it must be set as the trigger for distinct branching (unless it is a check box group or multi-select list box).

- Regardless of how many answers you define in a panel, only one can be set as the default to trigger distinct branching. Thus, the last answer definition or node for which this attribute was selected is the default for the panel.

Submit Button Code

An HTML submit button on every panel allows the script to continue to the next object in the flow at runtime. This is either:

- An explicitly programmed button answer definition following the custom Oracle Scripting button syntax, or
- A Continue button, generated automatically by the panel layout editor or explicitly in custom panel HTML, following the custom Oracle Scripting Continue button syntax.

Writing Panel Text Outside of the Script Author

You can write syntactically correct HTML for a panel outside of Scripting and import the code into a script using the Script Author. With the introduction of WYSIWYG panel editing, you can now create the entire contents of the panel HTML, including answers, in custom-generated HTML. Previously, answers were required to be created in the Script Author.

Requirements for Panel HTML Defined Using the Script Author

Before using an HTML editor or a text editor to create the panel HTML, you must ensure you have all the required information to code. You will need:

- Panel content, including:
  - Panel text (spoken and instructional text), including any formatting requirements.
  - Hypertext links.
  - Images, in Graphics Format Interchange (GIF) or Joint Photographic Experts Group (JPEG) format.
- Answer properties, including:
  - The boolean value for default for distinct branching.
  - The answer definition name (sometimes referred to as the question name).
  - The answer UI type. The range includes text fields, text areas, radio buttons, check boxes, buttons, drop-down lists, password fields, check box groups and multi-select list boxes.

- Data dictionary properties.

- Java Bean properties are no longer supported for answers in a panel. No information should be provided in the Answer Entry Dialog window in this section. You can still replace an entire panel with a Java bean.

- Data dictionary properties. Data dictionary properties may be defined for each answer definition. Possible data dictionary properties include:
  - Validation commands, to validate the answer provided at runtime.
  - Default commands, to provide a constant value for an answer.
  - Data sources, linking to tables to obtain answers from a database table.
  - Table information, to connect to database tables
  - Lookup reference information, to provide table, cursor, or command lookups, or to specify hard-coded lookup values as choices for each answer.

If creating a script for subsequent HTML modifications, you may want to use the Script Wizard to first generate a syntactically correct script, and then convert it to a graphical script, from which you can import custom HTML.

The Data Dictionary

Data dictionary properties may be defined for each answer definition. A separate data dictionary is maintained for each answer definition in a panel. Possible data dictionary properties include:

- Validation commands, to validate the answer provided at runtime.
- Default commands, to provide a constant value for an answer.
- Data sources, linking to tables to obtain answers from a database table.
- Table information, to connect to database tables
- Lookup reference information, to provide table, cursor, or command lookups, or to specify hard-coded lookup values as choices for each answer.

Some answer UI types require data dictionary properties to be defined. For example, buttons, radio buttons, drop-down lists, multi-select list boxes and checkbox groups require, at minimum, lookups to be defined.

> **Note:** Be aware that the standard checkbox answer UI type (as opposed to the checkbox group) obtains its contents not from a lookup value in

the data dictionary, but rather from the Label for reporting field in the Answer Entry Dialog window.

Text entry answer UI types (text fields, text areas and password fields) do not require any data dictionary properties to be specified. Nonetheless, data dictionary properties may be defined for any answer UI type. For example, password fields display asterisks when text is entered at runtime. However, without the script developer explicitly associating validation to an answer definition in the data dictionary, no validation is performed on password fields.

Drop-down lists and radio buttons only have binary validation. In other words, when an answer definition of one of these types is either the only answer definition in a panel, or is set as the default for distinct branching, the Scripting Engine will not allow the script end user to bypass either of these controls and progress to the next panel without the user first selecting an option from the control. Technically, the button answer UI type can also be considered to have binary validation, because a panel containing an explicitly defined button (which does not allow any other answer controls) must receive a button click in order to progress to the next panel.

Exporting Panel Text, Adding JavaScript, and Importing

Prerequisites

- Write the JavaScript code.
- Create a syntactically correct panel, including answer definitions.
- Identify the panel into which JavaScript will be included.

Steps

1. Create a complete and syntactically correct panel in the Script Author.
2. If the script is created using the Script Wizard, graph the script (convert it to a graphical script). From the Script Manager wizard window, select the appropriate script and click **Graph**.
3. Open the panel layout editor for the panel.
4. From the File menu, select **Export**.

   The Save window appears.
5. From the Location list and files displayed in the Files window, select the location on your network or filing system where you want to save the exported file.
6. In the File Name field, type the name you want to assign to the exported panel code, and click **Save**.

Note: The Script Author expects a file extension of .HTML. If you provide a file extension of .HTM, you may need to adjust the File Type filter to recognize the file when you attempt to re-import it.

7. Using a text editor or an HTML, modify the HTML code to include your JavaScript, and save your work.

8. In the Script Author, open the panel properties for the panel you edited.

9. From the panel properties tree, select Panel Layout.

10. In the Panel Layout pane, click Panel Layout Editor.

    The panel layout editor for that panel appears.

11. From the File menu, select **Import...**.

    The Open window appears.

12. From the Location list and files displayed in the Files window, locate the modified file you want to import. Use the File Type filter if necessary.

    When a suitable file is selected, its name appears in the File Name field.

13. Click **Open**.

    The code populates the panel. All information is visible in the panel layout editor.

14. From the File menu, select **Save** and then **Exit**.

    The panel text is saved and the panel layout editor window closes.

15. After any other desired changes in the Panel Properties window, click OK to save your work and close the window.

    The canvas appears.

16. From the File menu, select **Save**.

Guidelines

- Oracle Applications 11i-certified Web browsers such as Netscape Navigator and Microsoft Internet Explorer interpret JavaScript differently. This is traditionally an area of concern for Web developers. Issues with compatibility of custom JavaScript are generally a Web browser issue. As with all custom code, such issues are not supported by Oracle.

- Panels with JavaScript must either be created or edited outside of the Script Author and imported as panel text prior to compiling and deploying the script.

- Many custom JavaScripts are required to be included within the <HEAD> and </HEAD> tags of any HTML page.

- Microsoft FrontPage does not allow the inclusion of any content after the </HTML> tag. When editing panel text containing any embedded values, ensure that the HTML editor has not removed embedded value codes. One workaround is to create embedded values after otherwise customizing panel layout content.

Index

A

B

C

D

E

F

G

H

I

J

L

O

P

Q

R

S

U

W