Mercury IT Governance Center™
Commands, Tokens, and Validations Guide and Reference

Version: 6.0

MERCURY™

Mercury
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

If you have any comments or suggestions regarding this document, please send email to documentation@mercury.com.

Publication Number: ITG60CommandsTokens1104A

Table of Contents

List of Figures ix

List of Tables xi

Chapter 1: Introduction 15

About This Document 16

Who Should Read This Document 16

Related Documents 17

Overview 17

Chapter 2: Using Commands 19

Overview of Commands 20

Where Commands are Used 20

Commands Interface 20

Object Type Commands and Workflow 23

Request Type Commands and Workflow 23

Special Commands 24

Command Steps 25

Command Language 25

Command Conditions 25

Example Command Uses 26

Chapter 3: Special Commands 29

Overview of Special Commands 30

Special Command Interface 30

Special Command Workbench 31

Special Command Window ... 31
Special Command Window General Information Region ... 32
Parameters Tab ... 32
Commands Tab ... 33
Ownership Tab ... 37
Used By Tab ... 38
Creating and Editing Special Commands ... 38
Creating a New Special Command ... 38
Creating and Editing Special Command Parameters ... 40
Adding Parameters to Special Commands ... 40
Editing Special Command Parameters ... 42
Deleting Parameters ... 43
Setting Ownership for Special Commands ... 43
Using Special Commands ... 45
Adding Special Commands to Command Steps Using the Command Builder ... 46
Nesting Special Commands ... 47

Chapter 4: Using Tokens ... 49
Overview ... 50
What are Tokens? ... 50
Where Tokens Are Used ... 50
Token Builder Window Overview ... 51
Token Formats ... 53
Default Format ... 55
Explicit Entity Format ... 56
Using Tokens within Other Tokens ... 57
User Data Format ... 59
Parameter Format ... 60
Request Field Tokens ... 61
Sub-Entity Format ... 63
Environment and Environment Application Tokens ... 64
Token Evaluation ... 66

Chapter 5: Working with Validations ... 69
Overview of Working with Validations ... 70
What are Validations ... 70
Validation Component Types - Overview ... 71
Creating a Validation ... 73
User Data on the Validation Value ... 74
Editing Validations ... 75
Creating a URL to Open the Validation Window ... 76

Deleting Validations 77

Static List Validations 77

Dynamic List Validations 79
  SQL Validation 79
    SQL Validation Tips 81
  Command Validation 82

Configuring Auto-Complete Validations 82
  Configuring General Auto-complete Behavior 82
    Configuring Short List Auto-Complete Fields 83
    Configuring Long List Auto-Complete Fields 84
    Configuring the Automatic Value Matching and the Interactive Select Page 85
    Adding Search Fields to the Auto-Complete Window 88
    Special Case: Configuring an Auto-Complete List of Users 93
  Configuring the Auto-Complete Values 94
    Validation by Command With Delimited Output 95
    Validation by Command With Fixed Width Output 97
    User-Defined Multi-Select Auto-Complete Fields 99
    Example: Token Evaluation and Validation by Command with Delimited Output .100

Configuring Text Fields 104
  Creating a Text Field Validation Overview 104
  Available Text Data Masks 104
  Customizing the System Text Data Masks 106
    Customizing the Numeric Data Mask 106
    Customizing the Currency Data Mask 108
    Customizing the Percentage Data Mask 110
    Customizing the Telephone Data Mask 112
  Creating a Custom Data Mask 115

Using Directory and File Choosers 116
  Directory Chooser 116
  File Chooser 116

Date Field Formats 119

Creating 1800 Character Text Areas 120

Configuring the Table Component 121
  Defining the Table Component in the Validation Workbench 122
    Creating a Table Rule 126
    Calculating Column Totals 130
  Adding the Table Component to a Request Type 132

Package and Request Group Validations 134
  Package and Request Groups 134
  Request Type Category 135

Validation Special Characters 135

System Validations 136

Appendix A: System Special Commands **137**
Overview of System Special Commands 138
ksc_connect Special Commands 138
ksc_connect_dest_client 139
Example Using ksc_connect_dest_client 139
ksc_connect_dest_server 140
Example using ksc_connect_dest_server 140
ksc_connect_source_client 141
Example using ksc_connect_source_client 141
ksc_connect_source_server 142
Examples using ksc_connect_source_server 142
ksc_exit 143
ksc_copy Special Commands 143
ksc_copy_client_client 143
Example #1 using ksc_copy_client_client 144
Example #2 using ksc_copy_client_client 144
ksc_copy_client_server 144
Example using ksc_copy_client_server 145
ksc_copy_server_client 145
Example using ksc_copy_server_client 146
ksc_copy_server_server 146
Example using ksc_copy_server_server 147
ksc_copy_client_tmp 148
ksc_copy_server_tmp 148
ksc_copy_tmp_client 149
ksc_copy_tmp_server 150
ksc_respond 151
ksc_simple_respond 151
Examples using ksc_simple_respond 152
ksc_local_exec 153
Example using ksc_local_exec 153
ksc_replace 154
Example using ksc_replace 154
ksc_set 154
Example using ksc_set 155
ksc_set_env 155
ksc_store 156
Example using ksc_store 157
ksc_comment 157
ksc_concsub 158
Example using ksc_concsub 158

ksc_begin_script / ksc_end_script 159
Example using ksc_begin_script and ksc_end_script 160
ksc_copy_script Special Commands 160
ksc_copy_script_dest_client 161
ksc_copy_script_dest_server 161
ksc_copy_script_source_client 162
ksc_copy_script_source_server 162
ksc_om_migrate 163
Example using ksc_om_migrate 164
ksc_capture_output 164
ksc_gl_migrate 165
Example using ksc_gl_migrate 166
ksc_parse_jcl 166
ksc_submit_job 167
ksc_set_exit_value 167
ksc_clear_exit_value 168
ksc_run_sql 168
Example using ksc_run_sql 168
Summary of All Special Command Parameters 169

Appendix B: Tokens 175

Overview of Tokens 176
System Tokens 177
Field Group Tokens 213

Index 219

List of Figures

Figure 2-1 Commands tab and Edit Command window 21
Figure 2-2 New Command window 22
Figure 3-1 Special Command Workbench 31
Figure 3-2 Special Command window 32
Figure 3-3 Special Command window—Commands tab 34
Figure 3-4 Special Command Builder 37
Figure 3-5 Ownership tab 38
Figure 3-6 RCS File Migration object type 45
Figure 4-1 Example of a token used in a SQL statement 51
Figure 4-2 Token Builder window 52
Figure 4-3 Explicit Entity Format 56
Figure 4-4 Table component formats 62
Figure 5-1 Auto-complete using command validation 82
Figure 5-2 Short list auto-complete 83
Figure 5-3 Long list auto-complete 84
Figure 5-4 Auto-complete field and matching values in the Select page 85
Figure 5-5 Filter fields in the auto-complete select window 89
Figure 5-6 User icon 93
Figure 5-7 Auto-Complete List 94
Figure 5-8 Validation by command with delimited output 96
Figure 5-9 Validation by command with fixed width output 98
Figure 5-10 Validation window for the numeric data mask 107
Figure 5-11 Validation window for the currency data mask 109
Figure 5-12 Validation window for the percentage data mask 111

Figure 5-13 Validation window for the telephone data mask 113
Figure 5-14 Validation window for the custom data mask 115
Figure 5-15 Validation window for static environment override in file chooser. 117
Figure 5-16 Validation window for token-based environment override in file chooser. 118
Figure 5-17 Rules window accessed from the Rules tab 126
Figure 5-18 Sample validation for a Simple Order table component. 131
Figure 5-19 Sample table component displaying a column total. 132

List of Tables

Table 2-1 New Command window fields ... 22
Table 2-2 Command Steps region ... 25
Table 2-3 Example Conditions ... 26
Table 3-1 Special Command Workbench Query tab fields ... 31
Table 3-2 Special Command window general information fields ... 32
Table 3-3 Example Conditions ... 35
Table 3-4 New Command window fields ... 40
Table 4-1 Entities ... 53
Table 4-2 Tokens supporting explicit entity format ... 58
Table 4-3 Sample environment and app attributes ... 66
Table 4-4 Sample environment tokens ... 66
Table 5-1 Component Types ... 71
Table 5-2 Column Headers ... 80
Table 5-3 Automatic character matching field behavior ... 86
Table 5-4 Automatic character matching Select page behavior ... 86
Table 5-5 Fields in the Fields:New window ... 90
Table 5-6 Validation by command with delimited output ... 96
Table 5-7 Column headers ... 96
Table 5-8 Validation by command with fixed width output ... 98
Table 5-9 Column headers ... 98
Table 5-10 Data Mask Formats ... 105
Table 5-11 Fields for configuring the numeric data mask for text fields ... 107

Table 5-12 Fields for configuring the currency data mask for text fields 109
Table 5-13 Fields for configuring the percentage data mask for text fields 111
Table 5-14 Fields for configuring the telephone data mask for text fields 113
Table 5-15 Sample telephone data mask formats 114
Table 5-16 Sample custom data mask formats 115
Table 5-17 File chooser field 117
Table 5-18 Static environment override 118
Table 5-19 Token-based environment override 119
Table 5-20 Date field 120
Table 5-21 Example - table component validation settings 127
Table 5-22 Example - Set Unit Price rule settings 128
Table 5-23 Example - Calculate Total rule settings 129
Table A-1 ksc_connect_dest_client parameters 139
Table A-2 ksc_connect_dest_server parameters 140
Table A-3 ksc_connect_source_client parameters 141
Table A-4 ksc_connect_source_server parameters 142
Table A-5 ksc_copy_client_client parameters 143
Table A-6 ksc_copy_client_server parameters 145
Table A-7 ksc_copy_server_client parameters 146
Table A-8 ksc_copy_server_server parameters 147
Table A-9 ksc_copy_server_tmp parameters 148
Table A-10 ksc_copy_server_tmp parameters 149
Table A-11 ksc_copy_server_tmp parameters 149
Table A-12 ksc_copy_server_tmp parameters 150
Table A-13 ksc_replace parameters 154
Table A-14 ksc_set_env parameters 156
Table A-15 ksc_copy_script_dest_client parameters 161
Table A-16 ksc_copy_script_dest_server parameters 161
Table A-17 ksc_copy_script_source_client parameters 162
Table A-18 ksc_copy_script_source_client parameters 163
Table A-19 ksc_om_migrate parameters 163
Table A-20 ksc_gl_migrate parameters 165
Table A-21 ksc_parse_jcl parameters 166
Table A-22 ksc_submit_job parameters 167

Table A-23 ksc_run_sql parameters....168
Table A-24 Special command parameters....169
Table B-1 Token tables....176
Table B-2 App server properties....177
Table B-3 Budget....178
Table B-4 Contacts....178
Table B-5 Distribution....179
Table B-6 Document Management....180
Table B-7 Environments....180
Table B-8 Environment applications....183
Table B-9 Command execution....185
Table B-10 Notifications....185
Table B-11 Organization unit....186
Table B-12 Packages....187
Table B-13 Package lines....189
Table B-14 Package pending....190
Table B-15 Program....191
Table B-16 Project plans....191
Table B-17 Project plan details....194
Table B-18 Releases....194
Table B-19 Requests....195
Table B-20 Request details....198
Table B-21 Request pending....198
Table B-22 Report submissions....199
Table B-23 Resource pools....200
Table B-24 Security groups....201
Table B-25 Skill....201
Table B-26 Staffing profile....202
Table B-27 System....202
Table B-28 Tasks....203
Table B-29 Tasks pending....205
Table B-30 Users....206
Table B-31 Validations....208
Table B-32 Validation values....208

Table B-33 Workflows....209
Table B-34 Workflow steps....210
Table B-35 Workflow step transaction....212
Table B-36 Field group token tables....213
Table B-37 Demand Management field group tokens....214
Table B-38 Master project reference on request field group tokens....214
Table B-39 PFM asset field group tokens....214
Table B-40 PFM project field group tokens....215
Table B-41 PFM proposal field group tokens....216
Table B-42 PMO field group tokens....216
Table B-43 Program reference on request field group tokens....217
Table B-44 Work item field group tokens....217

Chapter 1 Introduction

In This Chapter:

- *About This Document*
- *Who Should Read This Document*
- *Related Documents*
- *Overview*

About This Document

This document provides information on using and configuring commands, tokens, and validations in Mercury IT Governance Center™. Each chapter or appendix covers specific topics on commands, tokens, or validations:

- Chapter 1, *Introduction,* on page 15

  Includes an overview of the use of commands, tokens, and validations, and details the document's intended audience and related guides.

- Chapter 2, *Using Commands,* on page 19

  Provides an overview and examples for using commands.

- Chapter 3, *Special Commands,* on page 29

  Discusses the interface for creating, editing and using special commands in the Mercury IT Governance Center.

- Chapter 4, *Using Tokens,* on page 49

  Provides an overview of how to use tokens.

- Chapter 5, *Working with Validations,* on page 69

  Discusses the creation and editing of validations.

- Appendix A: *System Special Commands* on page 137

  Discusses pre-defined special commands.

- Appendix B: *Tokens* on page 175

  Provides a list of all entity tokens.

Who Should Read This Document

The intended audience for this document includes:

- Configuration experts configuring a deployment system.
- Configuration experts configuring a request resolution system.
- Business modelers who need to modify the following entities: workflows, object types, request types, validations, notifications, and report types.

Note

Users must have the Configuration license to access the screens and windows described in this document.

Related Documents

Related documents for this book are:

- *Guide to Documentation*
- *Key Concepts*
- *Getting Started*
- *Mercury Demand Management: Configuring a Request Resolution System*
- *Mercury Change Management: Configuring a Deployment System*
- *Configuring the Standard Interface*

Overview

Commands, tokens, and validations are used throughout the Mercury IT Governance Center implementation to enable advanced automation and defaulting.

Commands define the heart of the execution layer within the deployment system and determine which steps to execute at a specific workflow step. This can involve activities such as migrating a file, executing a script, performing some data analysis, or compiling code.

Tokens are variables that can be used by Mercury IT Governance Center entities to reference information that is undefined until the entity is actually used in a particular context. This includes such things as setting variables in commands or using tokens within notifications to specify the recipients.

A field's validation determines not only its type, but the values it can accept. A workflow step's validation control the possible results exiting that step. Altering the validation for a field can range from changing its very nature (text field to auto-complete), while modifications to a workflow step validation make it possible to specify new results for that step.

Chapter

2

Using Commands

In This Chapter:

- *Overview of Commands*
  - *Where Commands are Used*
  - *Commands Interface*
  - *Object Type Commands and Workflow*
  - *Request Type Commands and Workflow*
  - *Special Commands*
- *Command Steps*
  - *Command Language*
- *Command Conditions*
- *Example Command Uses*

Overview of Commands

Commands define the heart of the execution layer within the deployment system and determine which steps to execute at a specific workflow step. This can involve activities such as migrating a file, executing a script, performing some data analysis, or compiling code. This chapter explains where to find commands, the interface used to manipulate commands, command structure, and gives some uses for commands.

Where Commands are Used

Commands are used in the following entities to enhance the implementation and enable sophisticated command-line automation:

- Object types
- Request types
- Report types
- Workflows
- Validations

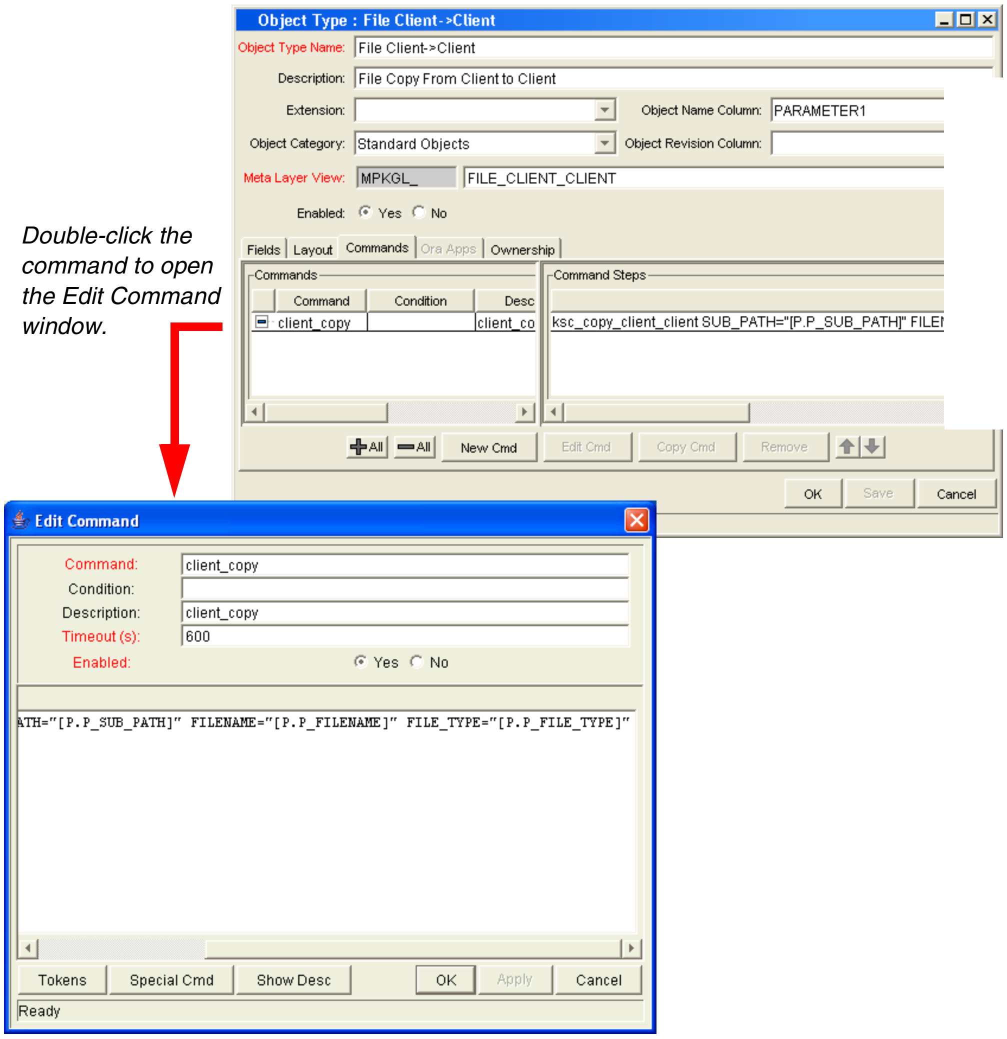
Commands Interface

Access commands through the **Commands** tab of the following screens:

- Object Type
- Request Type
- Report Type
- Validation
- Workflow Step Source
- Special Command

Commands consist of command information and command steps. In this chapter, the examples are accessed through the Change Mgmt: Object Types screen, but the interface is the same in other screens where commands are configured.

Double-click the command step to open the Edit Command window. The Edit Command window displays the shell script code in the Steps window, as shown in *Figure 2-1*.



Object Type Name: File Client->Client
Description: File Copy From Client to Client
Extension:
Object Name Column: PARAMETER1
Object Category: Standard Objects
Object Revision Column:
Meta Layer View: MPKGL_ FILE_CLIENT_CLIENT
Enabled: [✓ Yes] No

Command	Condition	Desc
client_copy		client_co

Command Steps: ksc_copy_client_client SUB_PATH="[P.P_SUB_PATH]" FILEI

Command: client_copy
Condition:
Description: client_copy
Timeout (s): 600
Enabled: [✓ Yes] No

```
ATH="[P.P_SUB_PATH]" FILENAME="[P.P_FILENAME]" FILE_TYPE="[P.P_FILE_TYPE]"
```

Figure 2-1. Commands tab and Edit Command window

To generate a new command, click **New Cmd** in the **Commands** tab. This opens the New Command window as shown in *Figure 2-2*. *Table 2-1* lists the fields contained in this window.

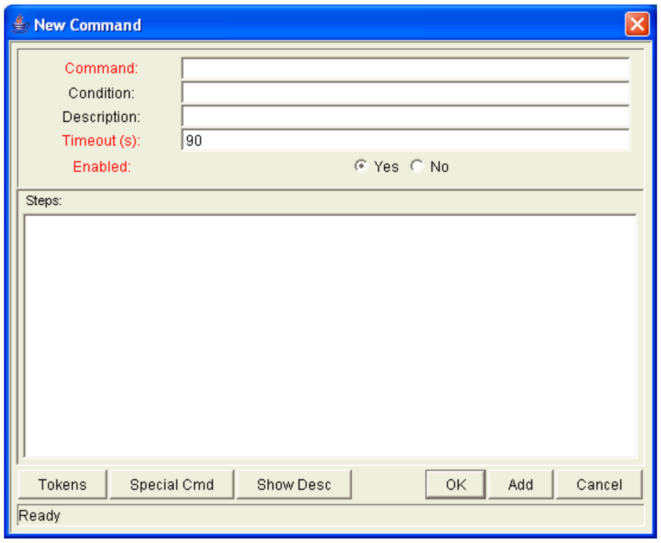


Figure 2-2. New Command window

Table 2-1. New Command window fields

Field	Description
Command	A simple name for the command.
Condition	A condition that determines whether the steps for the command are executed or not. (See *Command Conditions* on page 25 below for more information).
Description	A description of the command.
Timeout(s)	The amount of time the command will be allowed to run before its process is terminated. This mechanism is used to abort commands that are hanging or taking an abnormal amount of time.
Enabled	Determines whether the command is enabled for execution.

Each object type, request type, validation, workflow step source, or report type may have many commands, and each command may have many command steps. A command may be viewed as a particular function for an object. Copying a file may be one command, and checking that file into version control may be another. To perform these functions, a series of events needs to take place, and these events are defined in the command steps.

An additional level of flexibility is introduced when some commands must only be executed in certain cases. This is powered by the condition field of the commands and is discussed in *Command Conditions* on page 25.

Object Type Commands and Workflow

Object type commands are tightly integrated with the workflow engine. The commands contained in an object type are executed at Execution workflow steps in Mercury Change Management™ package lines.

It is important to note the following concepts regarding command/workflow interaction:

- To execute object type commands at a particular workflow step, the workflow step must be configured with the following parameters:
  - Workflow step must be an execution type step.
  - Workflow Scope = **Packages**.
  - Execution Type = **Built-in Workflow Event**.
  - Workflow Command = **execute_object_commands**.
- When the object reaches the workflow step (with Workflow Command = **execute_object_commands**), all object type commands whose conditions are satisfied will be run in the order they are entered in the object type's command panel.
- The object type can be configured to run only certain commands at a particular step. To do this, specify **command conditions**. For details, see *Command Conditions* on page 25.

Request Type Commands and Workflow

Similar to object type commands, request type commands define the execution layer within Mercury Demand Management™. While most of the resolution process for a request is analytically based, cases may arise for specific request types where system changes are required. In these cases, request type commands can be used to automatically perform these changes.

Request type commands are tightly integrated with the workflow engine. The commands contained in a request type are executed at Execution workflow steps.

It is important to note the following concepts regarding command/workflow interaction:

- To execute request type commands at a particular workflow step, the workflow step must be configured with the following parameters:

- Workflow step must be an execution type step
- Workflow Scope = **Requests**
- Execution Type = **Built-in Workflow Event**
- Workflow Command = **execute_request_commands**

- When the request reaches the workflow step (with Workflow Command = **execute_request_commands**), all commands whose conditions are satisfied will be run in the order they are entered in the request type's command panel.
- The request type can be configured to run only certain commands at a particular step. To do this, specify **command conditions**. For details, see *Command Conditions* on page 25.

Special Commands

Object types, request types, report types, workflows and validations all use commands to access the execution layer. In order to simplify the use of command executions, Mercury IT Governance Center contains a predefined set of special commands. Users can also create their own special commands.

Special commands are commands with variable parameters, and are used in object type, request type, report type, workflow, and validation command steps. These command steps perform a variety of functions, such as copying files between environments and establishing connections to environments for remote command execution. Mercury IT Governance Center features two types of special commands:

- **System special commands.** These commands are shipped with Mercury IT Governance Center. System special commands are read-only and have the naming convention "ksc_command_name." System special commands always begin with "ksc_."
- **User-defined special commands.** These commands are user-defined and have the naming convention "sc_command_name." User-defined special commands must begin with "sc_."

Special commands act as subprograms that can be reused where needed. It it often more convenient to create a special command for a program that will be used in multiple places, rather than placing the individual commands into every object type or request type that need them.

Command Steps

Command steps represent the actual directives that Mercury IT Governance Center specifies to execute the commands. A command step can be an actual command-line directive that is sent to the Mercury IT Governance Server or target machine, or it can be one of the many special commands. *Table 2-2* describes the fields in the Command Steps region of the New/Edit Commands dialog.

Table 2-2. Command Steps region

Field	Description
Steps	Defines the command-line directive or special command to be issued.
Description	Describes each of the command steps.

Note

The execution engine will execute the commands and command steps in the order they are displayed in the **Commands** tab. To change the order of the commands or the command steps, in the **Commands** tab, select the given command or command step and use the arrow buttons to move the selected item.

Command Language

The command steps in a command define the actual system-level executions that need to be performed to achieve the desired function of the command. Command steps can be UNIX commands, third party application commands, or special commands. Special commands are reusable routines defined in Mercury IT Governance Center. Mercury IT Governance Center also supplies a number of system special commands used to perform common execution events (such as connecting to environments or copying files). Tokens can be used within command steps.

Command Conditions

In many situations, it may be necessary to run a different set of commands depending on the context of execution. This flexibility is achieved through the use of conditional commands. The Condition field for a command is used to define the situation under which the associated command steps execute.

Conditions are evaluated as boolean expressions. If the expression evaluates to true, the command is executed. If false, the command is skipped and the next command is evaluated. If no condition is specified, the command is always executed. The syntax of a condition is identical to the “where” clause of a SQL statement, which allows enormous flexibility when evaluating scenarios. Some example conditions are detailed in the following table:

Table 2-3. Example Conditions

Condition	Evaluates to
BLANK	Command will be executed in all situations.
‘[P.P_VERSION_LABEL]’ IS NOT NULL	Command will be executed if the parameter with the token P_VERSION_LABEL in the package line is not null.
‘[DEST_ENV.ENVIRONMENT_ NAME]’ = ‘Archive’	Command will be executed when the destination environment is named “Archive”.
‘[AS.SERVER_TYPE_CODE]’ = ‘UNIX’	Command will be executed if the application server is installed on a UNIX machine.

Note

Be sure to place single quotes around string literals or tokens that will evaluate strings.

The condition can include tokens. For more information, see *Using Tokens* on page 49.

Example Command Uses

This section provides a number of operations that can be executed using commands. Sample code for configuring many of these cases is included in *System Special Commands* on page 137.

- Commands for connecting to machines
  - Connect to the destination environment and run system commands

    - Connect to an alternate environment and run command (environment override)
- Commands for manipulating data (fields and other information stored in files or database)
    - Set a value in a package line
    - Create, run and delete a script
    - Extract information from a file (version number)
- Commands for running operating system-specific commands (NT and UNIX)
    - Starting a server
    - Stopping a server
- Commands for running program-specific commands
    - Checking files in and out of a version control system
- Commands for copying files

Chapter 3 Special Commands

In This Chapter:

- *Overview of Special Commands*
- *Special Command Interface*
  - *Special Command Workbench*
  - *Special Command Window*
- *Creating and Editing Special Commands*
  - *Creating a New Special Command*
  - *Creating and Editing Special Command Parameters*
  - *Setting Ownership for Special Commands*
- *Using Special Commands*
  - *Adding Special Commands to Command Steps Using the Command Builder*
  - *Nesting Special Commands*

Overview of Special Commands

Object types, request types, report types, workflows and validations all use commands to access the execution layer. In order to simplify the use of command executions, Mercury IT Governance Center contains a predefined set of special commands. Users can also create their own special commands.

Special commands are commands with variable parameters and are used in object types, request types, report types, workflows, and validation command steps. (Workflows use special commands in their workflow step sources.) These command steps perform a variety of functions, such as copying files between environments and establishing connections to environments for remote command execution. Mercury IT Governance Center features two types of special commands:

- System special commands - These commands are shipped with the Mercury IT Governance Center. System special commands are read-only and have the naming convention ‘ksc_command_name.’ System special commands always begin with ‘ksc_.’
- User defined special commands - These commands are user-defined and have the naming convention ‘sc_command_name. User-defined special commands must begin with ‘sc_.’

This chapter discusses the interface for creating, editing and using special commands in Mercury IT Governance Center.

Note

See *System Special Commands* on page 137 for a detailed description of system special commands and their parameters.

Special Command Interface

Use the Workbench interface to create, view and edit special commands using the Special Command Workbench shown in *Figure 3-1*.

To access the Special Command Workbench:

1. Log on to Mercury IT Governance Center and open the Workbench.

2. From the shortcut bar, select **Configuration > Special Commands.**

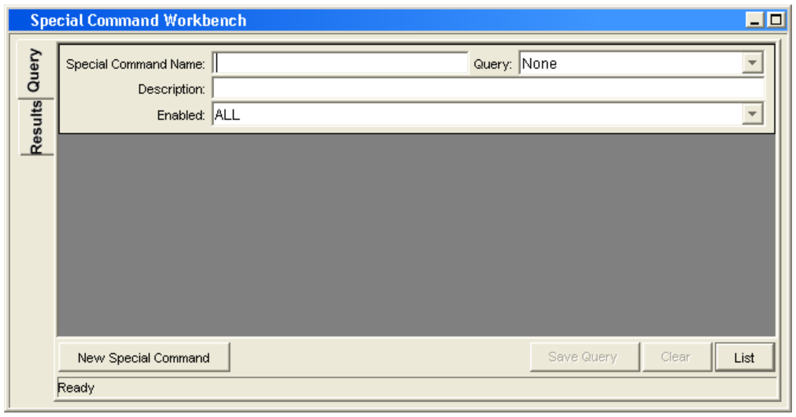
   The Special Command Workbench opens.

Special Command Workbench

Use the Special Command Workbench to search for a particular special command in the **Query** tab using the criteria defined in *Table 3-1*.

Table 3-1. Special Command Workbench Query tab fields

Field	Description
Special Command Name	Filter for special commands where the name matches a given string.
Description	Filter for special commands where the description matches a given string.
Enabled	Filter for special commands that are enabled or disabled.



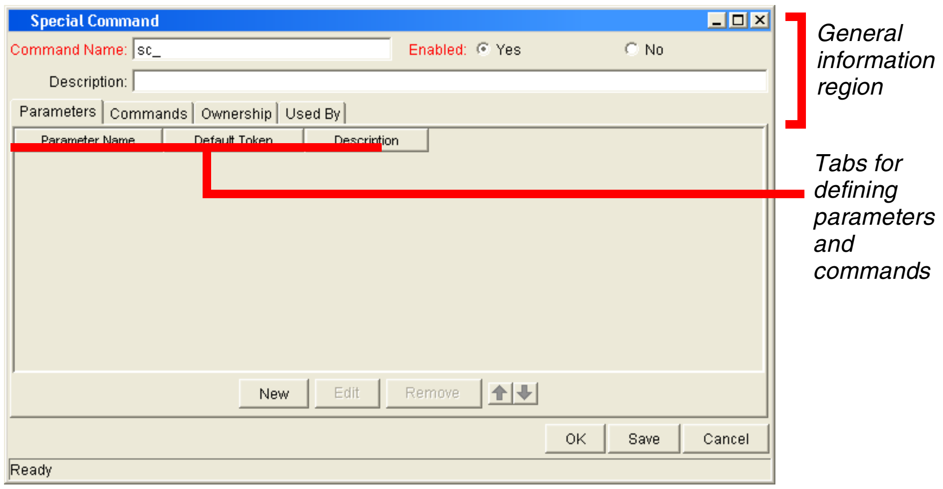
Special Command Name:
Query: None
Description:
Enabled: ALL

Ready

Figure 3-1. Special Command Workbench

Special Command Window

Use the Special Command window shown in *Figure 3-2* to define and configure special commands.


Special Command

Command Name: sc_

Enabled: [✓ Yes] No

Description:

Figure 3-2. Special Command window

Special Command Window General Information Region

The Special Command window's general information region displays the basic header information for the special commands. It consists of the fields described in *Table 3-2*.

Table 3-2. Special Command window general information fields

Field	Description
Command Name	The name of the special command. This can only be updated when generating or editing a user-defined special command.
Enabled?	Determines whether or not the special command is enabled for use in workflows, object types, report types, request types and validations.
Description	A description of the special command. This can only be updated when generating or editing a user-defined special command.

Parameters Tab

The **Parameters** tab displays the current parameters for the special command. Most special commands have parameters to override standard behavior. Nearly all parameters are optional. When a parameter is not passed to a special command and the default value for the parameter is a custom token, the entity using the command must contain a field with that token.

For example: The 'ksc_copy_server_server' special command shown is used in an object type. The parameter FILENAME is not specified and defaults to [P.P_FILENAME] because it is not explicitly passed.

```
ksc_copy_server_server
```

This makes 'ksc_copy_server_server' equivalent to:

```
ksc_copy_server_server FILENAME="[P.P_FILENAME]"
```

because "[P.P_FILENAME]" is the default token for the parameter FILENAME. The command execution engine evaluates the token [P.P_FILENAME] so it must be defined for the entity (the specific object type, report type or request type).

To override the default token, pass in another value for the parameter. A few examples are:

```
ksc_copy_server_server FILENAME="document.txt"
ksc_copy_server_server FILENAME="[P.DOCUMENT_NAME]"
```

This method of passing parameters is explained in more detail in the section entitled *Special Command Builder* on page 37.

Note

Custom tokens are defined for specific object types, request types, and report types, and are referenced using the "[P.TOKEN_NAME]" syntax. See *System Special Commands* on page 137 for a list of all predefined special command parameters and their default tokens.

Commands Tab

Use the **Commands** tab to define and configure the commands and command steps used by each user-defined special command. It is also possible to view the command information for the predefined system special commands.

Commands are designed to have a similar look-and-feel to the UNIX and DOS operating system command structure. The specific parts of a command, the command steps, are often just command-prompt directives.

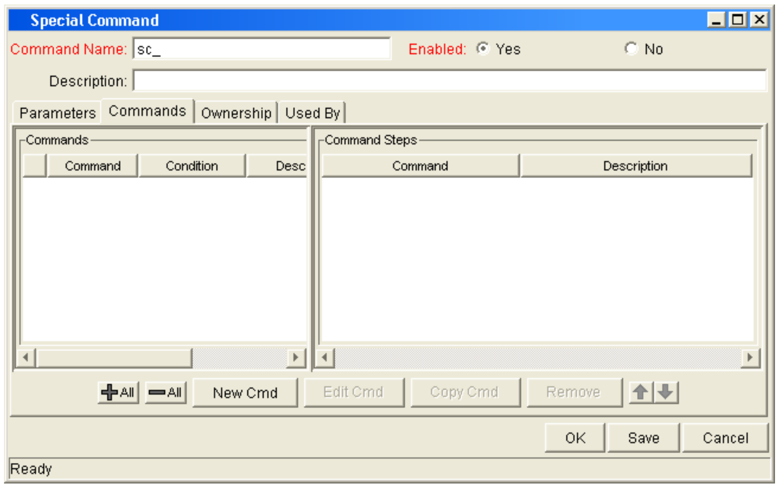


Figure 3-3. Special Command window—Commands tab

Commands are accessible through the **Commands** tab of the Special Commands window and consist of command information and command steps.

Command Conditions

In many situations, it may be necessary to run a different set of commands depending on the context of execution. For example, one command may be needed to update a Web page, while another command may be required to set-up an account on the Sales Automation application.

This flexibility is achieved through the use of conditional commands. The Condition field for an object command provides the ability to define the situation under which the associated command steps will execute.

Conditions are evaluated as Boolean expressions. If the expression evaluates to TRUE, the command is executed. If FALSE, the command is skipped and the next command is evaluated to see if it should run. If no condition is specified the command is always executed. The syntax of a condition is identical to the WHERE clause of a SQL statement, which allows flexibility when evaluating scenarios. Some example conditions are given in *Table 3-3*.

Table 3-3. Example Conditions

Condition	Evaluates to
BLANK	Command executes in all situations.
‘[REQ.DEPARTMENT]’ = ‘SALES’	Command executes when the department for the request is named SALES.
‘[REQ.PRIORITY]’ = ‘HIGH’	Command executes if the priority assigned to the request is HIGH.

Note

When using conditional commands, strings must be enclosed by single quotes.

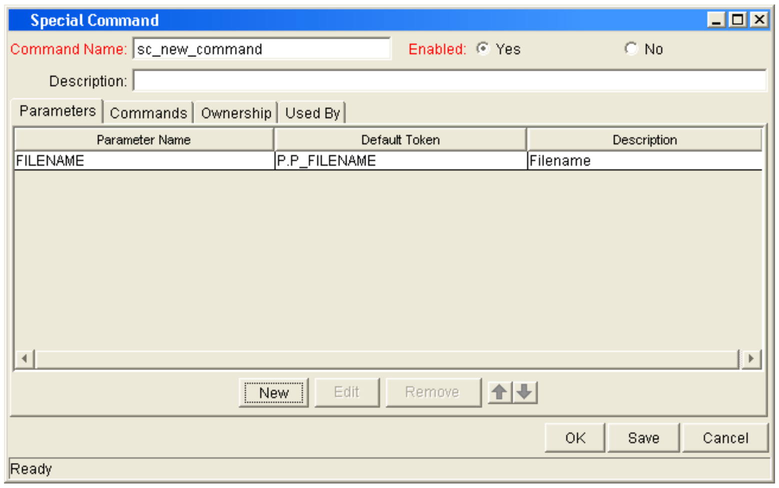
The condition can include a token. See *Using Tokens* on page 49 for more information.

Parameters in Command Steps

In the command steps within a special command, parameters are referred to as their default tokens. When the special command is executed with a value specified for a parameter, this value will replace the default token throughout the special command steps.

Example - Special Command

An existing special command echoes a string as an HTML tag named sc_echo_html and takes the parameter RAW_TEXT. This example shows how to create another special command named sc_new_command. This special command will use sc_echo_html to echo the parameter value FILENAME, which has a default token of [P.P_FILENAME].



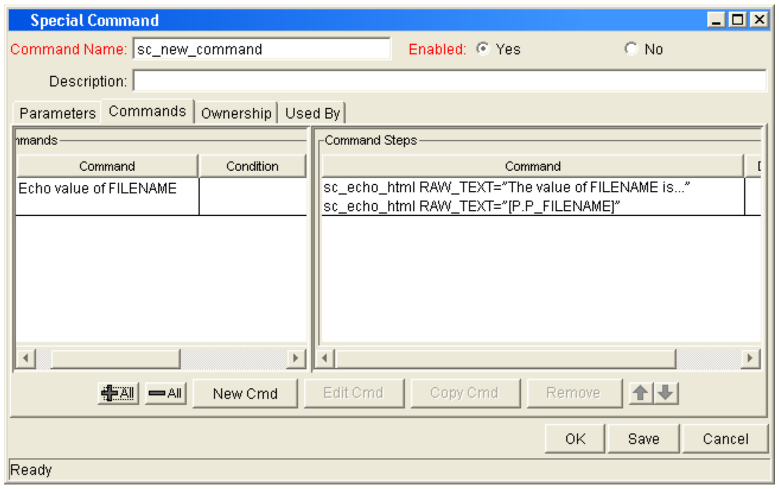
Command Name: sc_new_command
Enabled: (●) Yes ( ) No
Description:

Parameter Name	Default Token	Description
FILENAME	P.P_FILENAME	Filename

Ready

To accomplish this, the following command steps are entered in a command for sc_new_command:

```
sc_echo_html RAW_TEXT="The value of FILENAME is..."
sc_echo_html RAW_TEXT="[P.P_FILENAME]"
```



Command Name: sc_new_command
Enabled: (●) Yes ( ) No
Description:

Command	Condition
Echo value of FILENAME	

Command
sc_echo_html RAW_TEXT="The value of FILENAME is..."
sc_echo_html RAW_TEXT="[P.P_FILENAME]"

Ready

Note that the command step uses the default token to refer to the value of the special command parameter. The parameter name is only used when invoking a special command.

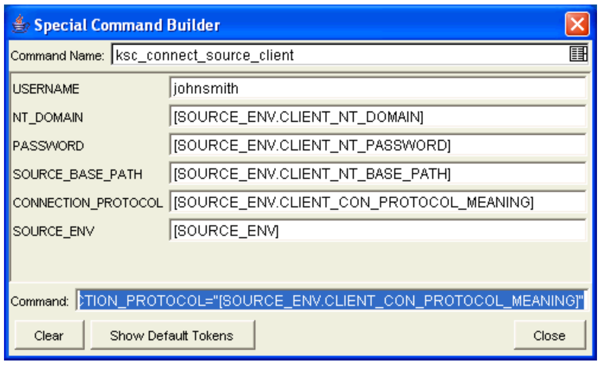
Note

Parameters cannot be used in command conditionals.

Continuing from the previous example, suppose that a special command has the parameter FILENAME, whose default token is [P.P_FILENAME]. In command conditionals, the token [P.P_FILENAME] will always be evaluated normally, regardless of whether our special command was called with a value for the parameter FILENAME.

Special Command Builder

The Special Command Builder is a tool designed to simplify the use of special commands by ensuring proper formatting of the command step. The Special Command Builder, shown in *Figure 3-4*, is an interface where a special command can be selected and appropriate parameters can be entered. The Special Command Builder outputs a line of text to the Command field which can be used as a command step.



Command Name: ksc_connect_source_client

USERNAME	johnsmith
NT_DOMAIN	[SOURCE_ENV.CLIENT_NT_DOMAIN]
PASSWORD	[SOURCE_ENV.CLIENT_NT_PASSWORD]
SOURCE_BASE_PATH	[SOURCE_ENV.CLIENT_NT_BASE_PATH]
CONNECTION_PROTOCOL	[SOURCE_ENV.CLIENT_CON_PROTOCOL_MEANING]
SOURCE_ENV	[SOURCE_ENV]

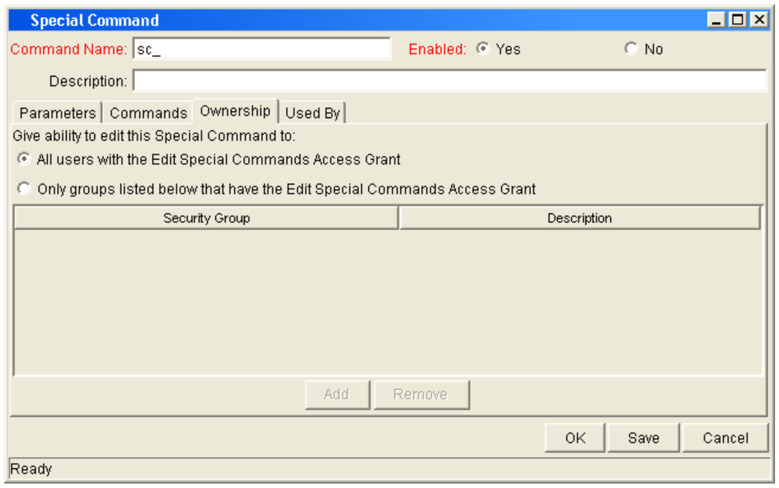
Command: :TION_PROTOCOL="[SOURCE_ENV.CLIENT_CON_PROTOCOL_MEANING]"

Figure 3-4. Special Command Builder

Ownership Tab

The **Ownership** tab is used to select ownership groups for a specific special command. Members of ownership groups are the only users who have the right to edit, copy or delete this special command. This tab also displays ownership groups that have been linked to this entity. Ownership groups can be deleted from this tab by selecting them and clicking **Remove**.

See *Setting Ownership for Special Commands* on page 43 for more information about setting ownership for a new or existing special command.



Special Command

Command Name: sc_   Enabled: [✓ Yes] No

Description:

Parameters | Commands | Ownership | Used By

Give ability to edit this Special Command to:

[✓ All users with the Edit Special Commands Access Grant]

Only groups listed below that have the Edit Special Commands Access Grant

Security Group	Description

Figure 3-5. Ownership tab

Used By Tab

Click the **Used By** tab to view a list of entities that currently refer to the selected special command.

Creating and Editing Special Commands

This section details key procedures for creating and editing special commands.

Creating a New Special Command

To create a new special command:

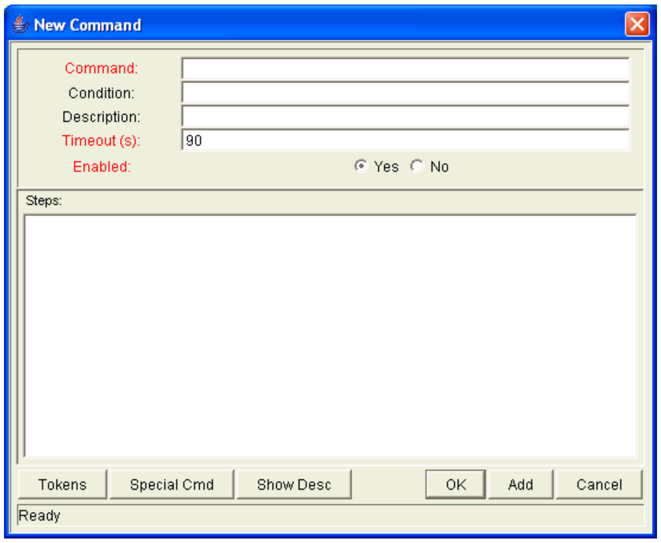
1. From the Special Command Workbench, click **New Special Command.**

   The Special Command window opens.

2. Click the **Commands** tab.

3. Click **New Cmd.**

   The New Command window opens. This window's fields are defined in *Table 3-4* on page 40.



Timeout (s): 90

Enabled: [✓ Yes] No

4. Enter information in the Command, Condition and Description fields.

   See *Command Conditions* on page 34 for more details about defining Conditions.

5. Select the **Yes** option for the Enabled radio button.

6. Add tokens to the new special command as desired.

   a. Click **Tokens.**

      The Token Builder window opens.

   b. Copy a token from the Token Builder window.

   c. Paste it into the New Command window's Steps text area.

7. Add another special command to the new special command.

   a. Click **Special Cmd.**

      The Special Command Builder window opens.

   b. In the Command Name field, select a special command and enter any required parameters.

   c. Copy the special command from the Special Command Builder window.

   d. Paste it into the New Command window's Steps text area.

8. To add the new command to the **Command** tab of the Special Command window without closing the New Command window, click **Add.**

9. To add the new command to the **Command** tab of the Special Command window and close the New Command window, click **OK.**

   The new special command has been created.

10. To save the new special command, click **Save.**

Table 3-4. New Command window fields

Field	Description
Command	The name of the command.
Condition	A condition that determines whether the command steps for the command are executed or not. (See *Command Conditions* on page 34 for more information).
Description	A description of the command.
Enabled?	Determines whether the command is enabled for execution.

Creating and Editing Special Command Parameters

This section describes procedures for creating and editing special command parameters.

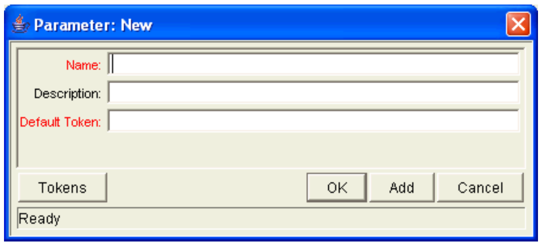
Adding Parameters to Special Commands

This section describes the procedure for adding parameters to a special command.

To add a new parameter to a user-defined special command:

1. In the **Parameters** tab of the Special Command window, click **New.**

   The Parameter: New window opens.

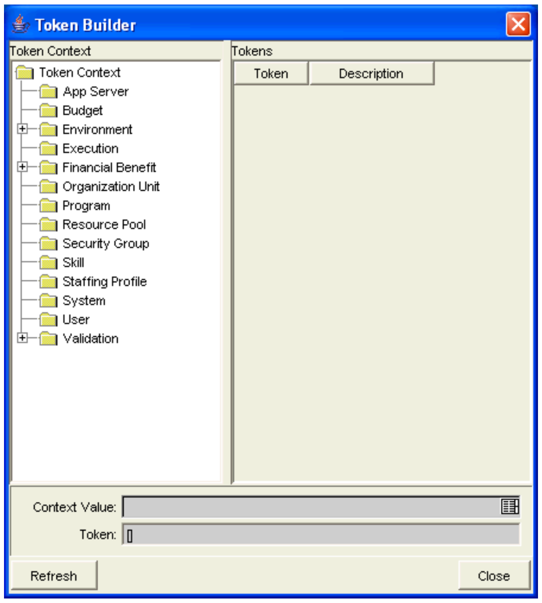


2. Fill in the Name, Description and Default Token fields.

   To select an existing global token, follow step 3 through step 9. To manually entered a token name in the Default Token field, go to step 7.

3. To select an existing global token, click **Tokens.**

   The Token Builder window opens.



4. In the Token Context pane of the window, select a folder.

   The available tokens for each folder display in the Tokens pane of the window.

5. In the Token column, select a token.

   When a token is selected, it enables the Token field and displays the name of the selected token (including its prefix).

6. Copy the token.

   a. Select the token in the Token field.

   b. Press **Ctrl+c** on the keyboard.

7. In the Parameter window, paste the token name into the Default Token field by pressing **Ctrl+v** on the keyboard.

8. To add the field to the **Parameters** tab and close the Parameter window, click **OK.**

9. To add the field to the **Parameters** tab without closing the Parameter window, click **Add.**

Editing Special Command Parameters

This section describes the procedure for editing special command parameters.

To edit an existing parameter:

1. Open the special command.

2. In the **Parameters** tab, double-click the parameter.

   The Parameter window opens.

3. Make the desired changes in the Parameter window.

4. Click **Apply** to apply the changes without closing the Parameter window.

5. Click **OK** to apply the changes and close the Parameter window.

Note

The parameter order can be altered by selecting a parameter in the **Parameters** tab and clicking either the **Up** or **Down** arrow.

Changes to parameters already used by existing request types, object types, or report types can affect the way these entities function.

Deleting Parameters

This section describes the procedure for deleting special command parameters.

To delete a parameter:

1. Open the special command.
2. Select the parameter in the **Parameters** tab.
3. Click **Remove.**
4. Click **OK** to save the information and close the Special Command window**.**
5. Click **Save** to save the information without closing the Special Command window.

   The parameter is deleted from the special command.

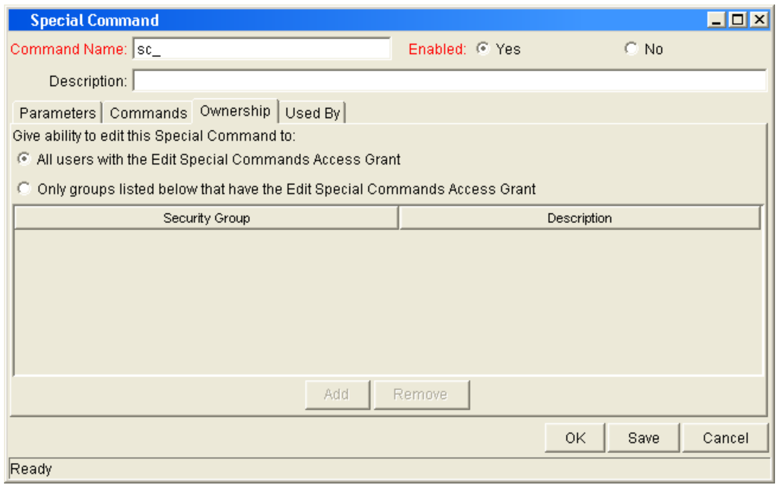
Setting Ownership for Special Commands

Different groups of users can have exclusive control over the special commands used by their group. These groups are referred to as ownership groups. Members of the ownership group are the only users who can edit, delete or copy the special commands. Each special command can be assigned multiple ownership groups.

Ownership groups are defined in the Security Group window in the Workbench. See the *Security Model Guide and Reference* for instructions on setting up security groups.

To set the ownership for a special command:

1. Open the Special Command window.
2. Click the **Ownership** tab.



Command Name: sc_
Enabled: [✓ Yes] No
Description:
Give ability to edit this Special Command to:
[✓ All users with the Edit Special Commands Access Grant]
Only groups listed below that have the Edit Special Commands Access Grant

Security Group	Description

Ready

3. Select the Only groups listed below that have the Edit Special Commands Access Grant option.

4. Click **Add.**

   The Add Security Groups window opens.

5. In the Security Group auto-complete list, select a security group.

6. To close the Add Security Group window, click **OK.**

   The selected security groups are display in the **Ownership** tab under the security group column.

7. To save the changes and close the window, click **OK** in the Special Command window.

   To save the selection and leave the Special Command window open, click **Save.**

Only members of the security group(s) specified in the **Ownership** tab can edit, delete, or copy this special command.

Note

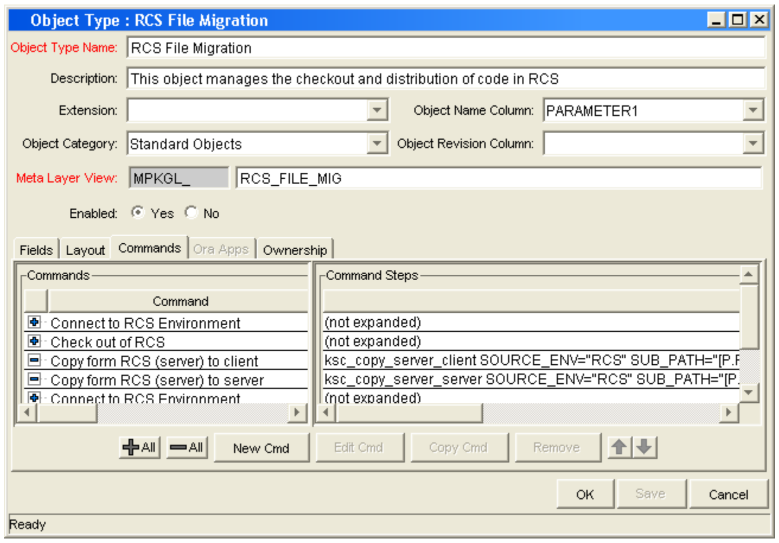
If no ownership groups are associated with the entity, the entity is considered global and any user with the Edit access grant for the entity can edit, copy or delete it. For more information on access grants, see the *Security Model Guide and Reference*.

By default, administrators have the 'Ownership Override' access grant and can access configuration entities even if the administrator is not a member of one of the ownership groups and does not have the Edit access grant.

If a security group is disabled or loses the Edit access grant, that group will no longer have edit access for the entity.

Using Special Commands

Special commands are added to command steps directly in the entity windows (object types, request types, report types, validations and workflows). For example, *Figure 3-6* shows an object type that has been generated using a combination of special commands.



Object Type Name: RCS File Migration
Description: This object manages the checkout and distribution of code in RCS
Extension:
Object Name Column: PARAMETER1
Object Category: Standard Objects
Object Revision Column:
Meta Layer View: MPKGL_ RCS_FILE_MIG
Enabled: [✓ Yes] No

Command	Command Steps
Connect to RCS Environment	(not expanded)
Check out of RCS	(not expanded)
Copy form RCS (server) to client	ksc_copy_server_client SOURCE_ENV="RCS" SUB_PATH="[P.F
Copy form RCS (server) to server	ksc_copy_server_server SOURCE_ENV="RCS" SUB_PATH="[P.
Connect to RCS Environment	(not expanded)

Figure 3-6. RCS File Migration object type

Adding Special Commands to Command Steps Using the Command Builder

Special commands can be added to any set of command steps in the following entities:

- Object types
- Request types
- Report types
- Validations
- Workflow step sources
- Other special commands

Access the Special Command Builder in the **Commands** tab for each of these entities.

To build a command step using the Special Command Builder:

1. Go to the **Commands** tab for the entity which commands will be added.
2. Click **New Cmd** or edit an existing command.

   The Command window opens.
3. Click **Special Cmd.**

   The Special Command Builder window opens.
4. Enter the a command name in the Command Name field, or select it from the auto-complete list.

   When selecting a command name from the auto-complete list, its parameters appear in the Special Command Builder.

Note

Both predefined (ksc_command) and user defined (sc_command) special commands can be used to build the command steps line. For more information on generating special commands, see *Special Command Interface* on page 30.

5. Replace the associated default token value with any desired parameter information.

   a. To view the default tokens, click **Show Default Tokens.**

   b. To hide the default tokens, click **Hide Default Tokens.**

6. When the parameters have been modified, select the text in the Command field.

   To copy the formatted special command, press **Ctrl+c** on the keyboard.

7. To close the Special Command Builder window, click **Close.**

8. To paste the special command step, click in the steps text area of the New Command window and press **Ctrl+v** on the keyboard.

9. Fill in the remaining fields in the New Command window.

10. Select the **Yes** option for the Enabled radio button.

11. To add the command step to the **Command** tab, click **OK.**

The new special command is now ready to be used in an object type, request type, report type, validation, or workflow.

Note

Special commands can be used in an execution workflow step source. After the workflow step source is created (which contains the special commands), it can be dragged and dropped into a workflow.

Nesting Special Commands

Special commands can be used within other special commands, but must be used within a command step. However, a special command cannot refer to itself.

Chapter

4

Using Tokens

In This Chapter:

- *Overview*
- *What are Tokens?*
- *Where Tokens Are Used*
- *Token Builder Window Overview*
- *Token Formats*
  - *Default Format*
  - *Explicit Entity Format*
  - *User Data Format*
  - *Parameter Format*
  - *Sub-Entity Format*
  - *Environment and Environment Application Tokens*
- *Token Evaluation*

Overview

This chapter provides an overview of how to use tokens.

What are Tokens?

While configuring certain features, it is often necessary to reference information that is undefined until Mercury IT Governance Center is actually used a particular context. Instead of generating objects that are valid only in specific contexts, Mercury IT Governance Center uses variables to facilitate the creation of general objects that can be applied to a variety of contexts. These variables are called tokens.

There are two types of tokens found within Mercury IT Governance Center: custom tokens and standard tokens. Standard tokens are provided with the product. Custom tokens are generated to suit specific needs. Each field of the following entities can be referenced as a custom token:

- Object types
- Request types and request header types
- Report types
- User data
- Workflow parameters

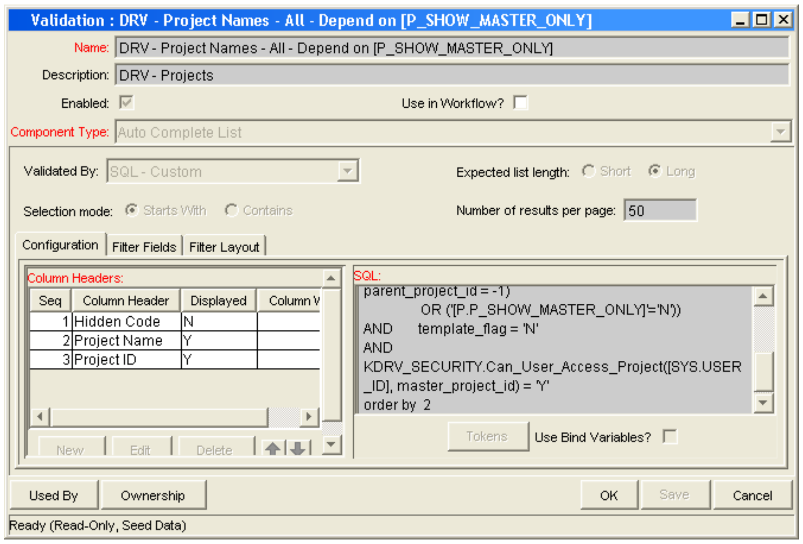
In addition, numerous standard tokens are available that provide other useful pieces of information related to the system. For example, Mercury IT Governance Center has a token that represents the users currently logged onto the system.

Where Tokens Are Used

Tokens can be used in the following entity windows:

- Object type commands
- Request type commands
- Validation commands and SQL statements

- Report type commands
- Executions and notifications for a workflow
- Workflow step commands
- Notifications in a report submission
- Special command commands
- Notifications for tasks
- Notes for request details


Validation : DRV - Project Names - All - Depend on [P_SHOW_MASTER_ONLY]

Name: DRV - Project Names - All - Depend on [P_SHOW_MASTER_ONLY]
Description: DRV - Projects
Enabled: [✓] Use in Workflow? [ ]
Component Type: Auto Complete List
Validated By: SQL - Custom
Expected list length: Short [✓ Long]
Selection mode: [✓ Starts With] Contains
Number of results per page: 50

Configuration
Filter Fields
Filter Layout

Column Headers:

Seq	Column Header	Displayed	Column V
1	Hidden Code	N	
2	Project Name	Y	
3	Project ID	Y	

SQL:

```
parent_project_id = -1)
          OR ('[P.P_SHOW_MASTER_ONLY]'='N'))
AND     template_flag = 'N'
AND
KDRV_SECURITY.Can_User_Access_Project([SYS.USER
_ID], master_project_id) = 'Y'
order by  2
```

Use Bind Variables? [ ]

Ready (Read-Only, Seed Data)

Figure 4-1. Example of a token used in a SQL statement

Token Builder Window Overview

In each of the entity windows listed in *Where Tokens Are Used* on page 50, a token can be created by opening the Token Builder window.

To open the Token Builder window through the Request Type window:

1. Open a Request Type window, either by generating a new request type or by opening an existing one.
2. Click the **Commands** tab.

3. Click **New Cmd.**

4. Click **Tokens.**

   The Token Builder window opens, as shown in *Figure 4-2*.

5. Use the Token Builder window to help construct valid tokens.

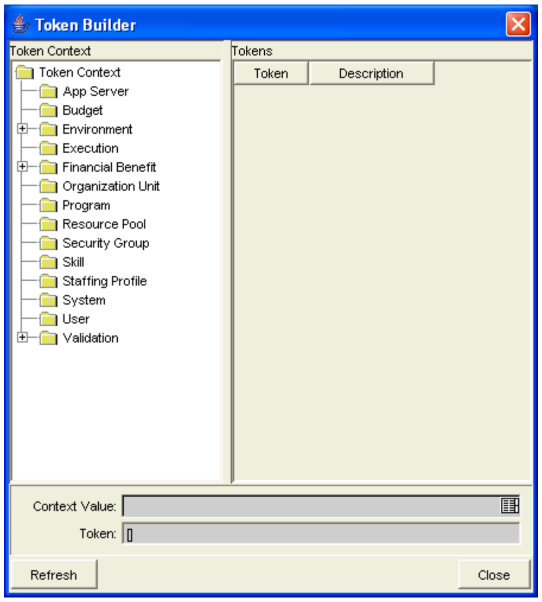


Figure 4-2. Token Builder window

Folders are displayed in the left pane of the Token Builder window. These folders contain groups of tokens that correspond to entities defined in Mercury IT Governance Center. For a list of entities and associated tokens, see *Tokens* on page 175. For instance, the Packages folder contains tokens that reference various package attributes. If the Packages folder is selected, the available package tokens are displayed in the list in the right pane of the window.

Some entities (folders) have sub-entities (sub-folders) that can be referenced by tokens. Click the plus sign (**+**) next to an entity to see the list of sub-entities for an entity. Each sub-entity also has tokens, and it is possible to reference any of the tokens of sub-entities, as well as tokens of the parent entity. For example, the package line entity is a sub-entity of the package entity.

As entity folders and the subsequent tokens in the list are selected, a character string is constructed in the Token field at the bottom of the Token Builder window. This is the formatted string used to reference the token. Either copy and paste the character string, or type this string where needed.

Token Formats

Tokens can use one of several different formats, depending on how they are going to be evaluated. Tokens can be expressed in the following formats:

- *Default Format*
- *Explicit Entity Format*
- *User Data Format*
- *Parameter Format*
- *Sub-Entity Format*
- *Environment and Environment Application Tokens* - the environment and environment app entities evaluate differently than the other entities.

Table 4-1 lists the entities and the formats each entity supports. Each format is discussed in a section following the table.

Table 4-1. Entities

Prefix (Entity)	Entity and Description	User Data Format?	Parameter Format?
AS	App server	N	N
BGT	Budget	Y	N
CON	Contact	Y	N
DEST_ENV	Destination environment. If an app code is specified, it will be used. Otherwise use only values from ENV.	Y	N
DEST_ENV.APP	Destination environment (for the environment application). Only use app code values, even if they're null.	Y	N
DEST_ENV.ENV	Destination environment. Ignores app codes and only uses the ENV values.	Y	N
DIST	Distribution	Y	N
ENV	Environment	Y	N
ENV.APP	Environment (for the environment application). Only use app code values, even if they're null.	Y	N
ENV.ENV	Environment. Ignores app codes and only uses the ENV values.	Y	N

Table 4-1. Entities

Prefix (Entity)	Entity and Description	User Data Format?	Parameter Format?
EXEC	Execution	N	N
NOTIF	Notification	N	N
ORG	Organization Unit	Y	N
PKG	Package	Y	N
PKG.PKGL	Package (package line)	Y	N
PKG.PEND	Package (pending package)	Y	N
PKGL	Package line	Y	Y
PRG	Program	Y	N
PRJ	Project plan	Y	N
PRJD	Project plan details	N	Y
REL	Release	N	N
REL.DIST	Release (distribution)	Y	N
REQ	Request	Y	Y
REQ.PEND	Request (pending)	N	N
REQD	Request details	N	Y
RP	Report submission	N	Y
RSCP	Resource pool	Y	N
SG	Security group	Y	N
SKL	Skill	Y	N
STFP	Staffing profile	Y	N
SOURCE_ENV	Source environment	Y	N
SOURCE_ENV.APP	Source environment (for environment application). Only use app code values, even if they're null.	Y	N
SOURCE_ENV.ENV	Source environment. Ignores app codes and only uses the ENV values.	Y	N
SYS	System	N	N
TSK	Task	Y	N
TSK.PEND	Task (pending)	N	N

Table 4-1. Entities

Prefix (Entity)	Entity and Description	User Data Format?	Parameter Format?
USR (User)	User	Y	N
VAL	Validation	N	N
VAL.VALUE	Validation (Value). Use this format to specify a specific validation.	Y	N
VALUE	Validation (Value)	Y	N
WF	Workflow	Y	N
WF.WFS	Workflow (step). Use this format to specify a specific workflow.	N	Y
WFS	Workflow step	Y	N

Default Format

Tokens are expressed as a prefix (a short name for the entity) followed by a token name. The prefix and token name are separated by a period and enclosed in square brackets with no spaces:

```
[PREFIX.TOKEN_NAME]
```

For example:

The token for the package number is expressed as:

```
[PKG.NUMBER]
```

The token for a request's workflow name is expressed as:

```
[REQ.WORKFLOW_NAME]
```

Certain tokens also support a sub-format. This sub-format is required for certain entities in order to evaluate to the correct context. For example, WF tokens will resolve to information related to the workflow, whereas WF.WFS tokens will resolve to workflow step information. Token sub-formats are included in the prefix, appended to the parent prefix, and separated by a period:

```
[PREFIX.SUB-PREFIX.TOKEN_NAME]
```

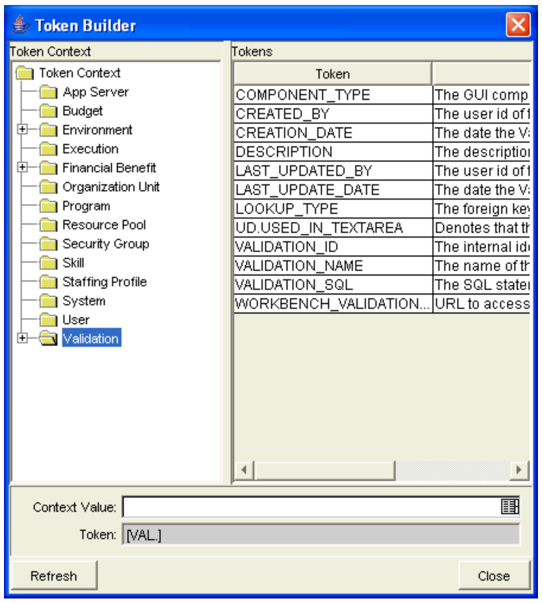
Tokens are evaluated according to the current context of Mercury IT Governance Center, which is derived based on information known at the time of evaluation. For more information, see *Token Evaluation* on page 66.

Explicit Entity Format

It is possible to provide a specific context value for an entity. This allows the default context to be overridden. Some tokens can never be evaluated in the default context. In these cases, the context must be set using an explicit entity format:

```
[PREFIX="<entity name>".TOKEN_NAME]
```

The Token Builder helps generate tokens in this format by providing a list of possible entity name values. When such a list is available, the Context Value auto-complete field at the bottom of the Token Builder becomes enabled. Like any other auto-complete field, either type into the field to reduce the list or click the auto-complete icon in the field to open the Validate window. Once a value is selected, it is inserted into the token in the Token field, generating an explicit entity token (see *Figure 4-3*).



Token	
COMPONENT_TYPE	The GUI comp
CREATED_BY	The user id of t
CREATION_DATE	The date the V:
DESCRIPTION	The descriptio
LAST_UPDATED_BY	The user id of t
LAST_UPDATE_DATE	The date the V:
LOOKUP_TYPE	The foreign key
UD.USED_IN_TEXTAREA	Denotes that th
VALIDATION_ID	The internal id
VALIDATION_NAME	The name of th
VALIDATION_SQL	The SQL staten
WORKBENCH_VALIDATION...	URL to access

Context Value:

Token: [VAL.]

Figure 4-3. Explicit Entity Format

For example, suppose the Email Address for the user "jsmith" is to be referenced. The token would be:

```
[USR="jsmith".EMAIL_ADDRESS]
```

To construct the above token in the Token Builder window:

1. Select the User folder.

Available tokens are displayed in the list on the right pane. The Context Value field at the bottom of the Token Builder is enabled. The string [USR.] appears in the Token field below the Context Value field.

2. Click the auto-complete icon in the Context Value field.

   A Validate window opens with a list of users.

3. Scroll through the list to find user "jsmith." Select this user and click **OK**.

   The string [USR="jsmith"] appears in the Token field.

4. In the list of tokens, select EMAIL_ADDRESS.

   The string [USR="jsmith".EMAIL_ADDRESS] appears in the Token field. This is the complete token. Since the token is "now complete, the Token field becomes enabled.

5. Select the token.

6. Press **Ctrl+c** on the keyboard to copy the token.

7. Press **Ctrl+v** on the keyboard to paste the token into another field.

Using Tokens within Other Tokens

The explicit entity format can be used to put tokens within other tokens to generate a value. For example, to print the description of the workflow that is associated with package #10203, the token would be:

```
[WF="[PKG="10203".WORKFLOW_NAME]".DESCRIPTION]
```

This token would have to be built in two steps. First, build the Description token for the workflow. Copy and paste that token into another field, then build the Workflow Name token for the package. Copy and paste that token within the Description token that was previously pasted.

Internally, this token is evaluated in two stages. The inner token is evaluated and the token has the following internal representation:

```
[WF="Workflow_Name".DESCRIPTION]
```

The remaining token is evaluated and the final result is printed:

```
description of my workflow
```

Table 4-2 includes a list of the tokens that support the explicit entity format.

Note

It is important to note that *entity_name* is case-sensitive and can contain spaces or other ASCII symbols.

Tokens for the user and security group entities can never be evaluated in the default format, and require the use of the explicit entity format. An example would be the token [USR.EMAIL_ADDRESS]. This token can never be evaluated because Mercury IT Governance Center cannot determine to which user it should refer.

Table 4-2. Tokens supporting explicit entity format

Token Prefix	Example	Acceptable Explicit Entry
BGT	[BGT=”Development Budget”.CREATED_BY]	Budget Name
CON	[CON=”Smith, John”.PHONE_NUMBER]	Last Name, First Name
ENV	[ENV=”ITG_SERVER".CLIENT_TRANSFER_ PROTOCOL]	Environment Name
ORG	[ORG="Project Managers".MANAGER_ID]	Organization Unit Name
PKG	[PKG="30010".CREATED_BY]	Package Number
REQ	[REQ="30006".CREATED_BY]	Request Number
RSCP	[RSCP="Development Resources".CREATED_ BY]	Resource Pool Name
SG	[SG=”Administrator”.LAST_UPDATED_BY]	Security Group Name
SKL	[SKL="Architect".AVERAGE_COST_RATE]	Skill Name
STFP	[STFP="ITG Pilot".CREATED_BY]	Staffing Profile Name
USR	[USR="jsmith".LAST_NAME]	User Name
VAL	[VAL="Date".CREATED_BY]	Validation Name
WF	[WF="Dev -> Test -> Prod".CREATED_BY]	Workflow Name
WF.WFS	[WF=“Workflow Name”.WFS=“1”.STEP_NAME]	Workflow Step Sequence Number

User Data Format

User data fields use tokens differently, as shown below:

```
[PREFIX.UD.USER_DATA_TOKEN]
```

The Prefix is the name of the entity that has user data. The modifier UD indicates that user data for that entity is being referenced. USER_DATA_TOKEN is the name of the token for the specific user data field. For example, suppose that a field for package user data has been generated whose token is GAP_NUMBER. In the default format, the token would be:

```
[PKG.UD.GAP_NUMBER]
```

In this context, PKG indicates that the package entity is being referenced, UD indicates that user data is being referenced, and GAP_NUMBER is the token name.

When user data fields are generated, a validation that has both a hidden and visible value can be used. For example, if the validation ‘KNTA - Usernames - All’ is used, the hidden value is the user ID and the displayed value is the username. The previous syntax references the hidden value only. To reference the visible value for a user data field, the syntax shown below must be used:

```
[PREFIX.VUD.USER_DATA_TOKEN]
```

If the modifier VUD is used instead of UD, the visible user data value is referenced.

Note

Drop-down lists and auto-complete lists may have different hidden and displayed values. For all other validations, the hidden and displayed values are identical.

When context can be determined, user data tokens are displayed with the system-defined tokens in the Token Builder.

Table 4-1 indicates which tokens support the user data format.

Parameter Format

Object type custom fields, request type custom fields, request header type fields, project plan fields, and workflow parameters use the parameter format for tokens as shown below:

```
[PREFIX.P.PARAMETER_TOKEN]
```

In this specific case, the Prefix is the name of the entity that uses a custom field. The modifier “P” indicates that parameters for that entity are being referenced. PARAMETER_TOKEN is the name of the token for the specific parameter field.

Note

- Package lines reference object type fields.
- Requests reference request type and request header type fields.
- Workflows reference workflow parameters.

For example, suppose a field for an object type named Gap Number (Token = GAP_NUMBER) has been generated that is used on package lines. In the default format the token would be:

```
[PKGL.P.GAP_NUMBER]
```

In this context, PKGL is the prefix since the package lines entity has been referenced, “P” indicates that parameters have been referenced, and GAP_ NUMBER is the token name.

Custom fields store both a hidden and visible value. For example, if the field uses the validation ‘KNTA - Usernames - All’, the hidden value is the user ID and the displayed value is the username. The previous syntax references the hidden value only. To reference the visible value for a parameter, use the syntax as shown:

```
[PREFIX.VP.PARAMETER_TOKEN]
```

If the modifier ‘VP’ is used instead of ‘P’, the visible parameter value is referenced.

Note

Drop-down lists and auto-complete lists may have different hidden and displayed values. For all other validations, the hidden and displayed values are identical.

Request Field Tokens

Tokens can access information on custom fields included on a request. These fields can be defined in a:

- Custom request type field
- Request header field (standard)
- Request header field (custom fields)
- Request header field (field groups)
- Table component field

Request Token Prefixes

All fields defined in the request header type (field group fields, custom header fields, and standard header fields) use the REQ prefix. The following examples could use “P” or “VP.”

```
REQ.<standard header Token>
```
Example: REQ.DEPARTMENT_CODE

```
REQ.P.<custom header field Token>
```
Example: REQ.P.BUSINESS_UNIT

```
REQ.P.<field group Token starting with KNTA_>
```
Example: REQ.P.KNTA_SKILL

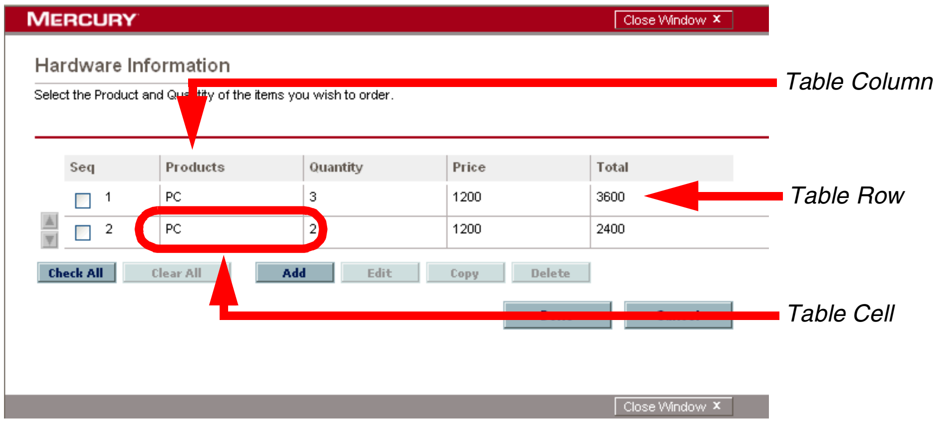
Fields defined in the request type use the REQD prefix. It is also possible to access standard header fields using the REQD prefix:

```
REQD.P.<custom detail field>
```

```
REQD.<standard header Token>
```

Tokens in Request Table Components

When referring to items in a table component, the tokens need to follow specific formats. These formats differ depending on the item that is being referenced within the table. *Figure 4-4* shows the basic elements of the table. These elements will be referenced when discussing the different options for referencing data within the table using tokens.



Hardware Information

Select the Product and Quantity of the items you wish to order.

	Seq	Products	Quantity	Price	Total
☐	1	PC	3	1200	3600
☐	2	PC	2	1200	2400

Figure 4-4. Table component formats

The format `[REQD.T.<TABLE_TOKEN>]` represents the table and specific tokens will be represented as `[REQD.T.<TABLE_TOKEN>.<SPECIFIC TOKENS>]`. The following sections provide examples of the formats used for tokens referencing items related to the table component:

- *To access the table row count from a Request context*
- *To access the Salary Column Total value from a Request context*
- *To access the Name of the first employee in the table from a Request*
- *To access the Code of the first employee in the table from a Request*
- *To access the Department Cell value of the current row (Table Row Context)*
- *To obtain a delimited list of a column's contents (Request Context)*

In these examples, the following example will be used. A table component named Employee with 4 columns:

- Name of Employee
- Years of Service of the Employee
- Department where the Employee belongs to
- Salary of the Employee.

These columns are defined as shown:

```
Table Component "Employee Table" with [EMPLOYEE] as the Token.
                    Column 1 - Name of Employee; Token = [NAME]
        Column 2 - Years of Service; Token = [YEARS_OF_SERVICE]
        Column 3 - Department of Employee; Token = [DEPARTMENT]
                Column 4 - Salary of Employee; Token = [SALARY]
```

To access the table row count from a Request context

```
[REQD.P.EMPLOYEE] - returns the raw row count without any
descriptive information.
```

```
[REQD.VP.EMPLOYEE] - returns the row count with descriptive
information. Example "13 Entry(s)".
```

```
WHERE: EMPLOYEE is the Token given to a table component type.
```

To access the Salary Column Total value from a Request context

```
[REQD.T.EMPLOYEE.TC.VP.SALARY.TOTAL]
```

```
WHERE: EMPLOYEE is the Token given to a table component type and
SALARY is the Token name given the table's first column.
```

To access the Name of the first employee in the table from a Request

```
[REQD.T.EMPLOYEE.TE="1".VP.NAME]
```

To access the Code of the first employee in the table from a Request

```
[REQD.T.EMPLOYEE.TE="1".P.NAME]
```

To access the Department Cell value of the current row (Table Row Context)

```
[TE.VP.DEPARTMENT]
```

It is possible to use this table component token in a Table Column Header validation SQL or in a table component rule SQL.

To obtain a delimited list of a column’s contents (Request Context)

```
[REQD.T.EMPLOYEE.TC.VP.NAME]
```

where `EMPLOYEE` is the token given to a table component type and `SALARY` is the token name given the table's first column.

This is particularly useful when a column is a list of user names, and this list can be used for sending these users notification.

Sub-Entity Format

Some entities have sub-entities that can be referenced. In the Token Builder, click the plus sign (+) next to an entity to see the list of its sub-entities. To reference a token from a sub-entity, in the context of a parent entity, use the syntax shown below:

```
[PREFIX.SUB_ENTITY_PREFIX.TOKEN]
```

In this case, the `PREFIX` is the name of the entity, the `SUB_ENTITY` prefix is the prefix for a sub-entity, and `TOKEN` is a token of the sub-entity. Typically, it is not necessary to use this syntax. However, it is possible to reference specific sub-entities using the explicit entity syntax.

For example, to reference the step name of the workflow step in the current context, both of the following tokens have the same meaning:

```
[WFS.STEP_NAME]
```

```
[WF.WFS.STEP_NAME]
```

However, to reference the step name of the first workflow step for the current workflow, use the following token:

```
[WF.WFS=”1”.STEP_NAME]
```

By not using the explicit entity format for the workflow entity, the token indicates that the workflow in the current context should be used. But by using the explicit entity format for the workflow step entity, the current context is overridden and a specific workflow step is referenced. In contrast, to reference the step name of the first workflow step in a workflow whose name is 'my workflow', use the following token:

```
[WF=“workflow_name”.WFS=“1”.STEP_NAME]
```

With this token, the current context for both the workflow and the workflow step will be overridden.

Environment and Environment Application Tokens

Tokens for the environments and environment application entities can have many different forms depending on the information to be referenced. During object type command execution, there is generally a source and a destination environment. The token prefixes SOURCE_ENV and DEST_ENV are used to reference the current source and destination, respectively, as shown in the following example:

```
[SOURCE_ENV.DB_USERNAME]
```

```
[DEST_ENV.SERVER_BASE_PATH]
```

In addition, a general ENV Prefix can be used in the explicit entity format to reference specific environments, as shown in the following example:

```
[ENV=”Prod”.CLIENT_USERNAME]
```

During normal environment token evaluation, the evaluation engine first evaluates the app code on the package line (if one is specified). If the corresponding app code token has a value, then the value is used. Otherwise, if no app code was specified or the app code token has no value, the corresponding base environment information is used.

To override the normal environment token evaluation and only evaluate the environment information (without first checking for the app code), construct the SOURCE_ENV and DEST_ENV tokens as shown in the following examples:

```
[SOURCE_ENV.ENV.DB_USERNAME]

[DEST_ENV.ENV.SERVER_BASE_PATH]

[ENV="Prod".ENV.CLIENT_USERNAME]
```

The evaluation engine can be instructed to look only at the app code information (without checking the base environment information if the app code token has no value). Construct the SOURCE_ENV and DEST_ENV tokens as shown in the following example:

```
[SOURCE_ENV.APP.DB_USERNAME]

[DEST_ENV.APP.SERVER_BASE_PATH]

[ENV="Prod".APP.CLIENT_USERNAME]
```

The prefix 'APP' can only be used in the sub-entity format. For example, the following token is invalid, since a context environment that includes the app code has not been specified.

```
[APP.SERVER_BASE_PATH]
```

In addition, the explicit entity format can be used with the app code entity to reference a specific app code, as shown in the following examples:

```
[SOURCE_ENV.APP="AR".DB_USERNAME]

[DEST_ENV.APP="OE".SERVER_BASE_PATH]

[ENV="Prod".APP="HR".CLIENT_USERNAME]
```

For example, suppose objects are being migrated on a package line at a given workflow step, and the line uses app code "HR". The workflow step has 'QA' as the source environment, and 'Prod' as the destination environment. *Table 4-3* shows other attributes of the environments and applications.

Table 4-3. Sample environment and app attributes

Environment	App Code	Server Base Paths
QA		/qa
QA	OE	/qa/oe
QA	HR	/qa/hr
Prod		/prod
Prod	OE	/prod/oe
Prod	HR	<no value>

Given this setup, *Table 4-4* shows some sample tokens and how each would evaluate.

Table 4-4. Sample environment tokens

Token	Evaluation
[SOURCE_ENV.SERVER_BASE_PATH]	/qa/hr
[DEST_ENV.SERVER_BASE_PATH]	/prod
[SOURCE_ENV.ENV.SERVER_BASE_PATH]	/qa
[DEST_ENV.ENV.SERVER_BASE_PATH]	/prod
[SOURCE_ENV.APP.SERVER_BASE_PATH]	/qa/hr
[DEST_ENV.APP.SERVER_BASE_PATH]	<no value>
[ENV=”QA”.APP=”OE”.SERVER_BASE_PATH]	/qa/oe

Token Evaluation

Tokens are evaluated at the point when Mercury IT Governance Center must know their context-specific values. At the time of evaluation, the token evaluation engine gathers information from the current context and tries to derive the value for the token. Values can only be derived for specific, known contexts (the current context is defined as the current package, package line, request, project plan, workflow step, or Source and destination environments).

The token evaluation engine takes as many passes as necessary to evaluate all tokens, so one token can be nested within another token. During each pass, if the evaluation engine finds a valid token, it replaces that token with its derived value. tokens that are invalid for any reason (such as the token is misspelled or no context is available) are left alone.

For example, suppose an object type command has the following Bourne-shell script segment as one of its command steps:

```
if [ ! -f [PKGL.P.P_SUB_PATH]/[PKGL.P.P_BASE_FILENAME].fmx ];
then exit 1; fi
```

At the time of execution, [PKGL.P.P_SUB_PATH] = “Forms” and [PKGL.P.P_BASE_FILENAME] = “obj_maint”. After token evaluation, this command step would reduce to:

```
if [ ! -f Forms/obj_maint.fmx ]; then exit 1; fi
```

As another example, suppose a user data field has been generated for all users called ‘MANAGER.’ The email address of the manager of the person who generated a request could be found using the token:

```
[USR=”[USR=”[REQ.CREATED_BY_NAME]”.VUD.MANAGER]”.EMAIL_ADDRESS]
```

The token evaluation engine would first evaluate the innermost token ([REQ.CREATED_BY_NAME]). Once that is complete, the next token ([USR=“<*name*>”.VUD.MANAGER]) is evaluated. Finally, the outermost token is evaluated, giving the manager's email address.

Tokens are evaluated at different points based on the token type. Tokens used in object type parameters and commands are evaluated during command execution. Tokens in a validation SQL statement are evaluated just before that statement is executed (such as generating a new package line). Tokens in an email notification are evaluated when a notification is generated.

Chapter

5

Working with Validations

In This Chapter:

- *Overview of Working with Validations*
- *What are Validations*
- *Validation Component Types - Overview*
- *Creating a Validation*
  - *User Data on the Validation Value*
- *Editing Validations*
  - *Creating a URL to Open the Validation Window*
- *Deleting Validations*
- *Static List Validations*
- *Dynamic List Validations*
  - *SQL Validation*
  - *Command Validation*
- *Configuring Auto-Complete Validations*
  - *Configuring General Auto-complete Behavior*
  - *Configuring the Auto-Complete Values*
- *Configuring Text Fields*
  - *Creating a Text Field Validation Overview*
  - *Available Text Data Masks*
  - *Customizing the System Text Data Masks*
  - *Creating a Custom Data Mask*
- *Using Directory and File Choosers*
  - *Directory Chooser*
  - *File Chooser*
- *Date Field Formats*
- *Creating 1800 Character Text Areas*

- *Configuring the Table Component*
    - *Defining the Table Component in the Validation Workbench*
    - *Adding the Table Component to a Request Type*
- *Package and Request Group Validations*
    - *Package and Request Groups*
    - *Request Type Category*
- *Validation Special Characters*
- *System Validations*

Overview of Working with Validations

This chapter provides an overview for how to use validations in your Mercury IT Governance system. Validations determine the acceptable input values for user-defined fields (such as object type or request type fields). Validations also determine the possible results that a workflow step can return.

What are Validations

Validations are used for two primary functions:

- **Fields.** Validations determine the field’s component type (text field, drop-down list, etc.) and the fields possible values. Fields can be created for a number of product entities: object types, request types, request header types, and user data.

- **Workflow step results.** Validations determine the possible results exiting a workflow step. For example, the validation WF - Standard Execution Results contains the possible execution step results of **Succeeded** or **Failed**.

Pre-seeded (system) validations are included with every product installation or upgrade. When configuring your system, you can select to use these system validations. If no validation exists that meets your specific requirements, you can create a new validation using the Validation Workbench. See *Creating a Validation* on page 73 for details.

Validation Component Types - Overview

The following table summarizes the available types of field components. Note that only certain component types can be used in a workflow step source's validation.

Table 5-1. Component Types

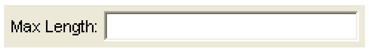
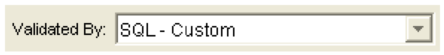
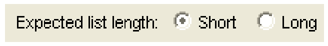
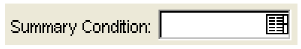
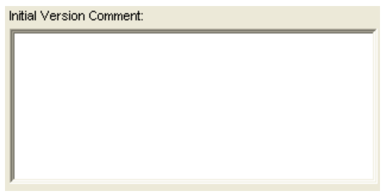
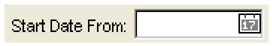
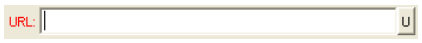
Component Type	Use In Workflow?	Example**	Description
Text Field	Yes	 Max Length:	Text entry fields displayed on a single line. Text fields can be configured to display the data according to a certain format. For example, you can configure a text field to accept and format a hyphenated nine-digit social security number or a ten digit telephone number.
Drop-down list	Yes	 Validated By: SQL - Custom	Field showing a column of choices.
Radio Button	No	 Expected list length: ● Short ○ Long	Field providing a Yes/No input.
Auto-complete list	Yes	 Summary Condition:	Field showing list of choices with multiple columns.
Text Area	No	 Initial Version Comment:	Text entry field that can span multiple lines.
Date Field	No	 Start Date From:	Supports a variety of date and time formats: long, medium, and short.
Web Address (URL)	No	 URL:	Text entry field for entering a URL. Pressing the U button opens a browser window to the specified web address.

Table 5-1. Component Types

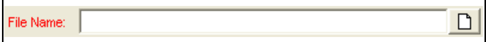
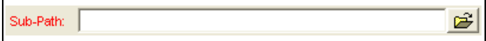
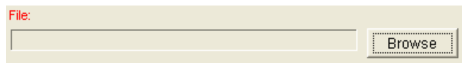
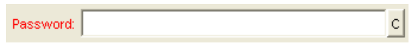
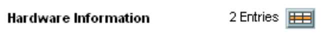
Component Type	Use In Workflow?	Example**	Description
File Chooser	No		Used only in object types. Requires that two fields be defined with the following tokens: P_FILE_LOCATION and P_SUB_PATH. See *Using Directory and File Choosers* on page 116 for configuration details.
Directory Chooser	No		Used only in object types. Requires that a parameter field be defined with the token P_FILE_LOCATION.
Attachment	No		Field for adding file attachments.
Password field	No		Field for capturing passwords.

Table 5-1. Component Types

Component Type	Use In Workflow?	Example**	Description
Table Component	No	 Hardware Information 2 Entries	Used to enter multiple records into a single component. The table component can be configured to include multiple columns of varied data types. Additionally, this component supports rules for populating elements within the table and provides functionality for capturing column totals. See *Configuring the Table Component* on page 121 for details. Fields of this component can only be added to request types, request header types and request user data.
Budget, Staffing Profile, Resource Pool	No	Budget: Team T Allocations	Field that can be added to the request type to enable access to view, edit or create budgets, staffing profiles, or resource pools associated with a request, project, or project plan. Fields of this component can only be added to a request type.

Creating a Validation

Generating certain workflow steps may require specific validations to ensure that business procedures are being followed. It is necessary to have both the Validation Editor and the Validation Values Editor access grants to add a new validation. See the *Security Model Guide and Reference* for a discussion of security groups and access grants.

To define a new validation:

1. Click **New Validation** on the Validation Workbench or select **File > New > Validation** from the menu.

   The Validation window opens.

2. Enter the name of the new validation in the Name field.
3. Enter a description of the new validation in the Description field.
4. Select whether the validation is enabled or not in the Enabled checkbox.
5. In the Use in Workflow checkbox, specify whether or not this validation can be used in a workflow step source.

   You can only use text field, drop-down list and auto-complete component types within workflow step sources.

6. Select the desired type of validation from the Component Type drop-down list.

   Choices are **Text Field, Drop Down List, Radio Buttons (Y/N), Auto Complete List, Text Area, Date Field, Web Address (URL), File Chooser, Directory Chooser, Password Field, Attachment, Table Component, Budget, Staffing Profile,** and **Resource Pool.** Selecting a value from this field will dynamically update the Validation window to display fields used to configure that type of validation.

7. Enter any additional information required for the component type selected.
8. Click **Ownership** to select which users will be able to edit, copy and delete this validation.
9. To save changes to the validation without closing the window, click **Save.** To save changes and close the window, click **OK.**

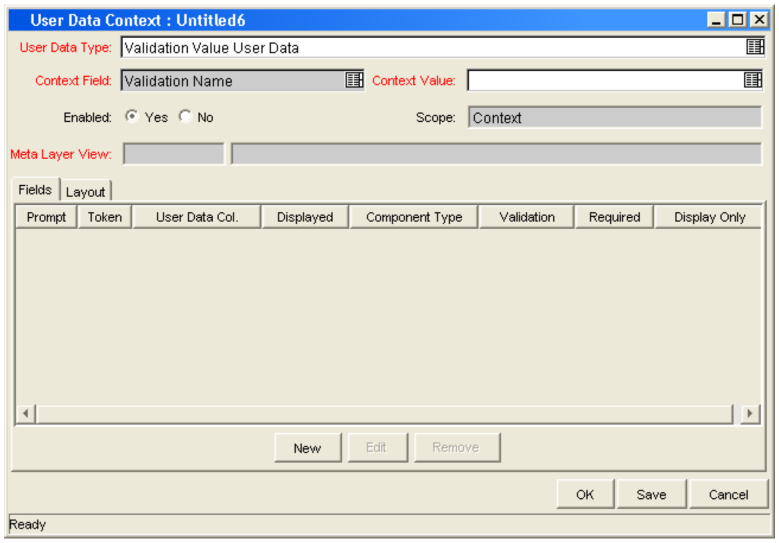
User Data on the Validation Value

You can enable the **User Data** tab to capture more information related to an individual validation value within a specific validation. For example, you can create a Description user data field that is associated with the Departments validation. When you add new values to the validation, you can click on the **User Data** tab and enter a description for that value.

The **User Data** tab can only be used when creating a drop-down or an auto-complete validated by a list.

To enable the **User Data** tab in the Edit Validation Value window:

1. Create the validation and note its name.
2. Open the User Data workbench.
3. Click **New User Data Context.**
4. Select **Validation Value User Data** from the User Data Type field.
5. Click **New** to create a user data field.



User Data Context : Untitled6

User Data Type: Validation Value User Data

Context Field: Validation Name

Context Value:

Enabled: (•) Yes ( ) No

Scope: Context

Meta Layer View:

Prompt	Token	User Data Col.	Displayed	Component Type	Validation	Required	Display Only

6. Save the settings in the User Data window.
7. On the Validation window, add or edit a validation value.

   The **User Data** tab is now enabled. You can select the tab and enter information in the newly defined user data field.

Editing Validations

You can open and edit validations using the Workbench. You should exercise caution when editing validations that are currently used by fields or workflow step sources. Both field and workflow step validations can be tied to workflow logic. Changing the validation values can invalidate a process.

For example, ACME changes the Priority field validation to include a new value **Very Easy**. ACME uses a deployment system workflow that has an Evaluate Priority step that routes the package based on the value in the Priority field (using a token execution type). ACME, however, did not update the workflow to enable a transition out of the step for the case when Priority = **Very Easy**. When a **Very Easy** package enters the Evaluate Priority step, it will get stuck.

The following restrictions apply to editing validations:

- User must have the following access grants:
  - Edit Validations
  - Edit Validation Values
- User must be a member of the ownership group for the validation.
- You cannot change which validation is associated with a workflow step source after a package has traversed that step. You can, however, still edit the values within that validation.

Creating a URL to Open the Validation Window

You can create a URL that opens a specific validation in the Workbench. This can provide a quick link to the configuration screen for a validation that is expected to change frequently. This URL can be included on your internal or external Web pages or a list of browser Favorites to provide convenient access to the validation’s definition.

Use the following URL format to access a specific Validation window:

```
http://host:port/kintana/servlet/
SmartURL?screen=VAL&pkname=<ValidationName>
```

Note

The following URL opens the Validation window for the validation named “Development Priorities.”

```
http://host:port/kintana/servlet/SmartURL?screen=VAL&pkname=
    Development+Priorities
```

Deleting Validations

Validations can be deleted from the Workbench. To delete a validation, you must be a member of the validation's ownership group and have the Edit Validations access grant.

A validation can not be deleted when:

- It is a system validation (a validation that is delivered with the product as seed data).
- It is being used by a workflow step source. Validations referenced by workflow step sources can only be disabled. A disabled validation continues to function in existing workflow steps, but can not be used when defining a new step source.
- It is being used by a field in a product entity (object type, request type, user data, report type, or project template field). Validations referenced by entity fields can only be disabled. A disabled validation continues to function in existing fields, but can not be used when defining a new field.

Note

Although you may not be able to delete a custom validation in all cases, you can disable it. This will allow the validation to be used in any active workflows or product entities, but will keep it from being used in any new workflow or entity definitions.

Static List Validations

You can create validations that provide a static list of options to the user. For example, ACME can create a validation for their engineering teams. They create a validation called Engineering Teams, consisting of the following values: **New Product Introduction**, **Product One**, and **Product Two**.

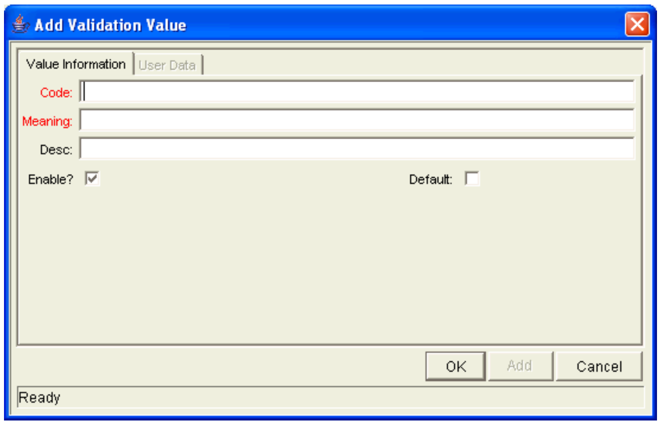
A static list validation can be a drop-down or an auto-complete list component.

To add values to the validation list:

1. In the Validation window, select **Drop Down List** or **Auto Complete List** from the Component Type field.
2. Select **List** from the Validated By field.

3. Click **New** and add a value.

   The Add Validation Window opens.



4. Enter information for the validation value as described in the following table.

Field	Definition
Code	The underlying code for the validation value. The code is the value stored in the database or passed to any internal functions, and is rarely displayed.
Meaning	The displayed meaning for the validation value in the drop-down list or auto-complete.
Default	The default value for the list. This value is initially displayed in drop-down lists (this is not used for auto-complete lists). There can be only one default value per list.

5. (Optional) Set the validation value as the default by checking the Default field.

   The default option is only available for drop-down lists.

6. Click **OK** to close the window and add the value to the validation. Click **Add** to add the value and keep the Add Validation Value open.

Validation values can be re-ordered using the up and down arrow buttons. The sequence of the validation values determines the order that the values are displayed in the list.

Note

You can copy existing values defined in other validations using the **Copy From** button. Click **Copy From** and query an existing list-validated validation and choose any of the validation values. Click **Add** or **OK** in the Copy From window and the selected value or values are added to the list.

Note

Be careful when creating validations (drop-down lists and auto-complete lists) that are validated by lists. Each time the set of values changes, you will be forced to update the validation. Consider, instead, validating using a SQL query or PL/SQL function to obtain the values from a database table.

Dynamic List Validations

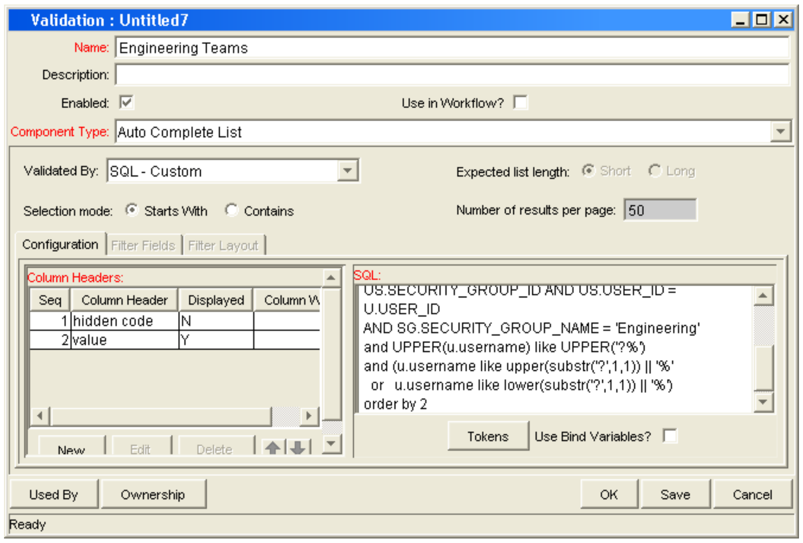
You can create validations that provide a dynamic list to the user. This is often a better approach than defining static list validations. Each time a static list validation needs to be updated, a manual update has to occur. Dynamic list validations can often be constructed in such a way as to automatically pick up and display the altered values.

For example, ACME needs a field validation that will list all users who are on their Support Team. They could construct a validation that is validated by a list of users, but any time the Support Team changed (members join or leave the department) the list would have to be manually updated. ACME decides instead to create a dynamic list validation. They create an auto-complete list validation that is validated by a SQL statement. The SQL statement returns all users who are a member of the **Support Team** security group. When the security group membership is altered, the validation is automatically updated with the correct values.

A dynamic list validation can be created using a drop-down or an auto-complete list component.

SQL Validation

You can use a SQL statement to generate the values in a validation. SQL can be used as a validation method for drop-down lists and auto-complete lists. To define a dynamic list of choices, set a drop-down list or auto-complete list to Validated By - **SQL**. Then in the SQL area, enter the Select statement that queries the necessary database.


Validation : Untitled7

Name: Engineering Teams

Description:

Enabled: ✓ Use in Workflow? ☐

Component Type: Auto Complete List

Validated By: SQL - Custom

Expected list length: [✓ Short] Long

Selection mode: [✓ Starts With] Contains

Number of results per page: 50

Configuration | Filter Fields | Filter Layout

Column Headers:

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	

New Edit Delete

SQL:

```
US.SECURITY_GROUP_ID AND US.USER_ID =
U.USER_ID
AND SG.SECURITY_GROUP_NAME = 'Engineering'
and UPPER(u.username) like UPPER('?%')
and (u.username like upper(substr('?',1,1)) || '%'
  or   u.username like lower(substr('?',1,1)) || '%')
order by 2
```

Tokens Use Bind Variables? ☐

Used By Ownership OK Save Cancel

Ready

If an auto-complete list is being used, you can define headers for the selected columns. These column headers are used in the window that opens when a value from an auto-complete list is selected. Click **New** under Column Headers. *Table 5-2* shows the fields that can be entered for a column header. If a column header is not defined for each column in a SQL query, a default name is used.

Table 5-2. Column Headers

Field	Definition
Column Header	The name of the column that is displayed in the auto-complete window.
Display	Determines whether or not the column is displayed. The first column is never displayed and the second column is always displayed.

For example, ACME, Inc., creates an auto-complete field that lists all users in the “Engineering” department. They choose to validate the list by SQL.

```
SELECT U.USER_ID, U.USERNAME, U.FIRST_NAME, U.LAST_NAME
FROM KNTA_USERS U, KNTA_SECURITY_GROUPS SG, KNTA_USER_SECURITY
US
WHERE SG.SECURITY_GROUP_ID = US.SECURITY_GROUP_ID AND US.USER_
ID = U.USER_ID
AND SG.SECURITY_GROUP_NAME = 'Engineering'
and UPPER(u.username) like UPPER('?%')
and (u.username like upper(substr('?',1,1)) || '%'
   or   u.username like lower(substr('?',1,1)) || '%')
order by 2
```

When a new user account is created and is added to the “Engineering” security group, that user will automatically be included in the auto-complete list.

Note

A validation may already exist that meets your process requirements. If it does, consider using that validation in your process. Also consider copying and modifying validations that are similar to the desired validation. See *System Validations* on page 136 for a complete list of validations that are delivered with the product.

SQL Validation Tips

The following guidelines are helpful when writing a SQL statement for a SQL-validated validation:

- The SQL statement must query at least two columns. The first column is a hidden value which is never displayed, and is often stored in the database or passed to internal functions. The second column is the value that is displayed in the field. All other columns are for information purposes and are only displayed in the auto-complete window. Extra columns are not displayed for drop-down lists.

- When something is typed into an auto-complete list field, the values in the auto-complete window that appear are constrained by what was first typed in the field. Generally, the constraint is case insensitive. This is accomplished by writing the SQL statement to query only values that match what was typed.

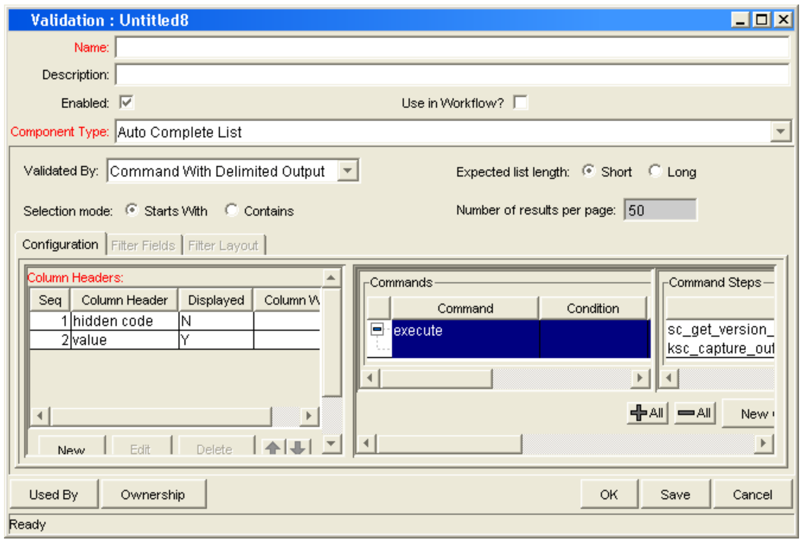
  Before the auto-complete window is displayed, all question marks in the SQL statement are replaced by the text that the user typed. In general, if the following conditions are added to the WHERE clause in a SQL statement, the values in the auto-complete window are constrained by what the user typed.

  ```
  where UPPER(<displayed_column>) like UPPER('?%')
  and (<displayed_column> like upper(substr('?',1,1)) || '%'
  or   <displayed_column> like lower(substr('?',1,1)) || '%')
  ```

  Any column aliases included directly in the SQL statement are not used. The names of the columns, as displayed in auto-complete lists, are determined from the Column Headers. Drop-down lists do not have column headers.

Command Validation

An auto-complete list can contain command line executions that return and display a list of values. To define a dynamic list of choices, set an auto-complete list to Validated By - **Command with Delimited Output** or **Command with Fixed Width Output.** Then enter commands the Commands area. See *Configuring the Auto-Complete Values* on page 94 for detailed instructions.



Validation : Untitled8

Name:
Description:
Enabled: ✓ Use in Workflow? ☐
Component Type: Auto Complete List

Validated By: Command With Delimited Output Expected list length: [✓ Short] Long
Selection mode: [✓ Starts With] Contains Number of results per page: 50

Column Headers:

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	

Commands

Command	Condition
execute	

Command Steps

sc_get_version_
ksc_capture_out

Figure 5-1. Auto-complete using command validation

Configuring Auto-Complete Validations

Auto-complete fields are used throughout the Mercury IT Governance Center to provide users with an efficient way to select field values from a set of valid choices. Configuring auto-complete fields consists of two activities:

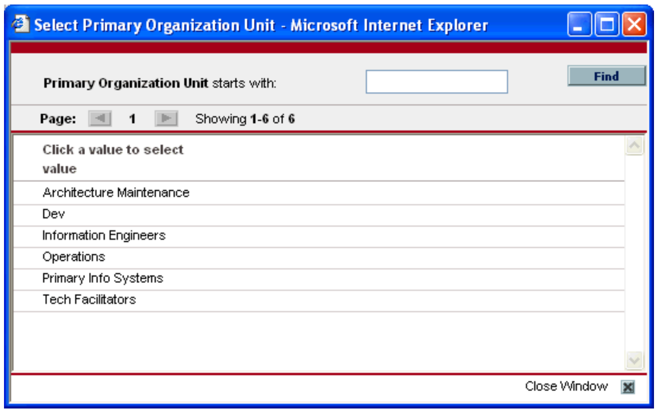
- Specifying general auto-complete behavior
- Configuring the validation values

Configuring General Auto-complete Behavior

Auto-complete fields can be used for validations with a small or large number of choices. The auto-complete can be configured to behave differently depending on the expected number of values. For example, if you expect a large number of entries, the auto-complete window will include an interface that allows you to page through your results. Additionally, you can configure how the “auto-complete” feature of the field behaves. For example, you can configure the auto-complete field to automatically complete entries that either start with or contain a text string.

Configuring Short List Auto-Complete Fields

Auto-complete fields configured to display a short list of entries, displaying all of the values on a single page. *Figure 5-2* shows the Select window for a short list auto-complete field.



Select Primary Organization Unit - Microsoft Internet Explorer

Primary Organization Unit starts with:

Find

Page: 1 Showing 1-6 of 6

Click a value to select
value

Architecture Maintenance
Dev
Information Engineers
Operations
Primary Info Systems
Tech Facilitators

Close Window

Figure 5-2. Short list auto-complete

Note

Auto-completes configured as short lists will load all values when the window is opened. This can lead to a slower load time and an unfavorable user experience. For fields with many possible values, consider formatting the auto-complete using the long list format.

To configure a short list auto-complete field:

1. Create a new validation or open an existing validation.

   The Validation window opens.

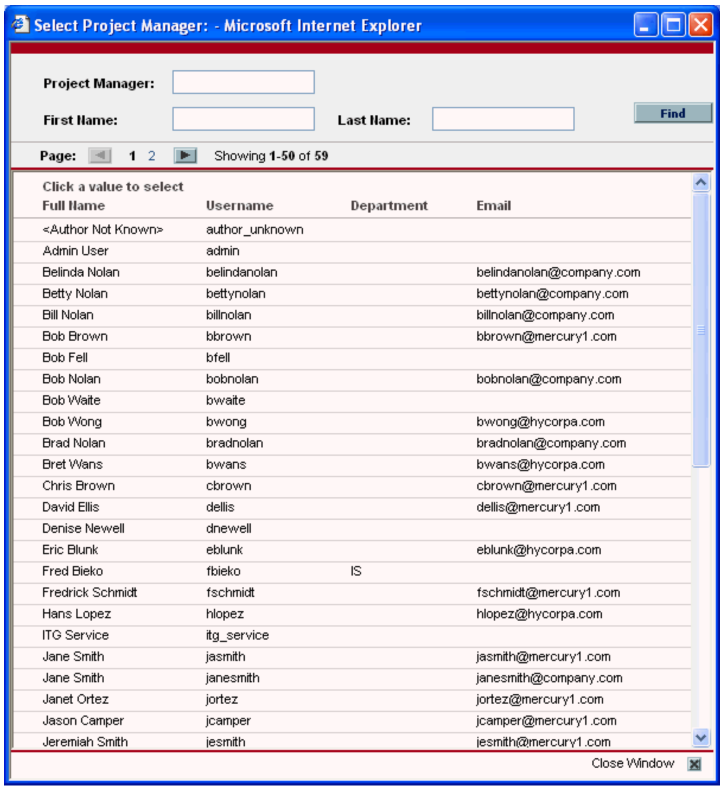
2. From the Component Type field, select **Auto Complete List.**

3. In the Expected list length field, select **Short.**

4. Click **Save.**

Configuring Long List Auto-Complete Fields

Auto-complete fields configured to display a long list of entries, dividing the results between multiple pages. By default, 50 results are shown per page. End users can page through the results or further limit the results by specifying text in one of the available filter fields at the top of the page. *Figure 5-3* shows the Select window for a long list auto-complete field.



Full Name	Username	Department	Email
<Author Not Known>	author_unknown		
Admin User	admin		
Belinda Nolan	belindanolan		belindanolan@company.com
Betty Nolan	bettynolan		bettynolan@company.com
Bill Nolan	billnolan		billnolan@company.com
Bob Brown	bbrown		bbrown@mercury1.com
Bob Fell	bfell		
Bob Nolan	bobnolan		bobnolan@company.com
Bob Waite	bwaite		
Bob Wong	bwong		bwong@hycorpa.com
Brad Nolan	bradnolan		bradnolan@company.com
Bret Wans	bwans		bwans@hycorpa.com
Chris Brown	cbrown		cbrown@mercury1.com
David Ellis	dellis		dellis@mercury1.com
Denise Newell	dnewell		
Eric Blunk	eblunk		eblunk@hycorpa.com
Fred Bieko	fbieko	IS	
Fredrick Schmidt	fschmidt		fschmidt@mercury1.com
Hans Lopez	hlopez		hlopez@hycorpa.com
ITG Service	itg_service		
Jane Smith	jasmith		jasmith@mercury1.com
Jane Smith	janesmith		janesmith@company.com
Janet Ortez	jortez		jortez@mercury1.com
Jason Camper	jcamper		jcamper@mercury1.com
Jeremiah Smith	jesmith		jesmith@mercury1.com

Figure 5-3. Long list auto-complete

Note

Auto-completes configured as long lists only load a limited set of values when the window is opened. For extremely long lists or lists that are at risk of loading slowly (for example the values are obtained from an alternate database), consider using the long list format.

To configure a long list auto-complete field:

1. Create a new validation or open an existing validation.

   The Validation window opens.

2. From the Component Type field, select **Auto Complete List.**

3. In the Expected list length field, select **Long.**

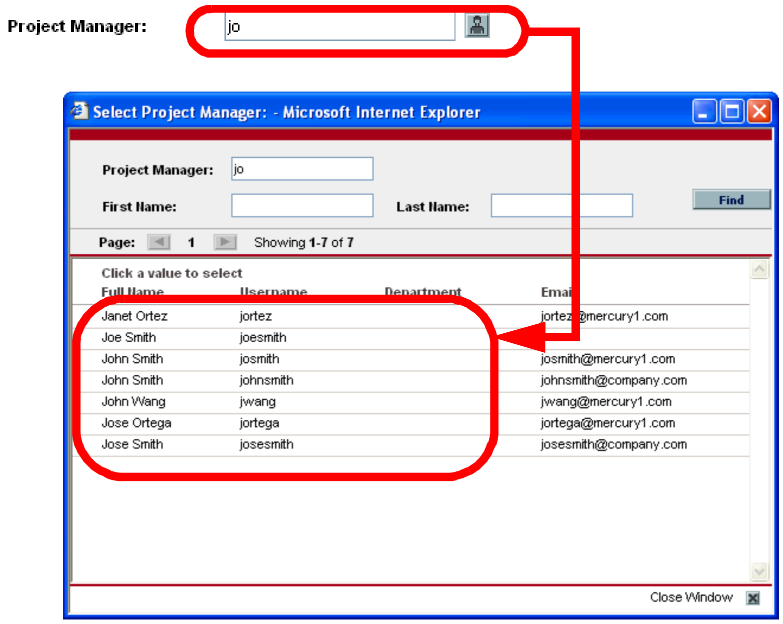
4. Click **Save.**

Note

All auto-completes that are validated by **SQL - User** are required to use the long list auto-complete format. This selection is automatically defaulted when the user selects **SQL - User** from the Validated By field on the Validation window.

Configuring the Automatic Value Matching and the Interactive Select Page

This section provides instructions for configuring auto-complete fields to filter a list of possible values based on a matching character string. It also provides instructions for configuring the automatic value-limiting that occurs on the auto-complete's Select page. *Figure 5-4* shows an auto-complete field that has opened to display matching values.



Project Manager: jo

Project Manager: jo

First Name:

Last Name:

Page: 1 Showing 1-7 of 7

Click a value to select

Full Name	Username	Department	Email
Janet Ortez	jortez		jortez@mercury1.com
Joe Smith	joesmith		
John Smith	josmith		josmith@mercury1.com
John Smith	johnsmith		johnsmith@company.com
John Wang	jwang		jwang@mercury1.com
Jose Ortega	jortega		jortega@mercury1.com
Jose Smith	josesmith		josesmith@company.com

Figure 5-4. Auto-complete field and matching values in the Select page

Functional Overview: Matching for "Starts with" or "Contains"

Auto-complete field behavior can be divided into the following areas:

- **Field behavior.** A user types a character in the field and presses the **Tab** key. If an exact match is not available, the Select page opens.

- **The** Select **page behavior.** For lists that are configured appropriately, when a user types a character or characters into the field at the top of the page, the results are automatically limited to display only matching entries.

For both the field and Select page behaviors, automatic value matching can be based on either “starts with” character matching or “contains” character matching. The following table summarizes this behavior:

Table 5-3. Automatic character matching field behavior

Character matching mode	Description of Behavior
Starts with	Type characters and press tab. The selection window will open and list entries that begin with the specified characters.
Contains	Type characters and press tab. The selection window will open and list entries that contain the specified character string. This is the same behavior as a wild card search, which uses the % character at the beginning of the search text.

Table 5-4. Automatic character matching Select page behavior

Character matching selection mode	Description of Behavior
Starts with	Type characters and the list will automatically be filtered for entries that begin with the specified characters.
Contains	Type characters and the list will automatically be filtered for entries that contain the character string.

Configuration Instructions

The field and the Select page behavior are configured distinctly in the Validation window for the specific auto-complete list. This section provides instructions for configuring the “starts with” and “contains” functionality in the field and Select page, as described in *Functional Overview: Matching for “Starts with” or “Contains”* on page 85.

To configure “starts with” matching from the auto-complete window to the selection window, add the following to the SQL WHERE clause:

```
UPPER(value) like UPPER('?%') and (value like
upper(substr('?',1,1)) || '%' or value like
lower(substr('?',1,1)) || '%')
```

To configure "contains" matching from the auto-complete window to the selection window, add the following to the SQL WHERE clause:

```
UPPER(value) like UPPER('%?%') and (value like '%' ||
upper(substr('?',1,1)) || '%' or value like '%' ||
lower(substr('?',1,1)) || '%')
-
```

To configure “starts with” matching within the interactive selection window:

1. Open the auto-complete’s Validation window.
2. From the Expected list length field, select **Short.**

   This feature is only available for short lists.
3. From the Selection mode radio button, select **Starts With.**
4. Save the validation.

Note: This setting only controls the matching in the Select page. Matching in the auto-complete field is controlled by including specific clauses in the auto-complete's SQL. See above for details.

To configure “contains” matching within the interactive selection window:

1. Open the auto-complete’s Validation window.
2. From the Expected list length field, select **Short.**

   This feature is only available for short lists.
3. From the Selection mode radio button, select **Contains.**
4. Save the validation.

Note: This setting only controls the matching in the Select page. Matching in the auto-complete field is controlled by including specific clauses in the auto-complete's SQL. See above for details.

Configuration Tips

Consider the following tips when configuring the “starts with” versus “contains” functionality for auto-complete fields and the Select page.

- Auto-completes should be configured such that the field matching behavior works the same way as the Select page matching behavior. Specifically, if the auto-complete field uses the “starts with” clauses in the SQL, then the selection window should use the “Starts With” Selection Mode. See *Configuration Instructions* on page 86 for details.
- Consider using the “Contains” Selection Mode for fields with multi-word values. For example, consider the possible values for the request type auto-complete field:

```
Development Bug
Development Enhancement
Development Issue
Development Change Request
IS Bug
IS Enhancement
IS Issue
IS Change Request
Support Issue
Support Change Request
```

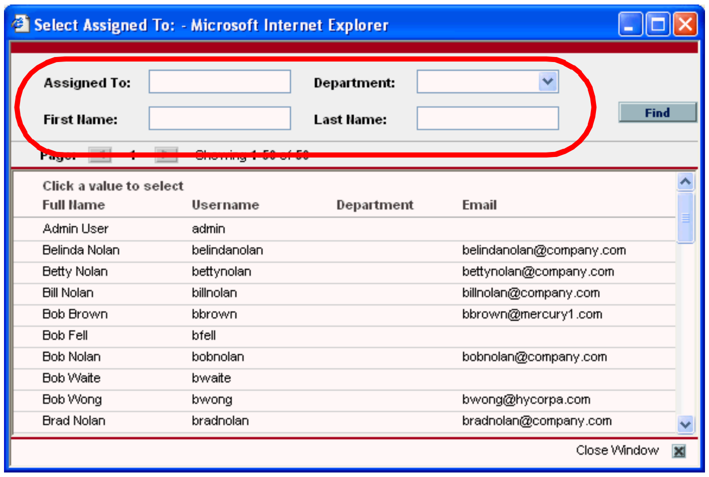
  Using “contains” can be useful here. The user knows that he needs to log a bug against one of the IS-supported Financial applications. The user types "bug" into the auto-complete field and presses the **Tab** key. The following items are returned:

```
   Development Bug
   IS Bug
```

  The user selects “IS Bug.” Without the “contains” feature enabled, typing “bug” would have returned the entire list. He might have also typed “Financial,” thinking that there might be a separate request type used for each type of supported application. This, too, would have returned the entire list. At that point, the user would be forced to try another “starts with” phrase or simply read the entire (potentially long) list.

Adding Search Fields to the Auto-Complete Window

Auto-completes with a long list of values can be configured to display additional filter fields in the Select window. These fields can be used to search other properties than the primary values in the list. Users can enter values in the filter fields and click **Find** to display only the values that match the search criteria. *Figure 5-3* shows the Select window with additional filter fields.



Assigned To:
Department:
First Name:
Last Name:

Page: 1 Showing 1-50 of 50

Click a value to select

Full Name	Username	Department	Email
Admin User	admin		
Belinda Nolan	belindanolan		belindanolan@company.com
Betty Nolan	bettynolan		bettynolan@company.com
Bill Nolan	billnolan		billnolan@company.com
Bob Brown	bbrown		bbrown@mercury1.com
Bob Fell	bfell		
Bob Nolan	bobnolan		bobnolan@company.com
Bob Waite	bwaite		
Bob Wong	bwong		bwong@hycorpa.com
Brad Nolan	bradnolan		bradnolan@company.com

Figure 5-5. Filter fields in the auto-complete select window

Note

Filter fields can not be configured when validating your list by **List**, **Command With Delimited Output,** or **Command With Fixed Width Output**.

To add a filter field to the auto-complete validation:

1. Open the validation for the auto-complete.

   Auto-complete validations must display **Auto Complete List** in the Component Type field.

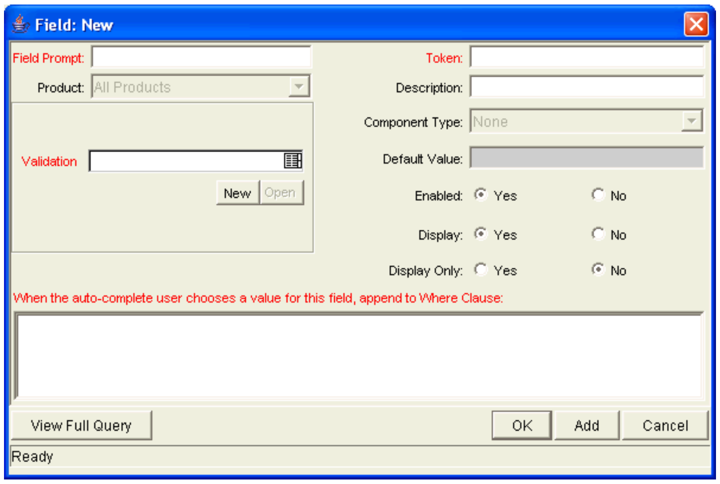
2. In the Expected list length field, select **Long.**

   Only long formatted auto-complete lists can include filter fields.

3. Click the **Filter Fields** tab.

4. Click **New.**

   The Field: New window opens.



5. Enter the required information.

   Table 5-5 defines all of the fields on this window.

Table 5-5. Fields in the Fields:New window

Field	Description
Field Prompt	The name that is displayed for the field in the auto-complete Select window.
Product	The Mercury IT Governance Center product the field is used by.
Validation	The validation for the filter field. You can select any type of validation, except for auto-complete type validations. The values accepted by this validation will be appended to the WHERE clause in the SQL query that determines the ultimate auto-complete list display.
New	Opens the Validation window where you can construct a new validation for the filter field. Note that you can not use an auto-complete type validation for the filter field.
Open	Opens the Validation window and displays the definition of the validation specified in the Validation field.
Token	The token for the field value. The token value will be appended to the WHERE clause in the SQL query that determines the ultimate auto-complete list display.
Description	The description of the filter field.

Table 5-5. Fields in the Fields:New window [continued]

Field	Description
Component Type	The component type for the filter field, determined by its validation.
Default Value	The default value for the filter field, determined by its validation.
Enabled	Determines whether the filter field is enabled.
Display	Determines whether the filter field is visible to the user in the auto-complete's Select window.
Display Only	Determines whether the filter field is updatable. When Display Only is set to **Yes**, the field can not be updated.
When the auto-complete user chooses a value for this field, append to WHERE clause:	The AND clause that is appended to the portlet's WHERE clause if the user enters a value in this filter field. Each filter field will append its term to the portlet query when a value is entered by the end user in the Select window. For example, if the filter field uses the CRT-Priority-Enabled validation and a filter field token of P_PRIORITY, enter the following into this field: `AND R.PRIORITY_CODE = '[P.P_PRIORITY]'` Note: The value in this field must start with "AND".
View Full Query	Opens a window showing the full query.

6. Click **OK.**

Note

Filter fields can offer a powerful method for enabling users to efficiently locate specific values in large lists. When adding filter fields to an auto-complete validation, consider the following tips:

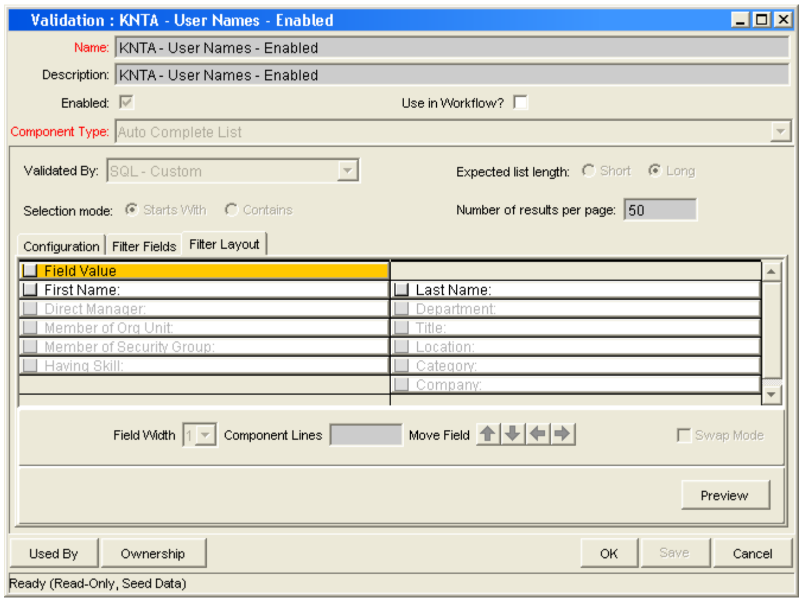
- Ensure that the filter fields are functionally related to the list of values. For example, a validation that provides a list of request types can include a filter field for a specific Department associated with the request types.
- Consider reusing (copying) an auto-complete validation and modifying the filter fields to display a subset of the entire list. Using the Displayed, Display Only, and Default fields in the Filter Field window, you can configure the auto-complete values to automatically limit the results.
- Performance can degrade if joining tables over database links.
- Only use this functionality for complex fields.

To modify the filter field layout:

1. Open the auto-complete validation that includes filter fields on the **Filter Fields** tab.

2. Click the **Filter Layout** tab.

   The tab lists the primary field and all of the filter fields that have been defined for the auto-complete. The primary field is named Field Value. This is the field that holds the eventual selected value.



Validation : KNTA - User Names - Enabled

Name: KNTA - User Names - Enabled
Description: KNTA - User Names - Enabled
Enabled: ✓
Use in Workflow?
Component Type: Auto Complete List
Validated By: SQL - Custom
Expected list length: Short [✓ Long]
Selection mode: [✓ Starts With] Contains
Number of results per page: 50

Field Value	
First Name:	Last Name:
Direct Manager:	Department:
Member of Org Unit:	Title:
Member of Security Group:	Location:
Having Skill:	Category:
	Company:

Ready (Read-Only, Seed Data)

3. Select the field that you would like to move.

   To select more than one field, use the **Shift** key while selecting a range. It is only possible to select a continuous set of fields (the **Ctrl+select** functionality is not supported).

4. Use the arrow buttons to move the fields to the desired location in the layout builder.

Note

A field or a set of fields cannot be moved to an area where other fields already exist. The other field(s) must be moved out of the way first.

5. To switch the positions of two fields:

a. Select the first field and check the Swap Mode checkbox.

   An “S” appears in the checkbox area of the selected field.

b. Double-click the second field that you want to switch positions with the first.

   This causes the two fields to change positions. Following the switch, the Swap Mode checkbox is turned off. To swap another set of fields, repeat this procedure.

6. To check what the layout looks like in actual use, click **Preview.**

   This opens a small window that shows the fields as they will appear. It is important to note that:

   - Any rows with no fields are ignored. They do not show up as a blank line.
   - Any non-displayed fields do not affect the layout. They are considered the same as a blank field.

Special Case: Configuring an Auto-Complete List of Users

User auto-completes or validations (Validated by: SQL-User) have three filter fields by default:

- Primary field—this field takes the name of the auto-complete field
- First name
- Last name

The user auto-complete always appears in the long list format, which uses the paging interface to display the items. Additionally, user auto-completes display a different icon, pictured in *Figure 5-6*, in the auto-complete field.

Figure 5-6. User icon

To configure a user auto-complete validation:

1. Create a new validation.

   The Validation window opens.

2. From the Component Type field, select **Auto Complete List.**

3. From the Validated By field, select **SQL - User.**

4. Configure the SQL query that will determine the users listed in the validation.

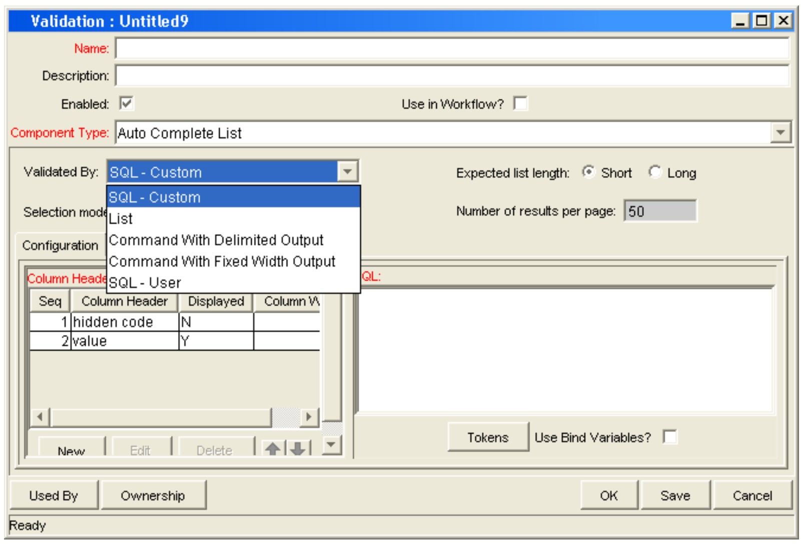
   See *Configuring the Auto-Complete Values* on page 94 for details.

5. Click **Save.**

Configuring the Auto-Complete Values

The values in an auto-complete list can be specified in the following ways. In the Validate By field, select one of the following:

- **List**: Used to enter specific values.
- **SQL**: Uses a SQL statement to build the contents of the list.
- **SQL - User**: Identical to SQL configuration, but includes a few additional preconfigured filter fields.
- **Command With Delimited Output**: Uses a system command to produce a character-delimited text string and uses the results to define the list.
- **Command With Fixed Width Output**: uses a system command to produce a text file and parses the result on the basis of the width of columns, as well as the headers.



Validation : Untitled9

Name:	
Description:	
Enabled:	✓
Use in Workflow?	☐
Component Type:	Auto Complete List
Validated By:	SQL - Custom
Expected list length:	[✓ Short] Long
Number of results per page:	50
Use Bind Variables?	☐

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	

Figure 5-7. Auto-Complete List

For more information on creating auto-completes validated by List or SQL, refer to the following sections:

- *Static List Validations* on page 77
- *Dynamic List Validations* on page 79

Validation by Command With Delimited Output

Validations that are validated by commands with delimited output can be used to get data from an alternate source, and use that data to populate an auto-complete field. This functionality provides additional flexibility when designing auto-complete lists.

Many enterprises need to use alternate sources of data within their applications. Examples of these sources are a flat file, an alternate database source, or output from a command line execution. Special commands may be used in conjunction with these alternate data sources, in the context of a validation, to provide a list of values.

To configure a validation by command with delimited output:

1. In the Validation Workbench, under Validated By, choose **Command With Delimited Output** and input the delimiting character.

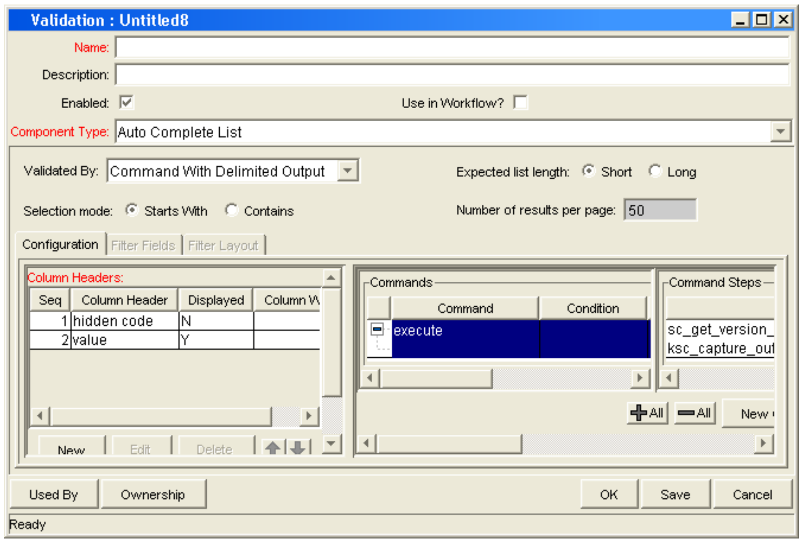
2. Under New Command, enter in the command steps to be executed.

   These can include Mercury IT Governance special commands. Your commands should include the special command `ksc_capture_output`, which captures and parses the delimited command output. If the `ksc_capture_output` special command is surrounded by the `ksc_connect` and `ksc_disconnect` commands, the command will be run on the remote system. Otherwise, the command will be run locally on the Mercury IT Governance server (similar to `ksc_local_exec`).

   The simple example below uses a comma for a delimiter and has the validation values red, blue and green. The script places the validations into the newfile.txt file, and then uses the special command ksc_capture_output to process the text of the file.

```
ksc_begin_script [AS.PKG_TRANSFER_PATH]newfile.txt
red,red
blue,blue
green,green
ksc_end_script
ksc_capture_output cat [AS.PKG_TRANSFER_PATH]newfile.txt
```

Table 5-6 shows the Validation window for Command with Delimited Output.



Validation : Untitled8

Name:

Description:

Enabled: ☑ Use in Workflow? ☐

Component Type: Auto Complete List

Validated By: Command With Delimited Output

Expected list length: [✓ Short] Long

Selection mode: [✓ Starts With] Contains

Number of results per page: 50

Configuration | Filter Fields | Filter Layout

Column Headers:

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	

Commands

Command	Condition
execute	

Command Steps

sc_get_version_
ksc_capture_out

Figure 5-8. Validation by command with delimited output

Table 5-6. Validation by command with delimited output

Field	Definition
Command Panel	Panel where new commands can be added to capture validation values.
Data Delimiter	Indicates the character or key by which the file will be separated into the validation columns.

Headers can also be defined for the columns selected. These column headers are used in the window that opens when a value is selected from an auto-complete list. To define a new header, click **New** under Column Header. *Table 5-7* shows the fields that can be entered for a column header. If a column header is not defined for each column in a command, a default name is used.

Table 5-7. Column headers

Field	Definition
Column Header	The name of the column that is displayed in the auto-complete window.
Display	Determines whether or not the header is displayed in the validation.

Validation by Command With Fixed Width Output

Validations by Command with Fixed Width Output can be used to obtain data from an alternate source, and use that data to populate an auto-complete field. This functionality provides additional flexibility when designing auto-complete lists.

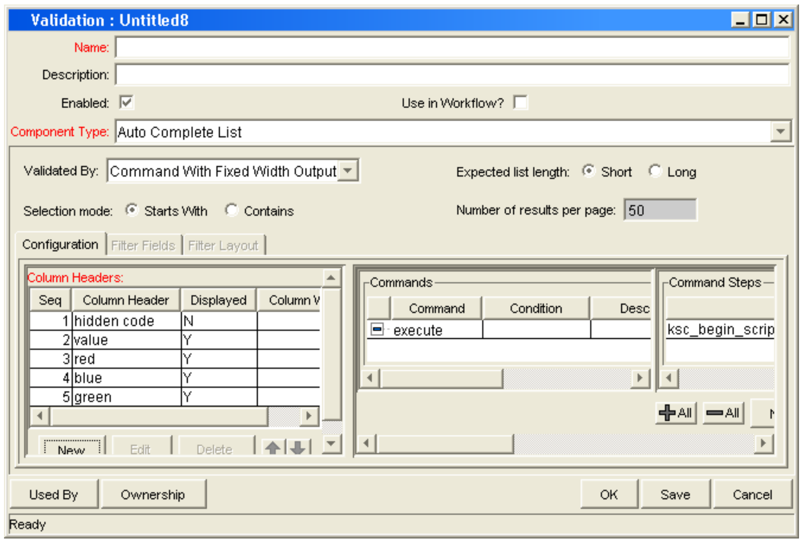
Many enterprises need to use alternate sources of data within their applications. Examples of these sources are a flat file, an alternate database source, or output from a command line execution. Special commands may be used in conjunction with these alternate data sources, in the context of a validation, to provide a list of values on the fly.

In the Validation Workbench, under Validated By, choose **Command With Fixed Width Output** and input the appropriate width information.

Then, under New Command, enter in the command steps to be executed. These can include special commands. Your commands should include the special command `ksc_capture_output`, which captures and parses the delimited command output. If the `ksc_capture_output` special command is surrounded by the `ksc_connect` and `ksc_disconnect` commands, the command will be run on the remote system. Otherwise, the command will be run locally on the Mercury IT Governance server (similar to `ksc_local_exec`).

The example below has the validations red, blue and green. The column width is set to a value of 6. The script places the validations into the newfile.txt file.

```
ksc_begin_script[AS.PKG_TRANSFER_PATH]newfile.txt
red    red
blue   blue
green  green
ksc_end_script
ksc_capture_output cat[AS.PKG_TRANSFER_PATH]newfile.txt
```


Validation : Untitled8

Name:

Description:

Enabled: ✓ Use in Workflow?

Component Type: Auto Complete List

Validated By: Command With Fixed Width Output

Expected list length: [✓ Short] Long

Selection mode: [✓ Starts With] Contains

Number of results per page: 50

Configuration | Filter Fields | Filter Layout

Column Headers:

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	
3	red	Y	
4	blue	Y	
5	green	Y	

Commands

Command	Condition	Desc
execute		

Command Steps

ksc_begin_scrip

Figure 5-9. Validation by command with fixed width output

Table 5-8. Validation by command with fixed width output

Field	Definition
Command Panel	The panel where new commands can be added to capture validation values.

Headers can also be defined for the columns selected. These column headers are used in the window that opens when a value is selected from an auto-complete list. To define a new column header, click **New** under Column Header. *Table 5-9* shows the fields can be entered for a column header. If a column header is not defined for each column in a command, a default name is used.

Table 5-9. Column headers

Field	Definition
Column Header	The name of the column that is displayed in the Auto Complete dialog.
Display	Whether or not the column is displayed. The first column is never displayed and the second column is always displayed.
Column Width	The number of characters in each column of the output generated as a result of the command.

User-Defined Multi-Select Auto-Complete Fields

A number of auto-complete fields in the Workbench have been pre-configured to allow users to open a separate window for selecting multiple values from a list. Users can also define custom auto-complete fields to have multi-select capability when creating various product entities.

The user-defined multi-select capability is supported for:

- User data fields
- Report type fields
- Request type fields
- Project template fields

The user-defined Multi-Select capability is **not** supported for:

- Request header types
- Object types

In order to use this feature when creating a new entity, users must:

- Select a validation for the new entity that has **Auto-Complete List** as the Component Type. This enables the Multi-Select Enabled field in the Field: New window.
- In the Field: New window, users must click **Yes** for the Multi-Select Enabled radio button.

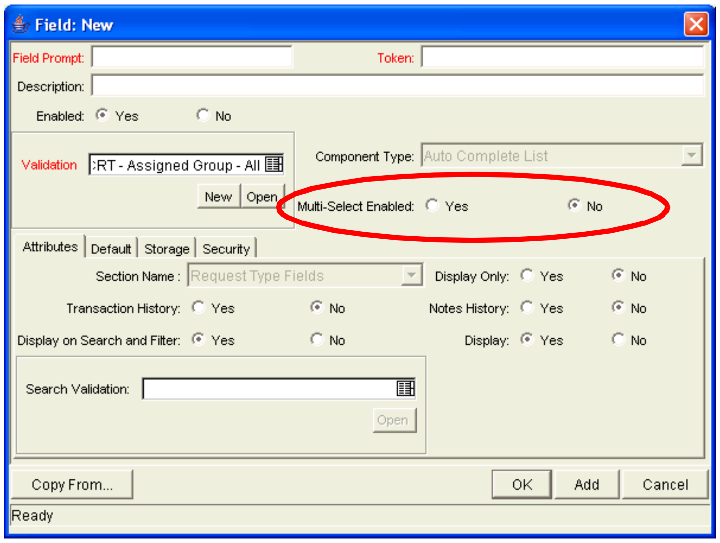
The step-by-step procedure for defining multi-select capability in user data, report type, request type, or project template fields is very similar. The procedure for enabling this capability for request type field is shown below as an example.

To define a multi-select auto-complete field for a request type:

1. Log on to Mercury IT Governance Center and open the Workbench.
2. From the shortcut bar, select **Create > Request Types.**

   The Request Type Workbench opens.
3. Click **New Request Type.**

   The Request Type window opens.
4. Click **New**. The Field: New window opens.



Field Prompt:

Token:

Description:

Enabled: [✓ Yes] No

Validation: :RT - Assigned Group - All

Component Type: Auto Complete List

Multi-Select Enabled: Yes [✓ No]

Section Name : Request Type Fields

Display Only: Yes [✓ No]

Transaction History: Yes [✓ No]

Notes History: Yes [✓ No]

Display on Search and Filter: [✓ Yes] No

Display: [✓ Yes] No

Search Validation:

Ready

5. Select a validation of type **Auto-Complete List** from the **Validation** field.

   The Multi-Select Enabled option is now enabled.

6. Select the **Yes** option for the Multi-Select Enabled radio button.

   The Possible Conflicts window opens. It warns you not to use a multi-select auto-complete for advanced queries, workflow transitions and reports. If this field is not going to be used in advanced queries, workflow transitions or reports, click **Yes** to continue.

7. Configure the other options in this window for the new request type.

8. Click **OK.**

The field is now enabled for multi-select auto-complete.

Example: Token Evaluation and Validation by Command with Delimited Output

The validation functionality can be extended to include field dependent token evaluation. Validations can be configured to dynamically change, depending on the client-side value entered in another field.

To use field dependent token evaluation, it is necessary to configure a validation in conjunction with an object type, request type, report type, project template, or user data definition. Consider the following example for setting up an object type using field dependent tokens.

1. Generate a validation and set the following parameters as shown:

   a. Name: **demo_client_token_parsing**

   b. Component Type: **Auto Complete List**

   c. Validated By: **Command With Delimited Output**

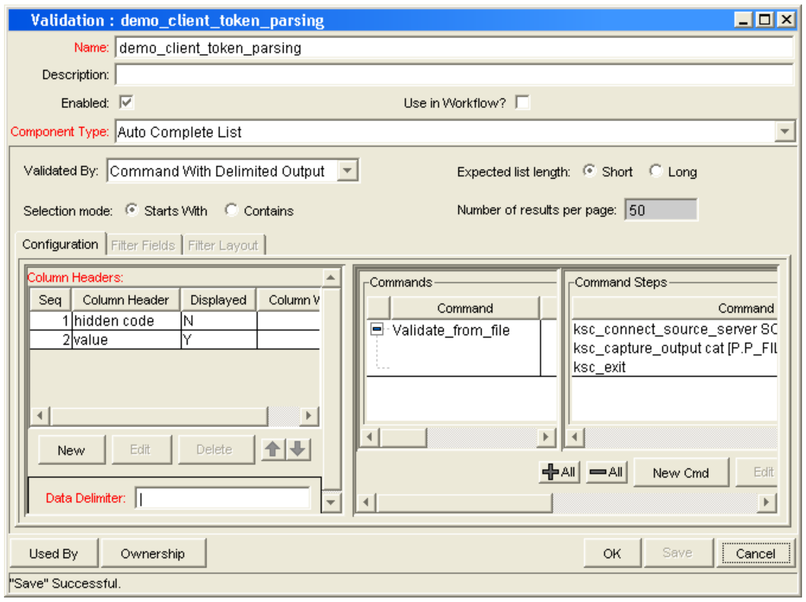
   d. Data Delimiter: | (bar)

   e. Command

      - Command: **Validate_from_file**

      - Steps

```
ksc_connect_source_server SOURCE_ENV="Your Env"
ksc_capture_output cat [P.P_FILENAME]
ksc_exit
```



Validation : demo_client_token_parsing

Name: demo_client_token_parsing
Description:
Enabled: ✓
Use in Workflow? ☐
Component Type: Auto Complete List
Validated By: Command With Delimited Output
Expected list length: (● Short) ○ Long
Selection mode: (● Starts With) ○ Contains
Number of results per page: 50

Configuration | Filter Fields | Filter Layout

Column Headers:

Seq	Column Header	Displayed	Column W
1	hidden code	N	
2	value	Y	

New | Edit | Delete

Data Delimiter: |

Commands

Command
Validate_from_file

Command Steps

Command
ksc_connect_source_server SC
ksc_capture_output cat [P.P_FIL
ksc_exit

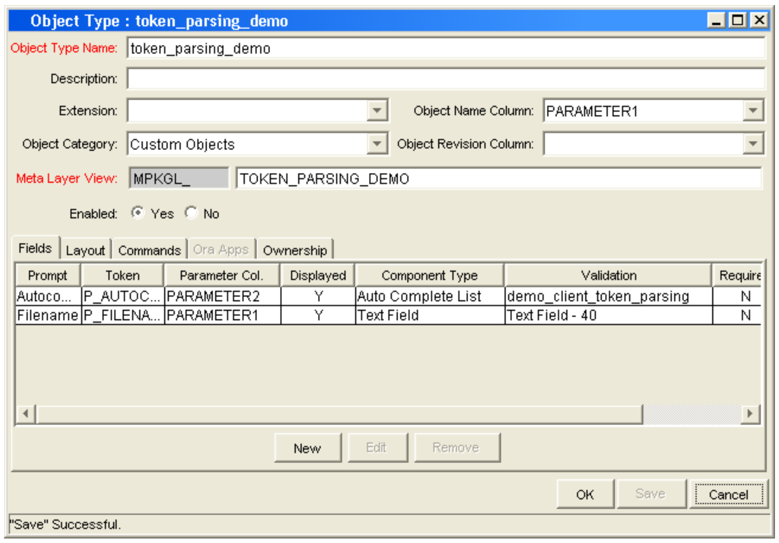
+All | −All | New Cmd | Edit

Used By | Ownership | OK | Save | Cancel

"Save" Successful.

When called, this validation will connect to an environment called ‘Your Env’ and retrieve data from a file specified by the token P_FILENAME. The file should be located in the directory specified in the Base Path in the Environment window.

2. Generate an object type named **token_parsing_demo.**



Object Type : token_parsing_demo

Object Type Name: token_parsing_demo

Description:

Extension:

Object Name Column: PARAMETER1

Object Category: Custom Objects

Object Revision Column:

Meta Layer View: MPKGL_ TOKEN_PARSING_DEMO

Enabled: (● Yes) ○ No

Prompt	Token	Parameter Col.	Displayed	Component Type	Validation	Require
Autoco...	P_AUTOC...	PARAMETER2	Y	Auto Complete List	demo_client_token_parsing	N
Filename	P_FILENA...	PARAMETER1	Y	Text Field	Text Field - 40	N

"Save" Successful.

a. Generate a new field with the following parameters:

- Name: **Filename**
- Token: **P_FILENAME**
- Validation: **Text Field - 40**

b. Generate a new field with the following parameters:

- Name: **AutoComp**
- Token: **P_AUTOCOMP**
- Validation: **demo_client_token_parsing** (this is the validation that was defined above)

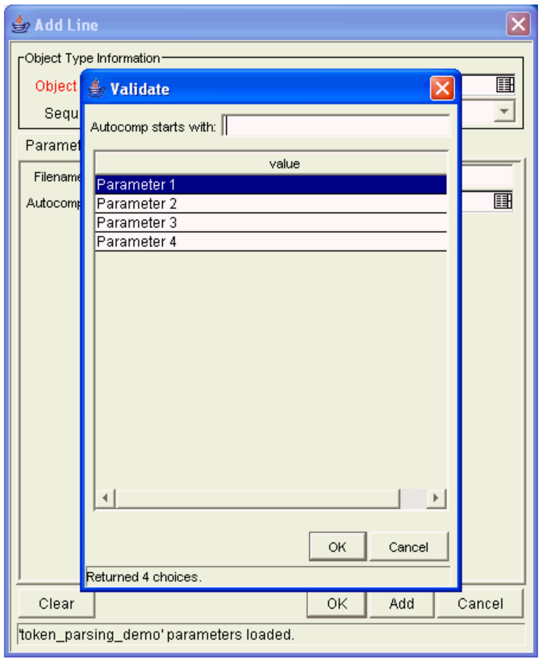
3. For this example to return any values in the auto-complete, a file must be generated in the directory specified in the Base Path in the Environment Detail of ‘Your Env’ environment. Generate a file named ‘parse_test1.txt’ with the following delimited data:

```
DELIMITED_TEXT1|Parameter 1
DELIMITED_TEXT2|Parameter 2
DELIMITED_TEXT3|Parameter 3
DELIMITED_TEXT4|Parameter 4
```

The object type ‘token_parsing_demo’ is now enabled to use this token evaluation.

To test the above configuration sample:

1. Generate a new package.
2. Select a workflow and click **Add Line.**
3. Select **token_parsing_demo** from the Object Type drop-down list. The following fields are displayed:
   - Filename
   - AutoComp
4. Type ‘parse_test1.txt’ in the Filename field.
5. Click on the auto-complete icon in the AutoComp field. The following Validation window opens, displaying the contents of the ‘parse_test1.txt’ file.



Parameter 1
Parameter 2
Parameter 3
Parameter 4

Returned 4 choices.

'token_parsing_demo' parameters loaded.

Configuring Text Fields

Text fields displayed on a single line. Text fields can be configured to display the data according to a certain format. For example, you can configure a text field to accept and format a ten digit telephone number or display a specific number of decimal places for a percentage.

Creating a Text Field Validation Overview

To create a text field validation:

1. Open the Validation window in the Workbench.
2. In the Name field, enter the name of the validation.
3. From the Component Type field, select **Text Field.**
4. From the Data Mask field, select the data mask that represents the desired format for the field.

   See "Available Text Data Masks" on page 104 for additional details.
5. (Optional) Configure the selected data mask.

   See *Customizing the System Text Data Masks* on page 106 for additional details.
6. Click **OK.**

Available Text Data Masks

The Mercury IT Governance Center includes a number of preconfigured data masks that can be used when creating text field validations. Each of these data masks can be configured to meet your specific data requirements. *Table 5-10* defines the data masks delivered with Mercury IT Governance Center.

Table 5-10. Data Mask Formats

Data Mask	Description
Alphanumeric	Field allows all alphanumeric characters. The maximum field length for fields using this validation can be specified.
Alphanumeric Uppercase	Field allows alphanumeric characters and formats all characters as uppercase text. The maximum field length for fields using this validation can be specified.
Numeric	Field allows only numeric characters. The following characteristics can be specified for this data mask: • Range of values (maximum and minimum) allowed for this field • Whether or not a zero is displayed when data is not entered into the field • Whether or not group separators (such as a comma) are used within large numbers • How negative numbers are displayed • Number of decimal places See *Customizing the Numeric Data Mask* on page 106 for details.
Currency	Field allows only numeric characters and is used to display currency data. The following characteristics can be specified for this data mask: • Range of values (maximum and minimum) allowed for this field • Whether or not a zero is displayed when data is not entered into the field • Whether or not group separators (such as a comma) are used within large numbers • How negative numbers are displayed • Number of decimal places See *Customizing the Currency Data Mask* on page 108 for details.

Table 5-10. Data Mask Formats [continued]

Data Mask	Description
Percentage	Field allows only numeric characters and is used to display percentages. The following characteristics can be specified for this data mask: • Range of values (maximum and minimum) allowed for this field • Whether or not a zero is displayed when data is not entered into the field • Whether or not group separators (such as a comma) are used within large numbers • How negative numbers are displayed • Number of decimal places See *Customizing the Percentage Data Mask* on page 110 for details.
Telephone	Field allows only numeric characters and is used to display telephone numbers. The following characteristics can be specified for this data mask: • Format—specify how many digits are included, and what delimiter should be used between groups of numbers. For example, you can select to use dashes (-) rather than periods (.) between numbers: 555-555-5555 or 555.555.5555. • Maximum and minimum number of digits See *Customizing the Telephone Data Mask* on page 112 for details.
Custom	Field allows a range of custom inputs. You can customize the field to accept digits, letters, spaces, and custom delimiters. See *Creating a Custom Data Mask* on page 115 for details.

Customizing the System Text Data Masks

Each data mask that is included in Mercury IT Governance Center can be customized.

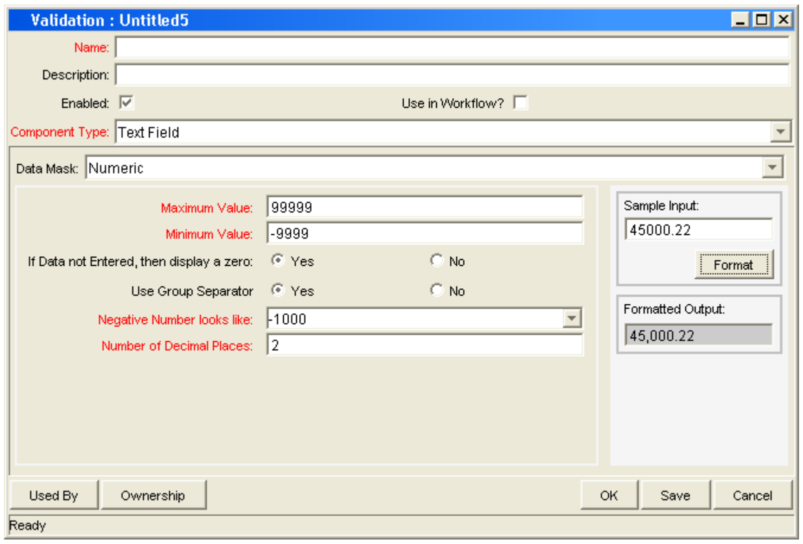
Customizing the Numeric Data Mask

The numeric data mask allows only numeric characters. When creating a validation using this data mask, the following characteristics can be specified:

- Range of values (maximum and minimum) allowed for this field

- Whether or not a zero is displayed when data is not entered into the field
- Whether or not group separators (such as a comma) are used within large numbers
- How negative numbers are displayed
- Number of decimal places

Figure 5-10 shows the fields that can be configured for this data mask. *Table 5-11* defines these fields.



Validation : Untitled5

Name:

Description:

Enabled: ✓

Use in Workflow? ☐

Component Type: Text Field

Data Mask: Numeric

Maximum Value: 99999

Minimum Value: -9999

If Data not Entered, then display a zero: [✓ Yes] No

Use Group Separator [✓ Yes] No

Negative Number looks like: -1000

Number of Decimal Places: 2

Sample Input: 45000.22

Formatted Output: 45,000.22

Ready

Figure 5-10. Validation window for the numeric data mask

Table 5-11. Fields for configuring the numeric data mask for text fields

Field	Description
Maximum Value	Largest value allowed for this field. This can be a positive or negative number.
Minimum Value	Smallest value allowed for this field. This can be a positive or negative number.
If Data not Entered, then display a zero	Determines if the field should display a zero when no data is entered.

Table 5-11. Fields for configuring the numeric data mask for text fields

Field	Description
Use Group Separator	Determines if the field should use a group separator (such as a comma) to divide characters within large numbers. For example: 1000000 versus 1,000,000. The character used for the separator defaults based on the machine's local, but can be configured in the Regional Settings window in the Workbench. Select **Edit > Regional Settings** to access this window.
Negative Number looks like	Determines the appearance of negative numbers. There are four options available: • (1000)—parenthesis and black text • (1000)—parenthesis and red text • -1000—minus sign (-) and black text • -1000—minus sign (-) and red text
Number of Decimal Places	Determines the number of allowed decimal places. Users will only be able to enter up to this number of digits beyond the decimal place.

To view your customized data mask:

1. In the Sample Input field, enter the digits that you would like to see formatted.

2. Click **Format.**

   The digits are formatted according to your settings and displayed in the Formatted Output field.

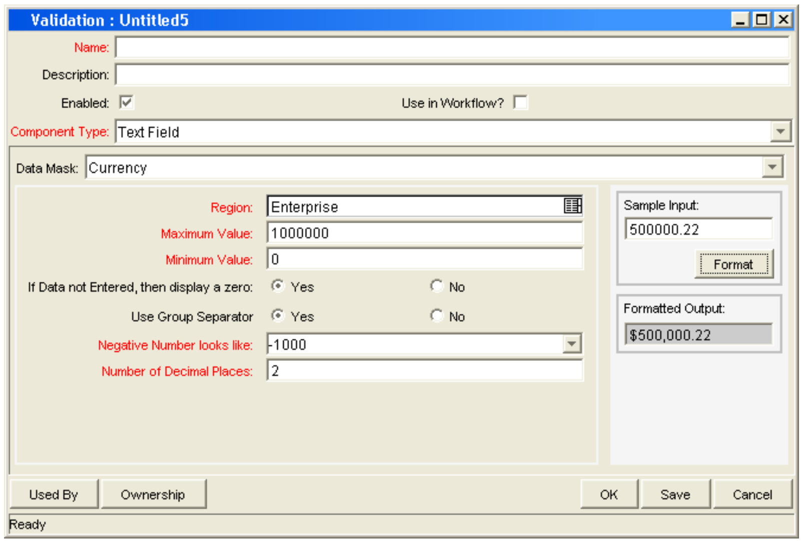
Customizing the Currency Data Mask

The currency data mask allows only numeric characters and is used to display currency data. When creating a validation using this data mask, the following characteristics can be specified:

- Range of values (maximum and minimum) allowed for this field
- Whether or not a zero is displayed when data is not entered into the field
- Whether or not group separators (such as a comma) are used within large numbers
- How negative numbers are displayed

- Number of decimal places

Figure 5-11 shows the fields that can be configured for this data mask. *Table 5-12* defines these fields.



Name:
Description:
Enabled: ✓ Use in Workflow? ☐
Component Type: Text Field
Data Mask: Currency
Region: Enterprise
Maximum Value: 1000000
Minimum Value: 0
If Data not Entered, then display a zero: [✓ Yes] No
Use Group Separator [✓ Yes] No
Negative Number looks like: -1000
Number of Decimal Places: 2
Sample Input: 500000.22
Formatted Output: $500,000.22
Ready

Figure 5-11. Validation window for the currency data mask

Table 5-12. Fields for configuring the currency data mask for text fields

Field	Description
Maximum Value	Largest value allowed for this field. This can be a positive or negative number.
Minimum Value	Smallest value allowed for this field. This can be a positive or negative number.
If Data not Entered, then display a zero	Determines if the field should display a zero when no data is entered.
Use Group Separator	Determines if the field should use a group separator (such as a comma) to divide characters within large numbers. For example: 1000000 versus 1,000,000. The character used for the separator defaults based on the machine's local, but can be configured in the Regional Settings window in the Workbench. Select **Edit > Regional Settings** to access this window.

Table 5-12. Fields for configuring the currency data mask for text fields

Field	Description
Negative Number looks like	Determines the appearance of negative numbers. There are four options available: • (1000)—parenthesis and black text • (1000)—parenthesis and red text • -1000—minus sign (-) and black text • -1000—minus sign (-) and red text
Number of Decimal Places	Determines the number of allowed decimal places. Users will only be able to enter up to this number of digits beyond the decimal place.

To view your customized data mask:

1. In the Sample Input field, enter the digits that you would like to see formatted.
2. Click **Format.**

   The digits are formatted according to your settings and displayed in the Formatted Output field.

Note

The INSTALLATION_CURRENCY server parameter dictates which currency symbol is displayed in the field. This parameter also dictated the position of the text in the field. For example:

```
INSTALLATION_CURRENCY=$;RIGHT
```

will right-align the text using a dollar sign.
Contact your system administrator for help with changing this setting.

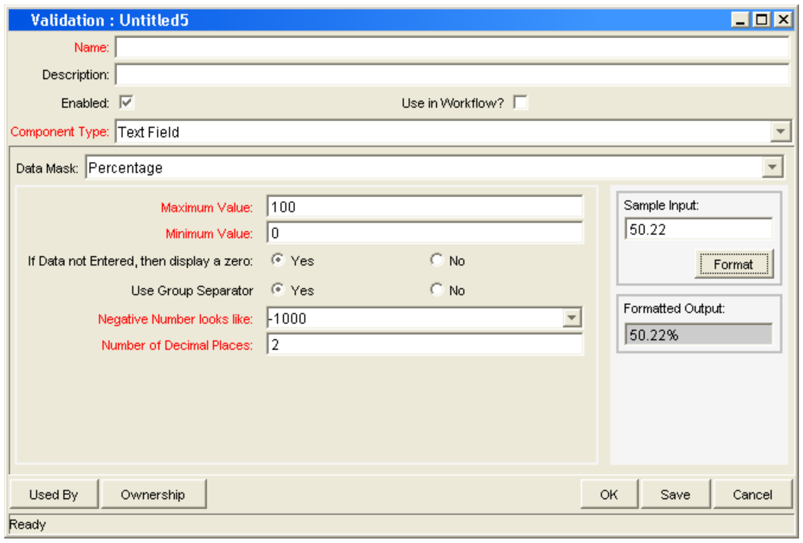
Customizing the Percentage Data Mask

The percentage data mask allows only numeric characters and is used to display percentages. When creating a validation using this data mask, the following characteristics can be specified:

- Range of values (maximum and minimum) allowed for this field
- Whether or not a zero is displayed when data is not entered into the field

- Whether or not group separators (such as a comma) are used within large numbers
- How negative numbers are displayed
- Number of decimal places

Figure 5-12 shows the fields that can be configured for this data mask. *Table 5-13* defines these fields.


Validation : Untitled5

Name:

Description:

Enabled: ✓ Use in Workflow? ☐

Component Type: Text Field

Data Mask: Percentage

Maximum Value: 100

Minimum Value: 0

If Data not Entered, then display a zero: [✓ Yes] No

Use Group Separator [✓ Yes] No

Negative Number looks like: -1000

Number of Decimal Places: 2

Sample Input: 50.22

Formatted Output: 50.22%

Ready

Figure 5-12. Validation window for the percentage data mask

Table 5-13. Fields for configuring the percentage data mask for text fields

Field	Description
Maximum Value	Largest value allowed for this field. This can be a positive or negative number.
Minimum Value	Smallest value allowed for this field. This can be a positive or negative number.
If Data not Entered, then display a zero	Determines if the field should display a zero when no data is entered.
Use Group Separator	Determines if the field should use a group separator (such as a comma) to divide characters within large numbers. For example: 1000000 versus 1,000,000. The character used for the separator defaults based on the machine's local, but can be configured in the Regional Settings window in the Workbench. Select **Edit > Regional Settings** to access this window.

Table 5-13. Fields for configuring the percentage data mask for text fields

Field	Description
Negative Number looks like	Determines the appearance of negative numbers. There are four options available: • (1000)—parenthesis and black text • (1000)—parenthesis and red text • -1000—minus sign (-) and black text • -1000—minus sign (-) and red text
Number of Decimal Places	Determines the number of allowed decimal places. Users will only be able to enter up to this number of digits beyond the decimal place.

To view your customized data mask:

1. In the Sample Input field, enter the digits that you would like to see formatted.
2. Click **Format.**

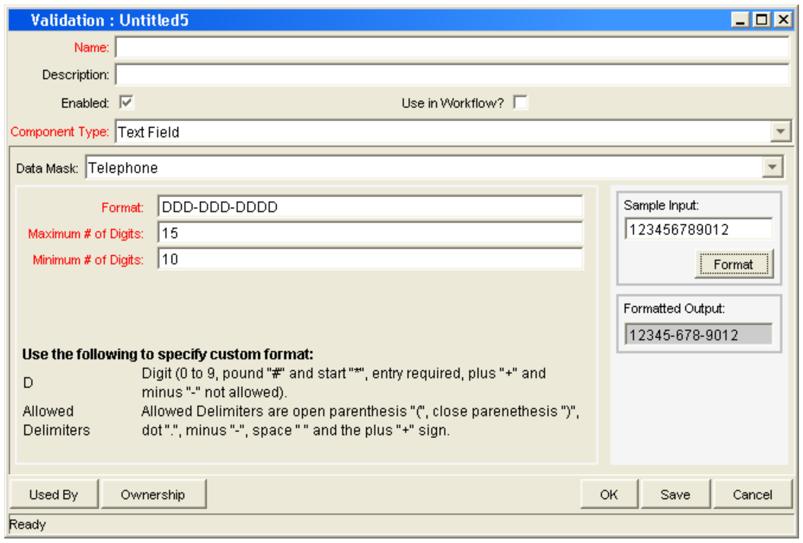
   The digits are formatted according to your settings and displayed in the Formatted Output field.

Customizing the Telephone Data Mask

The percentage data mask allows only numeric characters and is used to display telephone numbers. When creating a validation using this data mask, the following characteristics can be specified:

- Format—specify how many digits are included, and what delimiter should be used between groups of numbers. For example, you can select to use dashes (-) rather than periods (.) between numbers. For example, 555-555-5555 or 555.555.5555.
- Maximum and minimum number of digits.

Figure 5-13 shows the fields that can be configured for this data mask. *Table 5-14* defines these fields.



Name:
Description:
Enabled: ✓ Use in Workflow? ☐
Component Type: Text Field
Data Mask: Telephone
Format: DDD-DDD-DDDD
Maximum # of Digits: 15
Minimum # of Digits: 10
Sample Input: 123456789012
Formatted Output: 12345-678-9012

Use the following to specify custom format:

D	Digit (0 to 9, pound "#" and start "*", entry required, plus "+" and minus "-" not allowed).
Allowed Delimiters	Allowed Delimiters are open parenthesis "(", close parenethesis ")", dot ".", minus "-", space " " and the plus "+" sign.

Ready

Figure 5-13. Validation window for the telephone data mask

Table 5-14. Fields for configuring the telephone data mask for text fields

Field	Description
Format	The rule that dictates how the digits are formatted, including any spaces or delimiters. The following delimiters are allowed in the format definition: • Open and close parentheses ( ) • Period (.) • Dash (-) • Space • Plus sign (+) See *Table 5-15* on page 114 for a few examples of different Telephone formats.
Maximum # of Digits	Largest number of digits that will be accepted in this field.
Minimum # of Digits	Smallest number of digits that will be accepted in this field. If the user enters fewer than this number of digits in the field and then tries to move from the field, he will receive an error.

Table 5-15. Sample telephone data mask formats

Format Rule	Text Entered By User	Sample Formatted Output
D-DDD-DDD-DDDD	15555555555	1-555-555-5555
DDD DDD DDDD	5555555555	555 555 5555
(DDD) DDD-DDDD	5555555555	(555) 555-5555

To view your customized data mask:

1. In the Sample Input field, enter the digits that you would like to see formatted.
2. Click **Format.**

   The digits are formatted according to your settings and displayed in the Formatted Output field.

Note

Special behavior applies to the extra characters, if your format is defined to allow a range of entries. Extra characters will always be grouped with the first set of characters. For example, if the telephone data mask is configured with a minimum of 10 characters and a maximum of 15 characters, then the following behavior is expected:

```
Format: DDD-DDD-DDDD
   Min: 10
   Max: 15
```

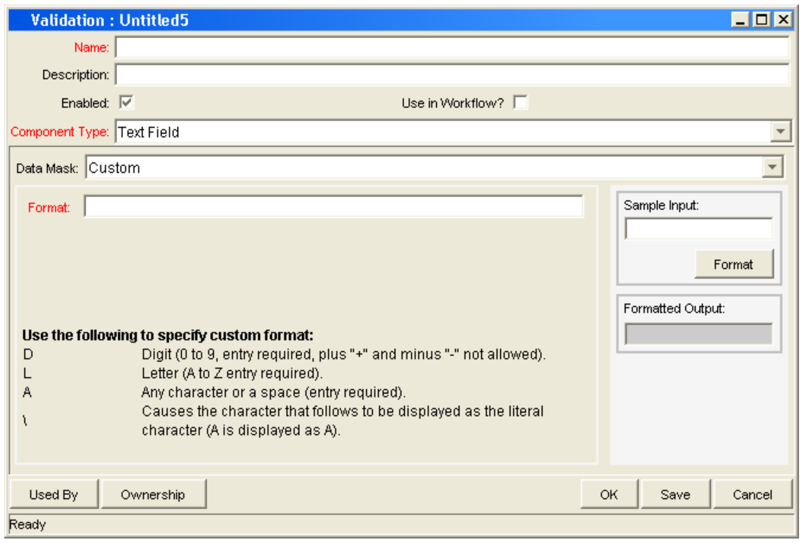
Input: `1234567890`
Output: `123-456-7890`

Input 2: `12345678901`
Output 2: `1234-567-8901`

Creating a Custom Data Mask

A custom data mask can be defined that will allow a range of inputs and format them to your specification. You can customize the field to accept digits, letters, spaces, and custom delimiters.

Figure 5-13 shows the fields that can be configured for this data mask.



Name:
Description:
Enabled: ✓ Use in Workflow? ☐
Component Type: Text Field
Data Mask: Custom
Format:
Sample Input:
Formatted Output:

Use the following to specify custom format:

D	Digit (0 to 9, entry required, plus "+" and minus "-" not allowed).
L	Letter (A to Z entry required).
A	Any character or a space (entry required).
\	Causes the character that follows to be displayed as the literal character (A is displayed as A).

Ready

Figure 5-14. Validation window for the custom data mask

To configure a custom format, enter a combination of symbols into the Format field. This field can accept the following entries:

- D—Specifies a required digit between 0 and 9.
- L—Specifies a required letter between A and Z.
- A—Specifies a required character or space.
- \ (backslash)—Causes the character that follows to be displayed as the literal character. For example: "\A" will be displayed as "A"

Table 5-16 displays some examples of custom formats.

Table 5-16. Sample custom data mask formats

Format Rule	Text Entered By User	Formatted Output
DDD\-DD\-DDDD	555555555	555-55-5555
AA\-DDD	BC349	BC-349

To view your customized data mask:

1. In the Sample Input field, enter the digits that you would like to see formatted.

2. Click **Format.**

   The digits are formatted according to your settings and displayed in the Formatted Output field.

Using Directory and File Choosers

Directory and File Choosers are only used with Mercury Change Management object types.

Directory Chooser

The Directory Chooser field can be used to select a valid directory from an environment. Mercury Change Management connects to the first source environment on a workflow and allows navigation through the directory structure and the selection of a directory from the list.

When implementing the Directory Chooser, note the following:

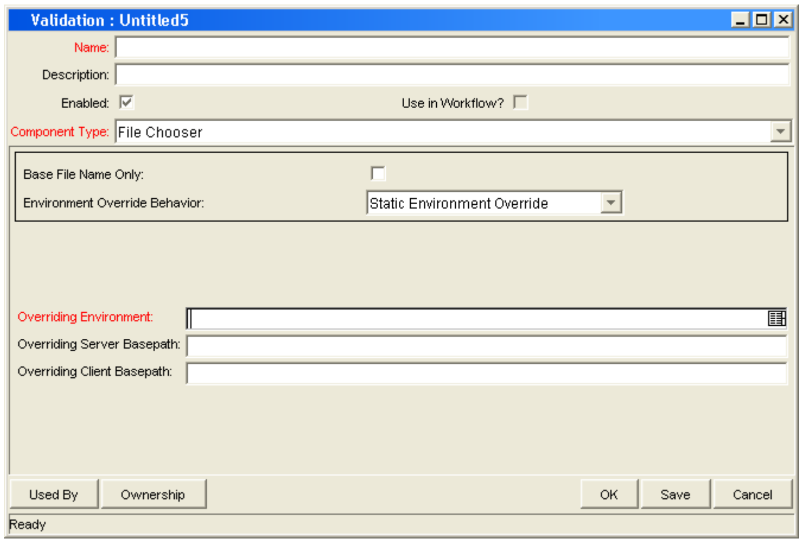
- The Directory Chooser field can only be used on an object type.
- On every object type that a Directory Chooser is chosen, it is also necessary to have a field whose token is **P_FILE_LOCATION** and whose validation is **DLV - File Location**. The possible values for this field are **Client** and **Server**. If **Client** is chosen, the Directory Chooser connects to the Client Base Path of the source environment. If **Server** is chosen, the Directory Chooser connects to the Server Base Path of the source environment.

File Chooser

A File Chooser field can be used by object types to select a valid file from an environment. Mercury Change Management connects to the first source environment on a workflow and provides the ability to view all files within a specific directory and select one from the list.

On every object type that a File Chooser is chosen, it is necessary to define the following fields:

- The first is a field for the File Location for the directory chooser, described in the previous section.
- The second is a field whose token is 'P_SUB_PATH'. This field is the directory from which the file is selected and is usually a Directory Chooser field.


Validation : Untitled5

Name:

Description:

Enabled: ✓ Use in Workflow? ☐

Component Type: File Chooser

Base File Name Only: ☐

Environment Override Behavior: Static Environment Override

Overriding Environment:

Overriding Server Basepath:

Overriding Client Basepath:

Figure 5-15. Validation window for static environment override in file chooser.

Table 5-17. File chooser field

Field	Definition
Base File Name Only	Defines whether the base file name only (without its suffix) or the complete name is displayed.
Environment Override Behavior	Used to select files from a specific environment other than the default environment.

The Environment Override Behavior drop-down list contains three options: **Default Behavior**, **Static Environment Override**, and **Token-Based Environment Override.**

Static Environment Override provides the ability to override one environment at a time. The fields for static environment override are pictured in *Figure 5-15* and described in *Table 5-18*.

Table 5-18. Static environment override

Field	Definition
Overriding Environment	Selects the environment to be overridden.
Overriding Server Basepath	The server basepath of the environment may be overridden.
Overriding Client Basepath	The client basepath of the environment may be overridden.

Token-based Environment Override provides the ability to select a token that will resolve to the overriding environment. The fields for **Token-based Environment Override** are shown in *Figure 5-16* and defined in *Table 5-19*.

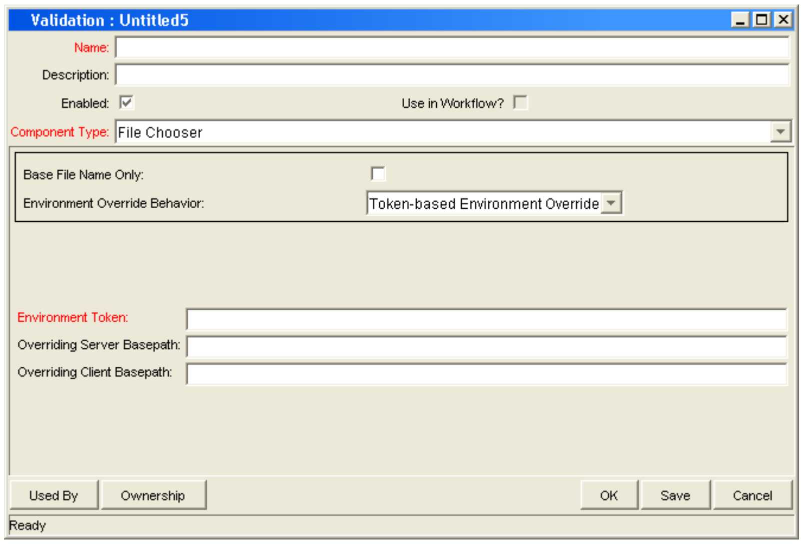

Validation : Untitled5
Name:
Description:
Enabled: ✓
Use in Workflow?
Component Type: File Chooser
Base File Name Only:
Environment Override Behavior: Token-based Environment Override
Environment Token:
Overriding Server Basepath:
Overriding Client Basepath:
Used By
Ownership
OK
Save
Cancel
Ready

Figure 5-16. Validation window for token-based environment override in file chooser.

Table 5-19. Token-based environment override

Field	Definition
Environment Token	Select the token that will resolve to the overriding environment.
Overriding Server Basepath	The server basepath of the environment that is to be resolved by the token may be overridden.
Overriding Client Basepath	The client basepath of the environment that is to be resolved by the token may be overridden.

Date Field Formats

Date fields can accept a variety of formats. The current date field validations are separated into two categories: all systems, and systems using only the English language. These formats are defined in Table 3-14.

Table 5-20. Date field

Field		Definitions
Name	Systems	
Date Format	All	The format for the date part of the field. Choices are: · Long - “January 2, 1999". · Medium - “02-Jan-99”. · Short - “1/2/99”. · None - no date is displayed.
Date Format	English Only	The format for the date part of the field. Choices are: · MM/DD/YY (6/16/99). · DD-MON-YY (16-Jun-99). · MONTH DD, YYYY (June 16, 1999). · Day, Month DD, YYYY (Monday, June 16, 1999). · DD-MON (16-JUN) - Defaults to current year. · DD-MON-YYYY (16-JUN-1999). · MM-DD-YYYY (06-16-1999). · MM-DD-YY (06-16-99). · DD [Defaults to the current month and year]. · MM/DD (06/16) - Defaults to current year. · MM/DD/YYYY (06/16/1999).
Time Format	All	The format for the time part of the field. Choices are: · Long - the time is displayed as “12:00:00 PM PST”. · Medium - the time is displayed as “12:00:00 PM”. · Short - the time is displayed as “12:00 PM”. · None - no time is displayed.

Creating 1800 Character Text Areas

Standard Text Areas are either 40 or 200 characters. You can, however, create a Text Area validation with a character length of 1800.

To create a validation with a character length of 1800:

1. Open the Validation Workbench.

2. Search for “**Text Area - 1800**.”
3. In the results tab, select **Text Area - 1800.**
4. Click **Copy.**
5. Rename the validation.

   The new Text Area validation (with a length of 1800) can be used when defining a custom field in the product.

Note

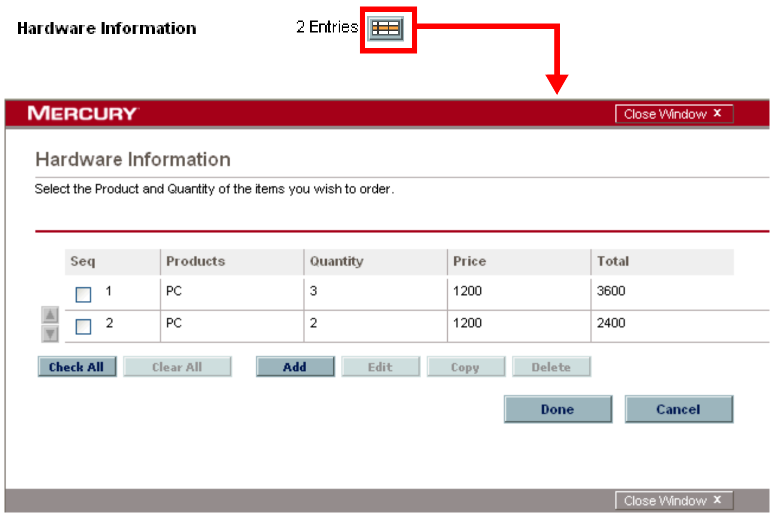
You can only create a Text Field or Area of length 40, 200, or 1800.

Configuring the Table Component

The table component is used to enter multiple records into a single field on a request. The table component can be configured to include multiple columns of varied data types. Additionally, this component supports rules for populating elements within the table and provides functionality for capturing column totals.

For example, ACME creates a request type to request quotes and parts for hardware. Each entry of this type has four elements: Product, Quantity, Price, and Total. ACME creates a table component field called Hardware Information to collect this information.

When the user logs a request for new hardware, the request displays the Hardware Information field. The user opens the field. He selects a Product, which triggers a rule to populate the Price and Total. He submits the request, which now contains all of the information required to successfully order the hardware.



Hardware Information

Select the Product and Quantity of the items you wish to order.

Seq	Products	Quantity	Price	Total
1	PC	3	1200	3600
2	PC	2	1200	2400

1. Click the Table Component icon to open the Table Component entry page.

2. Add, edit, or delete entries in the list.

Fields of this component can only be added to request types, request header types and request user data.

To configure and use a table component:

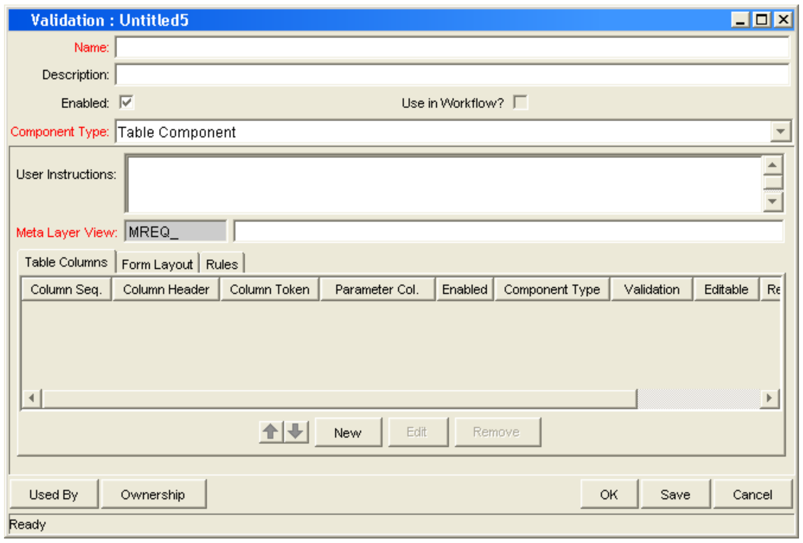
1. Defining the Table Component in the Validation Workbench
2. Adding the Table Component to a Request Type

Defining the Table Component in the Validation Workbench

To create a table component field:

1. Log on to Mercury IT Governance Center and open the Workbench.
2. From the shortcut bar, select **Configuration > Validations.**
3. The Validation Workbench opens.
4. Click **New Validation.**

   The Validation window opens.
5. Select **Table Component** from the Component Type drop-down list.



Name:

Description:

Enabled: ✓

Use in Workflow? ☐

Component Type: Table Component

User Instructions:

Meta Layer View: MREQ_

Column Seq.	Column Header	Column Token	Parameter Col.	Enabled	Component Type	Validation	Editable	Re

Ready

6. Enter a validation Name and Description.

7. Enter any User Instructions.

   This text will appear on the top of the table entry page.

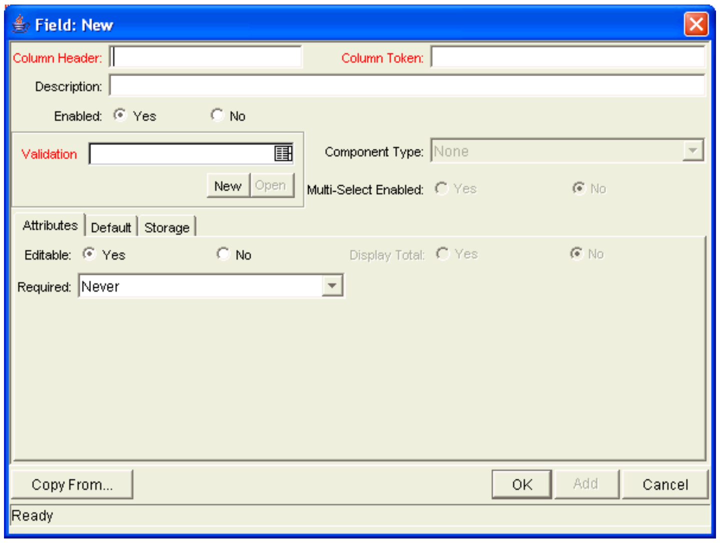
8. Create the table columns.

   a. Click **New** in the **Table Columns** tab. The Field window opens.

   b. Define the type of information that will be stored in that column's entries. This may require you to create a new validation for the column.

Note

File attachments can not be used in a Table component column.

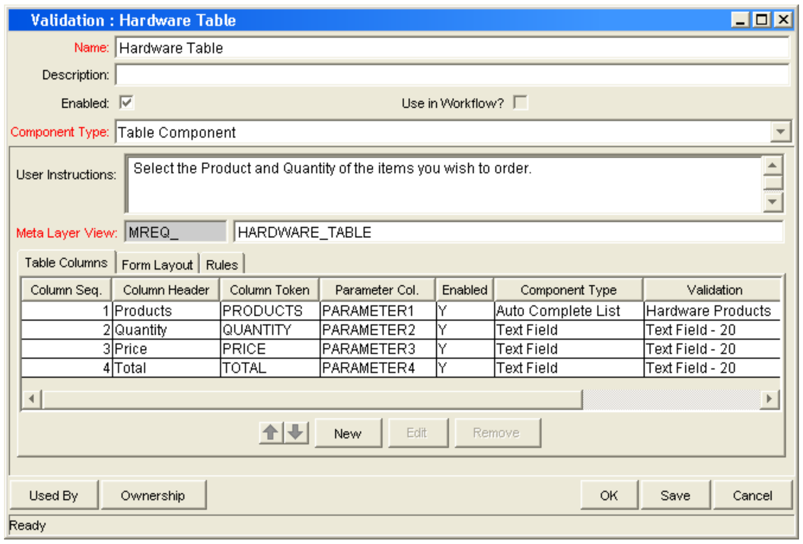


Column Header:	
Column Token:	
Description:	
Enabled:	[✓ Yes] No
Validation	
Component Type:	None
Multi-Select Enabled:	Yes [✓ No]
Editable:	[✓ Yes] No
Display Total:	Yes [✓ No]
Required:	Never

Ready

c. Specify the **Attributes** (Editable or Required) and any **Default** behavior.

d. Click **Add** to save the column information and add another column. When you are finished adding columns, click **OK** to close the Field window.



Name:	Hardware Table
Description:	
Enabled:	✓
Use in Workflow?	
Component Type:	Table Component
User Instructions:	Select the Product and Quantity of the items you wish to order.
Meta Layer View:	MREQ_ HARDWARE_TABLE

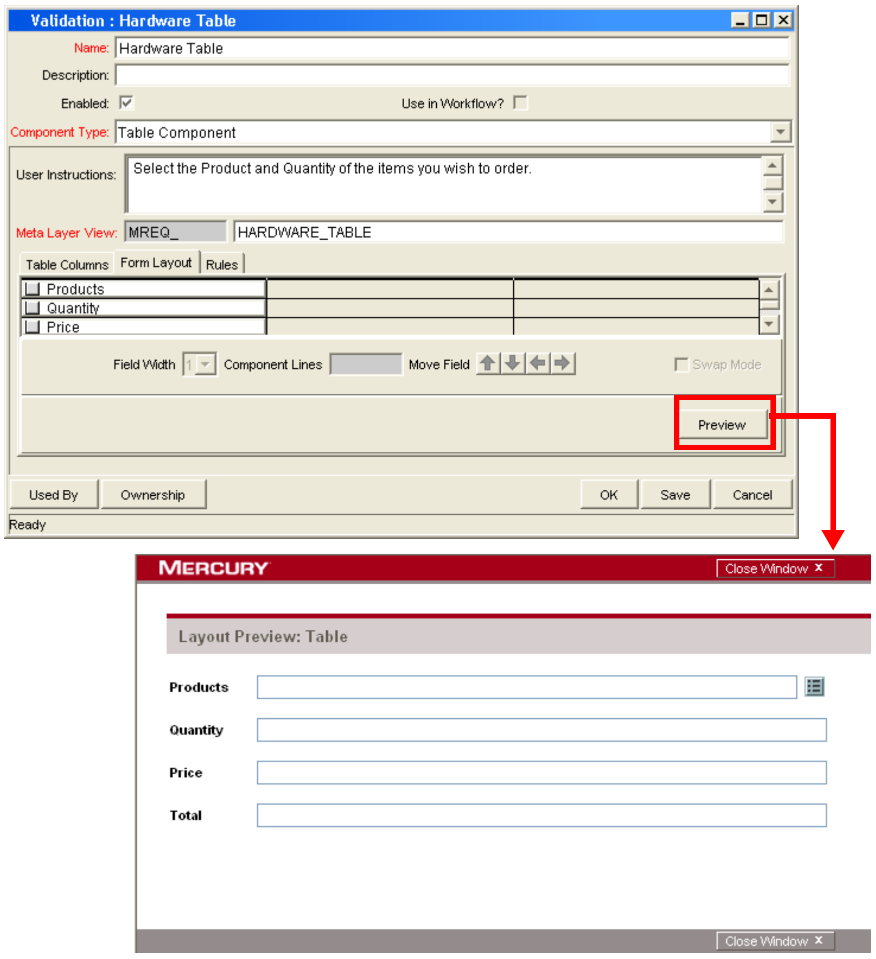
Column Seq.	Column Header	Column Token	Parameter Col.	Enabled	Component Type	Validation
1	Products	PRODUCTS	PARAMETER1	Y	Auto Complete List	Hardware Products
2	Quantity	QUANTITY	PARAMETER2	Y	Text Field	Text Field - 20
3	Price	PRICE	PARAMETER3	Y	Text Field	Text Field - 20
4	Total	TOTAL	PARAMETER4	Y	Text Field	Text Field - 20

Ready

9. Configure the form layout.

a. Click the **Form Layout** tab.

b. Select the fields and move their positions using the arrow buttons.



Validation : Hardware Table

Name: Hardware Table

Description:

Enabled: ✓

Use in Workflow?

Component Type: Table Component

User Instructions: Select the Product and Quantity of the items you wish to order.

Meta Layer View: MREQ_ HARDWARE_TABLE

Products		
Quantity		
Price		

Field Width 1 Component Lines Move Field Swap Mode

Layout Preview: Table

Products

Quantity

Price

Total

c. Click **Preview** to see a representation of the final positioning.

Note that the preview loads a window in the Workbench, but the actual table component will only be available to users in the standard interface (HTML).

10. Configure any table logic in the **Rules** tab.

Rules are used for advanced defaulting behavior and calculating column totals.

a. Click the **Rules** tab.

b. Click **New** to define a new rule.

See *Creating a Table Rule* on page 126 for detailed instructions.

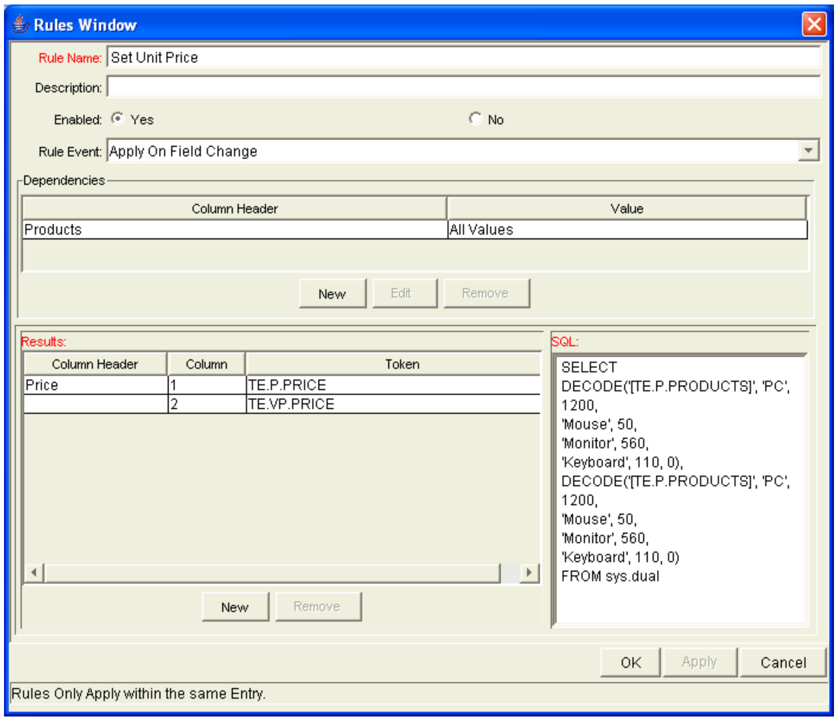
11. Click **OK** to save the validation.

The new table component field can be included on a request type, request header type or request user data field.

Creating a Table Rule

Table rules are configured in the same manner as advanced request type rules. Essentially, you can configure fields (columns) in the table to default to certain values based on an event or value in another field in the table. Because the table component rules are configured using a SQL statement, you are given enormous flexibility for the data that is populated in the table cells.

Table rules are configured using the **Rules** tab on the Validation window.



Rule Name: Set Unit Price

Description:

Enabled: [✓ Yes] No

Rule Event: Apply On Field Change

Dependencies

Column Header	Value
Products	All Values

Results:

Column Header	Column	Token
Price	1	TE.P.PRICE
	2	TE.VP.PRICE

SQL:

```
SELECT
DECODE('[TE.P.PRODUCTS]', 'PC',
1200,
'Mouse', 50,
'Monitor', 560,
'Keyboard', 110, 0),
DECODE('[TE.P.PRODUCTS]', 'PC',
1200,
'Mouse', 50,
'Monitor', 560,
'Keyboard', 110, 0)
FROM sys.dual
```

Rules Only Apply within the same Entry.

Figure 5-17. Rules window accessed from the Rules tab

Example: Using a Table Component on an Order Form

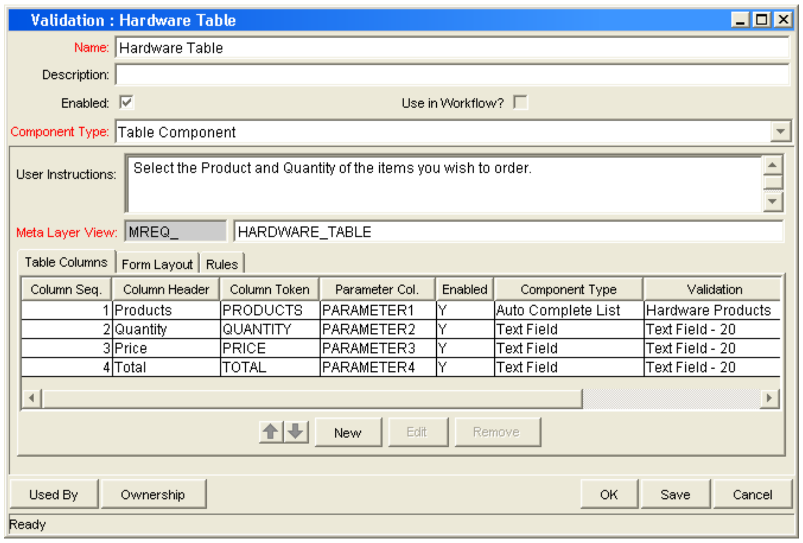
The following example illustrates the table component rules functionality.

ACME uses a request for creating and tracking employee computer hardware equipment orders. ACME has included a table component field on their request type for gathering the order information. When the employee selects a Product, the Unit Price is automatically updated. Then, when they update the Quantity, the total line cost is automatically calculated and displayed in the table.

To enable this functionality, ACME first has to configure a new validation with the following specifications:

Table 5-21. Example - table component validation settings

Setting	Value / Description
Validation Name	Product Order Information
Component Type	Table Component
Column 1	Column Header = Products Column Token = PRODUCTS Validation = Auto complete list with the following list values: PC, MOUSE, MONITOR, KEYBOARD
Column 2	Column Header = Quantity Column Token = QUANTITY Validation = Numeric Text Field
Column 3	Column Header = Price Column Token = PRICE Validation = Numeric Text Field
Column 4	Column Header = Total Column Token = TOTAL Validation = Numeric Text Field



Name: Hardware Table

Description:

Enabled: ✓ Use in Workflow?

Component Type: Table Component

User Instructions: Select the Product and Quantity of the items you wish to order.

Meta Layer View: MREQ_ HARDWARE_TABLE

Column Seq.	Column Header	Column Token	Parameter Col.	Enabled	Component Type	Validation
1	Products	PRODUCTS	PARAMETER1	Y	Auto Complete List	Hardware Products
2	Quantity	QUANTITY	PARAMETER2	Y	Text Field	Text Field - 20
3	Price	PRICE	PARAMETER3	Y	Text Field	Text Field - 20
4	Total	TOTAL	PARAMETER4	Y	Text Field	Text Field - 20

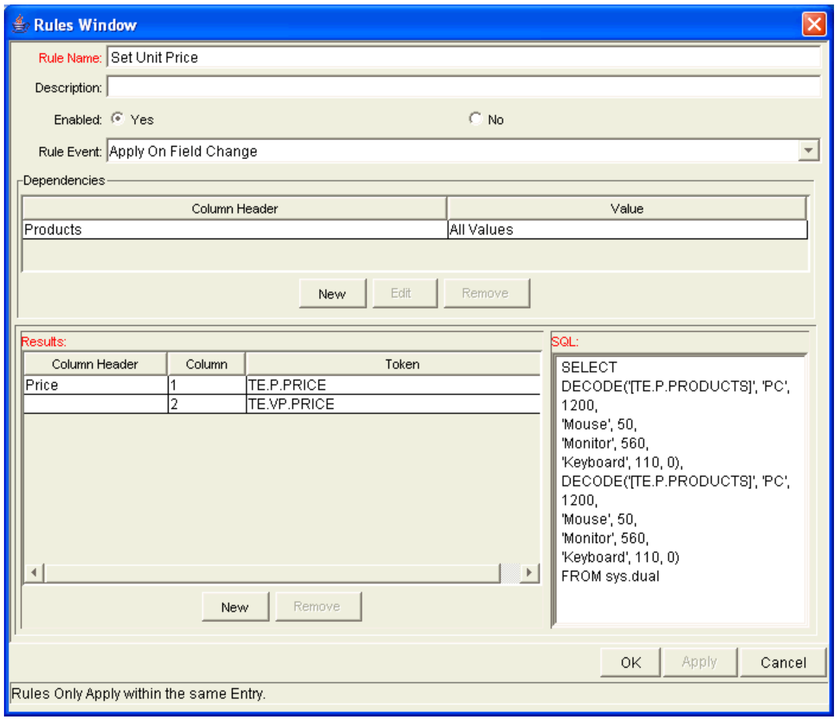
Once the validation's columns have been defined, the rules can be configured:

Rule 1: Set Unit Price.

ACME uses the following rule to set the default unit price in the Price cell based on the Product selection.

Table 5-22. Example - Set Unit Price rule settings

Setting	Value / Description
Rule Name	Set Unit Price
Rule Event	Apply on Field Change
Dependencies	Column = Products All Values = Yes
Results	Column Header = Price
SQL	SELECT DECODE('[TE.P.PRODUCTS]', 'PC', 1200, 'Mouse', 50, 'Monitor', 560, 'Keyboard', 110, 0), DECODE('[TE.P.PRODUCTS]', 'PC', 1200, 'Mouse', 50, 'Monitor', 560, 'Keyboard', 110, 0) FROM sys.dual



Rule Name: Set Unit Price

Description:

Enabled: [✓ Yes] No

Rule Event: Apply On Field Change

Dependencies

Column Header	Value
Products	All Values

Results:

Column Header	Column	Token
Price	1	TE.P.PRICE
	2	TE.VP.PRICE

SQL:

```
SELECT
DECODE('[TE.P.PRODUCTS]', 'PC',
1200,
'Mouse', 50,
'Monitor', 560,
'Keyboard', 110, 0),
DECODE('[TE.P.PRODUCTS]', 'PC',
1200,
'Mouse', 50,
'Monitor', 560,
'Keyboard', 110, 0)
FROM sys.dual
```

Rules Only Apply within the same Entry.

Rule 2: Calculate Total.

ACME uses the following rule to set the calculate and display the total line price in the Total column based on the values in the Products and Quantity cells.

Table 5-23. Example - Calculate Total rule settings

Setting	Value / Description
Rule Name	Calculate Total
Rule Event	Apply on Field Change

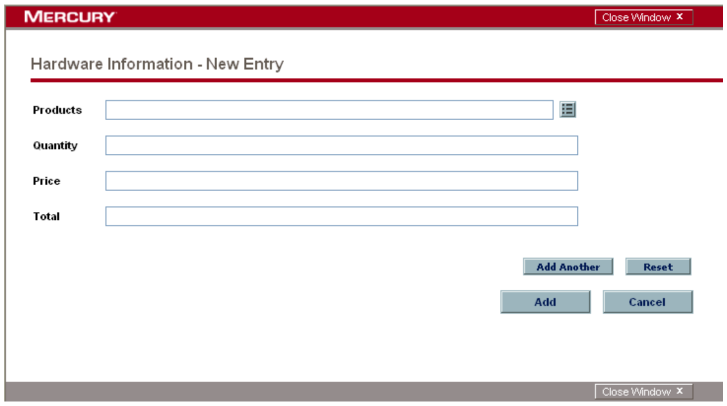
Dependencies	Column = Price [All Values = Yes]
	Column = Quantity [All Values = Yes]

Results	Column Header = Total

SQL	SELECT [TE.P.PRICE] * [TE.P.QUANTITY], [TE.P.PRICE] * [TE.P.QUANTITY]
	from sys.dual

Using the table component

Add a field to a request type that is validated by this table component validation. When a user opens the field to enter information, the table rules will be applied to each row that is created.



Tokens in the Table Components

Each column included in the table component has an associated token. These tokens can be used in the same manner as other field tokens, such as for commands, notifications or advanced field defaulting. See *Tokens in Request Table Components* on page 61 for details on referencing tokens related to table components.

Calculating Column Totals

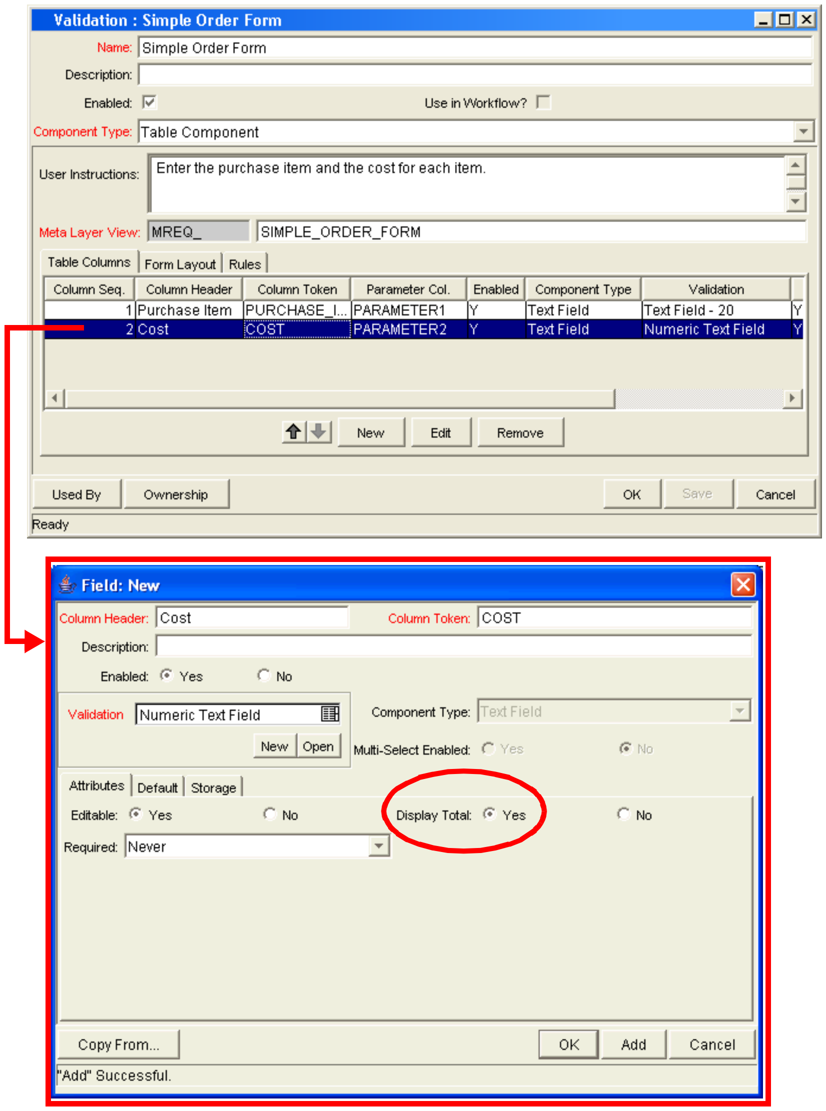
You can configure columns that are validated by a number to calculate the total for that column. This is configured in the validation's Field window. The following example illustrates how to configure a column to calculate and display the column total.

ACME, Inc., uses a request for creating and tracking simple employee equipment orders. ACME has included a table component field on their request type for gathering the order information. Employees enter the Purchase Items and Cost for each item. The table component automatically calculates the total cost for the Cost column.

ACME creates a validation with the following settings:

- Component Type = **Table Component**.
- Column 1 = Purchase Item (text field)

- Column 2 = Cost (number). In the Field window for the Cost column, select Display Total = **Yes.** The Display Total field is only enabled if the field's validation is a number.



Validation : Simple Order Form

Name: Simple Order Form
Description:
Enabled: ✓ Use in Workflow?
Component Type: Table Component
User Instructions: Enter the purchase item and the cost for each item.
Meta Layer View: MREQ_ SIMPLE_ORDER_FORM

Table Columns | Form Layout | Rules

Column Seq.	Column Header	Column Token	Parameter Col.	Enabled	Component Type	Validation	
1	Purchase Item	PURCHASE_I...	PARAMETER1	Y	Text Field	Text Field - 20	Y
2	Cost	COST	PARAMETER2	Y	Text Field	Numeric Text Field	Y

Ready

Field: New

Column Header: Cost Column Token: COST
Description:
Enabled: [✓ Yes] No
Validation: Numeric Text Field Component Type: Text Field
Multi-Select Enabled: Yes [✓ No]

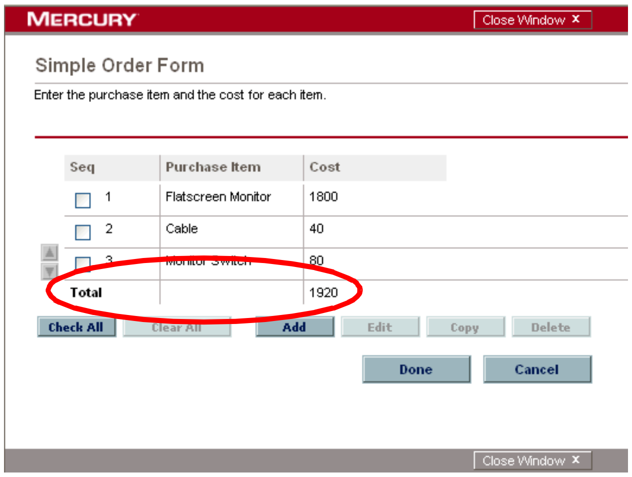
Attributes | Default | Storage

Editable: [✓ Yes] No Display Total: [✓ Yes] No
Required: Never

"Add" Successful.

Figure 5-18. Sample validation for a Simple Order table component.

ACME includes adds a field to their Order request type that uses this validation. When a user creates a request using that request type, he can click on the table component icon next to the field to open the order form. The total for the Cost column is displayed at the bottom of the table.



Simple Order Form

Enter the purchase item and the cost for each item.

Seq	Purchase Item	Cost
1	Flatscreen Monitor	1800
2	Cable	40
3	Monitor Switch	80
Total		1920

Figure 5-19. Sample table component displaying a column total.

Adding the Table Component to a Request Type

Table component fields can be included on a request type, request header type or request user data field.

To add a table component field to a request type:

1. Open the Request Type window.

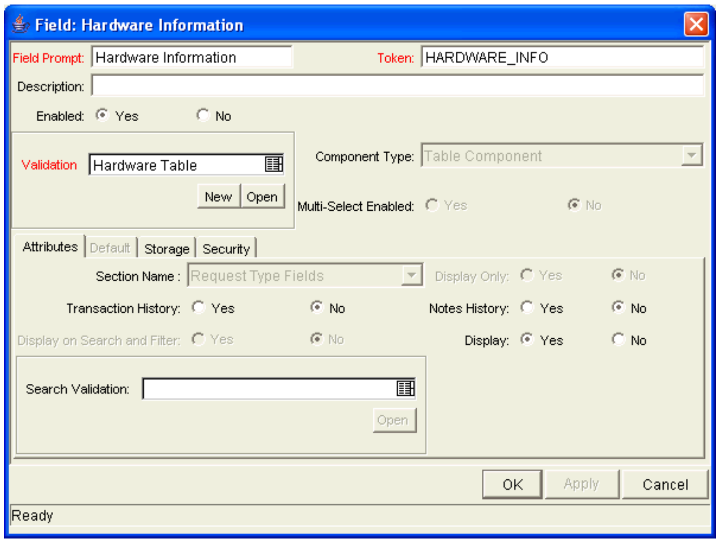
2. Click New in the **Fields** tab.

   The Field window opens.

3. Enter the Field Prompt, Token, and Description.

4. In the Validation field, select a table component validation.

   If you have not created a table component validation, click **New** to create one. See *Defining the Table Component in the Validation Workbench* on page 122 for instructions.


Field: Hardware Information

Field Prompt: Hardware Information
Token: HARDWARE_INFO
Description:
Enabled: [✓ Yes] No
Validation: Hardware Table
Component Type: Table Component
Multi-Select Enabled: Yes [✓ No]

Attributes | Default | Storage | Security

Section Name: Request Type Fields
Display Only: Yes [✓ No]
Transaction History: Yes [✓ No]
Notes History: Yes [✓ No]
Display on Search and Filter: Yes [✓ No]
Display: [✓ Yes] No
Search Validation:

Ready

5. Click **OK** to add the field to the request type.

6. Save the request type.

The table component field will now appear on requests of this request type.

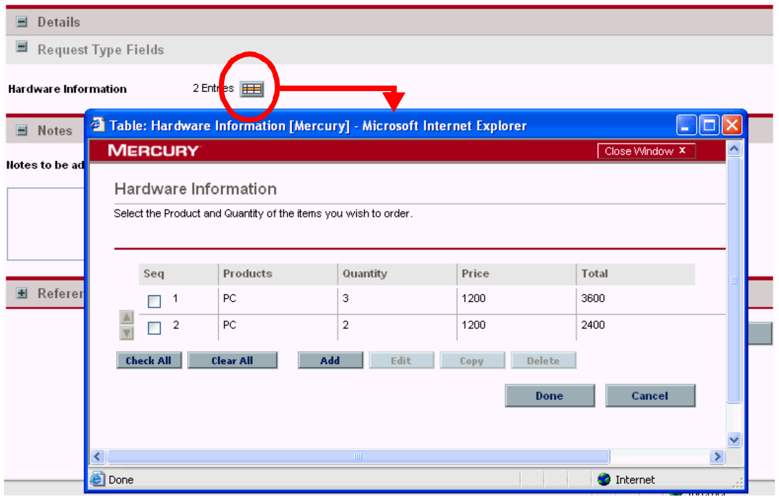

Details
Request Type Fields
Hardware Information 2 Entries
Notes
Notes to be ad
Referen

Table: Hardware Information [Mercury] - Microsoft Internet Explorer

Hardware Information

Select the Product and Quantity of the items you wish to order.

Seq	Products	Quantity	Price	Total
1	PC	3	1200	3600
2	PC	2	1200	2400

Done Internet

Package and Request Group Validations

Two particular entity-specific validations can be accessed in the Workbench without entering the Validations screen group.

Package and Request Groups

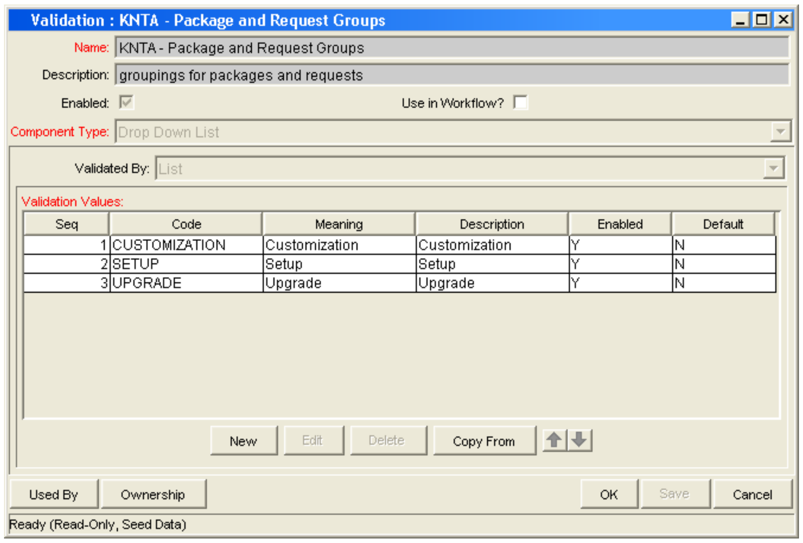
The KNTA-Package and Request Groups validation can be accessed directly from the **Package** screen. To specify that a package belongs to a new or unique package group that is not named in the auto-complete validation list, it is not necessary to proceed through the Validation Workbench.

To access the KNTA-Package and Request Groups validation window from the **Package** screen:

Select **New Package Group** from the **Package** menu. The Validation window will appear, listing the existing Mercury Change Management package groups.

Note

All users are granted read access to this screen, but only users with appropriate security privileges can alter the KNTA-Package and Request Groups validation list.



Name: KNTA - Package and Request Groups
Description: groupings for packages and requests
Enabled: ✓
Use in Workflow? ☐
Component Type: Drop Down List
Validated By: List

Validation Values:

Seq	Code	Meaning	Description	Enabled	Default
1	CUSTOMIZATION	Customization	Customization	Y	N
2	SETUP	Setup	Setup	Y	N
3	UPGRADE	Upgrade	Upgrade	Y	N

Ready (Read-Only, Seed Data)

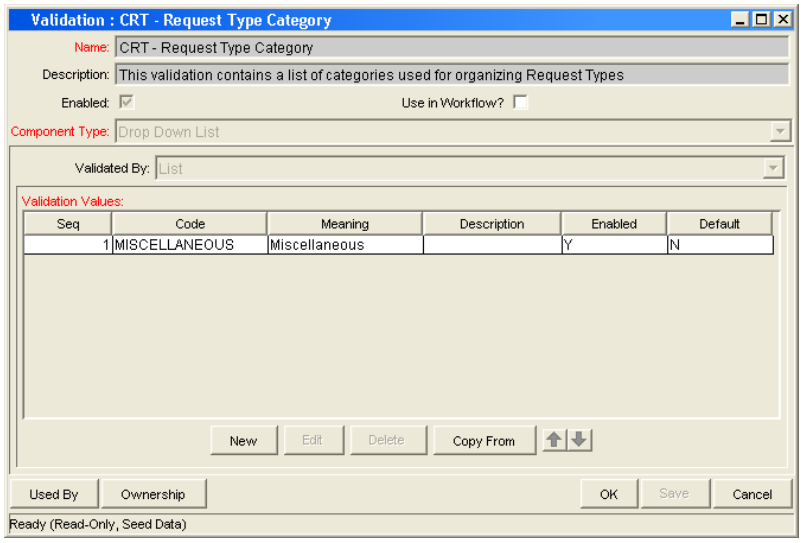
Request Type Category

The CRT - Request Type Category validation can be accessed directly from the **Request Types** workbench.

Access the CRT - Request Type Category validation window from the **Request Types** workbench by selecting **Request Type Category Setup** from the **Request Type** menu. The Validation window will appear, listing the existing request type categories.

Note

All users are granted read access to this screen, but only users with appropriate security privileges can alter the CRT - Request Type Category validation list.



Name: CRT - Request Type Category
Description: This validation contains a list of categories used for organizing Request Types
Enabled: ✓
Use in Workflow?
Component Type: Drop Down List
Validated By: List

Validation Values:

Seq	Code	Meaning	Description	Enabled	Default
1	MISCELLANEOUS	Miscellaneous		Y	N

Ready (Read-Only, Seed Data)

Validation Special Characters

The Validation Name field for all validations cannot contain a question mark (‘?’). The Workbench prevents this character from being entered into the field, but all previously configured validation names (validations entered before release 4.5) should be checked and corrected.

System Validations

There are a number of validations that are provided with Mercury IT Governance Center. Note that many of these validations may have been altered to better match your company's specific business needs. Use the Validations report to get a list of all validations currently in your system. The report includes information on validation values and commands.

Appendix A
System Special Commands

In This Appendix:

- *Overview of System Special Commands*
- *ksc_connect Special Commands*
  - *ksc_connect_dest_client*
  - *ksc_connect_dest_server*
  - *ksc_connect_source_client*
  - *ksc_connect_source_server*
- *ksc_exit*
- *ksc_copy Special Commands*
  - *ksc_copy_client_client*
  - *ksc_copy_client_server*
  - *ksc_copy_server_client*
  - *ksc_copy_server_server*
  - *ksc_copy_client_tmp*
  - *ksc_copy_server_tmp*
  - *ksc_copy_tmp_client*
  - *ksc_copy_tmp_server*
- *ksc_respond*
- *ksc_simple_respond*
  - *Examples using ksc_simple_respond*
- *ksc_local_exec*
  - *Example using ksc_local_exec*
- *ksc_replace*
  - *Example using ksc_replace*
- *ksc_set*
  - *Example using ksc_set*

- *ksc_set_env*
- *ksc_store*
  - *Example using ksc_store*
- *ksc_comment*
- *ksc_concsub*
  - *Example using ksc_concsub*
- *ksc_begin_script / ksc_end_script*
  - *Example using ksc_begin_script and ksc_end_script*
- *ksc_copy_script Special Commands*
  - *ksc_copy_script_dest_client*
  - *ksc_copy_script_dest_server*
  - *ksc_copy_script_source_client*
  - *ksc_copy_script_source_server*
- *ksc_om_migrate*
  - *Example using ksc_om_migrate*
- *ksc_capture_output*
- *ksc_gl_migrate*
  - *Example using ksc_gl_migrate*
- *ksc_parse_jcl*
- *ksc_submit_job*
- *ksc_set_exit_value*
- *ksc_clear_exit_value*
- *ksc_run_sql*
  - *Example using ksc_run_sql*
- *Summary of All Special Command Parameters*

Overview of System Special Commands

This appendix discusses the pre-defined special commands.

ksc_connect Special Commands

The ksc_connect special commands instruct the execution engine to open a connection to a specified environment. This command initiates a TELNET, SSH or SSH2 session with the server or client defined for the environment.

The command then sends all command steps that follow it directly to the machine, as though someone was actually typing the command on that machine. In this way, the execution engine is able to run virtually any command-line directive that the machine understands.

Note

All ksc_connect special commands must end with the 'ksc_exit' special command to exit the TELNET, SSH, or SSH2 session.

ksc_connect_dest_client

This command initiates a TELNET, SSH, or SSH2 session with the client of the destination environment. The destination environment refers to the destination environment of the workflow step initiating command execution.

Table A-1. ksc_connect_dest_client parameters

Parameter	Default Token	Description
USERNAME	[DEST_ENV.CLIENT_ USERNAME	Username on [DEST_ENV].
PASSWORD	[DEST_ENV.CLIENT_ PASSWORD	Password on [DEST_ENV].
NT_DOMAIN	[DEST_ENV.CLIENT_NT_ DOMAIN	Windows NT Domain name of [DEST_ENV].
DEST_BASE_ PATH	[DEST_ENV.CLIENT_ BASE_PATH	Base Path of [DEST_ENV].
CONNECTION_ PROTOCOL	[DEST_ENV.CLIENT_ CON_PROTOCOL_ MEANING]	Specifies the connection protocol. Possible values are listed in validation "CONNECTION_PROTOCOL".
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example Using ksc_connect_dest_client

```
# Make a remote connection to the client of the
# destination environment defined for the current
# workflow step.
```

```
ksc_connect_dest_client
<commands>
ksc_exit

# Make a remote connection to the client defined for
# the environment named 'STAGING'.

ksc_connect_dest_client DEST_ENV="STAGING"
<commands>
ksc_exit
```

ksc_connect_dest_server

This command initiates a TELNET, SSH, or SSH2 session with the server of the destination environment. The destination environment refers to the destination environment of the workflow step initiating command execution.

Table A-2. ksc_connect_dest_server parameters

Parameter	Default Token	Description
USERNAME	[DEST_ENV.SERVER_ USERNAME	Username on [DEST_ENV].
PASSWORD	[DEST_ENV.SERVER_ PASSWORD	Password on [DEST_ENV].
NT_DOMAIN	[DEST_ENV.SERVER_ NT_DOMAIN	Windows NT Domain name of [DEST_ENV].
DEST_BASE_ PATH	[DEST_ENV.SERVER_ BASE_PATH	Base Path of [DEST_ENV].
CONNECTION_ PROTOCOL	[DEST_ENV.SERVER_ CON_PROTOCOL_ MEANING]	Specifies the connection protocol. Possible values are listed in validation "CONNECTION_ PROTOCOL".
DEST_ENV	[DEST_ENV. ENVIRONMENT_NAME]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example using ksc_connect_dest_server

```
# Make a remote connection to the server of the
# destination environment defined for the current
# workflow step.

ksc_connect_dest_server
<commands>
```

```
ksc_exit

# Make a remote connection to the server defined for
# the environment named 'Staging'.

ksc_connect_dest_server DEST_ENV="STAGING"
<commands>
ksc_exit
```

ksc_connect_source_client

This command initiates a TELNET, SSH, or SSH2 session with the client of the source environment. The source environment refers to the source environment of the workflow step initiating command execution.

Table A-3. ksc_connect_source_client parameters

Parameter	Default Token	Description
USERNAME	[SOURCE_ENV.CLIENT_USERNAME	Username on [SOURCE_ENV].
PASSWORD	[SOURCE_ENV.CLIENT_PASSWORD	Password on [SOURCE_ENV].
NT_DOMAIN	[SOURCE_ENV.CLIENT_NT_DOMAIN	Windows NT Domain name of [SOURCE_ENV].
SOURCE_BASE_PATH	[SOURCE_ENV.CLIENT_BASE_PATH	Base Path of [SOURCE_ENV].
CONNECTION_PROTOCOL	[SOURCE_ENV.CLIENT_CON_PROTOCOL_MEANING]	Specifies the connection protocol. Possible values are listed in validation "CONNECTION_PROTOCOL".
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.

Example using ksc_connect_source_client

```
# Make a remote connection to the client of the source
# environment defined for the current workflow step.

ksc_connect_source_client
<commands>
ksc_exit

# Make a remote connection to the client defined for
# the environment named 'STAGING'.
```

```
ksc_connect_source_client SOURCE_ENV="STAGING"
<commands>
ksc_exit
```

ksc_connect_source_server

This command initiates a TELNET, SSH, or SSH2 session with the server of the source environment. The source environment refers to the source environment of the workflow step initiating command execution.

Table A-4. ksc_connect_source_server parameters

Parameter	Default Token	Description
USERNAME	[SOURCE_ENV.SERVER_ USERNAME	Username on [SOURCE_ENV].
PASSWORD	[SOURCE_ENV.SERVER_ PASSWORD	Password on [SOURCE_ENV].
NT_DOMAIN	[SOURCE_ENV.SERVER_ NT_DOMAIN	Windows NT Domain name of [SOURCE_ENV].
SOURCE_ BASE_PATH	[SOURCE_ENV.SERVER_ BASE_PATH	Base Path of [SOURCE_ENV].
CONNECTION_ PROTOCOL	[SOURCE_ENV.SERVER_ CON_PROTOCOL_ MEANING]	Specifies the connection protocol. Possible values are listed in validation “CONNECTION_PROTOCOL”.
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.

Examples using ksc_connect_source_server

```
# Make a remote connection to the server of the source
# environment defined for the current workflow step.

ksc_connect_source_server
<commands>
ksc_exit

# Make a remote connection to the server defined for
# the environment named ‘STAGING’.

ksc_connect_source_server SOURCE_ENV="STAGING"
<commands>
ksc_exit
```

ksc_exit

This command exits the TELNET, SSH, or SSH2 session initiated by the *ksc_connect Special Commands*. For examples using ksc_exit, see *ksc_connect Special Commands*.

ksc_copy Special Commands

The ksc_copy special commands provide the mechanism for transferring files to and from the various environments defined in Mercury IT Governance Center.

Note

The default use of these commands requires that the entity containing the command has three fields with the following tokens defined:

```
[P.P_FILENAME]
[P.P_FILE_TYPE]
[P.P_SUB_PATH]
```

If these fields are not defined as part of the entity, they must be passed as parameters or the command will fail.

Files are copied using either FTP, SCP or SCP2, depending on the configuration of the environment.

ksc_copy_client_client

This command copies a file from the source client environment to the destination client environment.

Table A-5. ksc_copy_client_client parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV. CLIENT_BASE_PATH]	The base path of the source client environment to be used instead of what is defined for the current source environment.

Table A-5. ksc_copy_client_client parameters

Parameter	Default Token	Description
DEST_BASE _PATH	[DEST_ENV. CLIENT_BASE_PATH]	The base path of the destination client environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example #1 using ksc_copy_client_client

```
# Copy a file between source and destination clients.

ksc_copy_client_client SUB_PATH="forms"
  FILENAME="[P.P_MODULE].fmb" FILE_TYPE="BINARY"

# Copy a file between the client defined in the ‘STAGING’
# environment and the destination client.

ksc_copy_client_client DEST_ENV=”STAGING”
```

Example #2 using ksc_copy_client_client

```
# Override the base path of the destination directory.

ksc_copy_client_client DEST_BASE_PATH=”/u1/datatree/ex1” SUB_
PATH=”.” FILENAME="[P.P_MODULE].fmb" FILE_TYPE="BINARY"
```

ksc_copy_client_server

This command copies a file from the source client environment to the destination server environment.

Table A-6. ksc_copy_client_server parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV.CLIENT_ BASE_PATH]	The base path of the source client environment to be used instead of what is defined for the current source environment.
DEST_BASE _PATH	[DEST_ENV. SERVER_BASE_PATH]	The base path of the destination server environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example using ksc_copy_client_server

```
# Copy a file between source client and
# destination server.

ksc_copy_client_server SUB_PATH="install/sql"
  FILENAME="[P.P_SQL_SCRIPT]" FILE_TYPE="ASCII"
```

ksc_copy_server_client

This command copies a file from the source server environment to the destination client environment.

Table A-7. ksc_copy_server_client parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV. SERVER_BASE_PATH]	The base path of the source server environment to be used instead of what is defined for the current source environment.
DEST_BASE _PATH	[DEST_ENV. CLIENT_BASE_PATH]	The base path of the destination client environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example using ksc_copy_server_client

```
# Copy a file between source server and
# destination client.

ksc_copy_server_client SUB_PATH="[P.P_SUB_DIRECTORY]"
 FILE_TYPE="[P.P_FILE_TYPE]"
```

ksc_copy_server_server

This command copies a file from the source server environment to the destination server environment.

Table A-8. ksc_copy_server_server parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV. SERVER_BASE_PATH]	The base path of the source server environment to be used instead of what is defined for the current source environment.
DEST_BASE _PATH	[DEST_ENV. SERVER_BASE_PATH]	The base path of the destination server environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

Example using ksc_copy_server_server

```
# Copy a file between source and destination servers.
ksc_copy_server_server FILENAME="[P.P_FILE]"

# Copy a file between the source server and the
# destination server overriding the base bath.

ksc_copy_server_server FILENAME=”install_driver.sh”
                              DEST_BASE_PATH=”/u2/app/drivers”

# Copy a form between the ‘STAGING’ and destination servers.

ksc_copy_server_server SOURCE_ENV="STAGING" SUB_PATH="forms"
  FILENAME="[P.P_MODULE].fmb" FILE_TYPE="BINARY"
```

ksc_copy_client_tmp

This command copies a file from the source client environment to the temporary package transfer directory on the application server. This temporary directory is automatically cleaned up after an execution completes and can be referenced using the [AS.PKG_TRANSFER_PATH] token.

Table A-9. ksc_copy_server_tmp parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV. CLIENT_BASE_PATH]	The base path of the source client environment to be used instead of what is defined for the current source environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.

ksc_copy_server_tmp

This command copies a file from the source server environment to the temporary package transfer directory on the application server. This temporary directory is automatically cleaned up after an execution completes and can be referenced using the [AS.PKG_TRANSFER_PATH] token.

Table A-10. ksc_copy_server_tmp parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
SOURCE_BASE _PATH	[SOURCE_ENV. SERVER_BASE_PATH]	The base path of the source server environment to be used instead of what is defined for the current source environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.

ksc_copy_tmp_client

This command copies a file from the temporary package transfer directory on the application server to the destination client environment. This temporary directory is automatically cleaned up after an execution completes and can be referenced using the [AS.PKG_TRANSFER_PATH] token.

Table A-11. ksc_copy_server_tmp parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
DEST_BASE _PATH	[DEST_ENV. CLIENT_BASE_PATH]	The base path of the destination server environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.

Table A-11. ksc_copy_server_tmp parameters

Parameter	Default Token	Description
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

ksc_copy_tmp_server

This command copies a file from the temporary package transfer directory on the application server to the destination server environment. This temporary directory is automatically cleaned up after an execution completes and can be referenced using the [AS.PKG_TRANSFER_PATH] token.

Table A-12. ksc_copy_server_tmp parameters

Parameter	Default Token	Description
SUB_PATH	[P.P_SUB_PATH]	The sub-directory that should be used to locate the file relative to the base path of each environment.
DEST_BASE _PATH	[DEST_ENV. SERVER_BASE_PATH]	The base path of the destination server environment to be used instead of what is defined for the current destination environment.
FILENAME	[P.P_FILENAME]	Name of the file to be copied.
FILE_TYPE	[P.P_FILE_TYPE]	The file type associated with the file (ASCII or BINARY).
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

ksc_respond

This command is currently only used to support Patch*Applicator. This command is able to intelligently respond to interactive prompts generated by the Oracle “adpatch” and “adadmin” programs. General use of this special command for arbitrary programs is not yet supported. For simple interactive programs, see *ksc_simple_respond* on page 151.

ksc_simple_respond

This command executes an interactive UNIX command on a remote computer. This command is useful when the command to be executed will prompt for additional information (such as the UNIX ‘su’ command to switch user accounts) or may not return an exit code upon completion (such as starting up a new shell using ‘sh’).

Note

This command can only be used from within a remote execution session, such as between ‘ksc_connect’ and ‘ksc_exit’ commands.

The following syntax is supported:

```
ksc_simple_respond “command”
ksc_simple_respond “command” “prompt 1" “response 1" [“prompt 
2" “response 2" … ]
ksc_simple_respond “command” -hide “prompt 1" “response 1" 
[“prompt 2" “response 2" … ]
```

There can be as many prompt-response pairs as necessary. Each prompt must be matched with a response, even if the response is an empty string. The prompts must appear in the exact order they will be displayed as the command is run. All arguments must be enclosed in quotes. In addition, if the command or any of the arguments contains double quotes (“), any other character can be used as the quote character. The first character after the string ‘ksc_simple_ respond’ will be interpreted as the quote character, and that character must appear at the beginning and end of each argument.

By using the -hide option, the value passed in for the response will not be displayed in the execution log. In the log, the value will be displayed as ****. This flag should be used for each prompt/response pair that needs this treatment.

Note

The execution engine will wait for each specified prompt. If a prompt does not appear for some reason, then the execution engine will continue to wait for it until the command times out.

Examples using ksc_simple_respond

If it becomes necessary to invoke a new shell while in a remote session, it would be ideal to simply use the command ‘sh’. However, this can cause the execution engine to wait indefinitely while waiting for an exit code. To avoid this problem, the ‘sh’ command can be encapsulated in a ksc_simple_respond command with no prompts as shown:

```
ksc_simple_respond “sh”
```

As another example, suppose it becomes necessary to switch to another user account while in a remote session using the ‘su’ command. This command always prompts for password, unless performed by a root user. By utilizing the -hide feature, the password will not be displayed in the execution logs. This interactivity can be handled using ksc_simple_respond as follows:

```
ksc_simple_respond "su <username>" -hide "word:" "<password>"
```

Note that “word:” was used as the prompt instead of the entire word “password:”. The execution engine will wait for the specified prompt string, whether it is all—or just a part—of the prompt text.

As one more example, consider the following Bourne shell command:

```
echo "Enter a string:\c"; read str; echo $str
```

Normally, this command line would cause the execution engine to hang while waiting for an exit code (the command will never exit because it is waiting for input), which would eventually timeout when the execution timeout time is reached. Use ksc_simple_respond to process this command as shown (this command should be entered on a single line):

```
ksc_simple_respond #echo "Enter a string:\c"; read str; echo
$str# #a string:# #my_value#
```

Since the command line contained double quotes, the pound sign (#) is used as the quote character. During execution, this command step will prompt “Enter a string:” and wait for input. The string “my_value” would be entered automatically, this value will then be echoed to the output device (in this case, the execution log), and execution will continue as normal with the next command step.

ksc_local_exec

This command invokes a local process on the machine running the Mercury IT Governance Server. It can be used to run any program that does not require interactive input. Each call using 'ksc_local_exec' is an independent process. It does not execute in the context of other commands that precede it. The starting directory for the processes generated using 'ksc_local_exec' is the home directory of the Mercury IT Governance Server. Full paths to the executable being called are necessary if the Mercury IT Governance Server does not have the correct system path information.

Note

The ksc_local_exec command does not open a TELNET, SSH or SSH2 connection to the Mercury IT Governance Server. It operates by creating a new child process on the machine that is running the Mercury IT Governance Server. Therefore, the user account and password for this process will be the same as the account and password used to start the Mercury IT Governance Server.

Example using ksc_local_exec

```
# Rename existing file 'file.txt' to 'newfile.txt'
ksc_local_exec mv file.txt newfile.txt
```

```
# Run a DOS batch file
ksc_local_exec cmd /c runme.bat
```

System commands do not invoke either Unix shells or DOS shells. This means that the following code segment using 'ksc_local_exec' is not valid, because it cannot use the 'pipe' (|) or redirect commands (>):

```
ksc_local_exec cat names.txt | grep address > file.out
```

An effective way to use the ksc_local_exec command is to put a series of commands into a .sh file, and then execute the .sh file as shown:

```
ksc_begin_script + [AS.CR_TRANSFER_PATH] run.sh
..
```

```
<series of commands>
ksc_end_script

ksc_local_exe ksh run.sh
```

ksc_replace

This command is used to edit the contents of a file and place it into another file. This command works in a way similar to the ‘sed’ utility and supports the same substituting expressions.

The files must be located on the Mercury IT Governance Server in the [AS.PKG_TRANSFER_PATH] directory. This requires the use of the ksc_copy_tmp_* commands.

Table A-13. ksc_replace parameters

Parameter	Default Token	Description
FILENAME	[P.P_FILENAME]	Name of the source file to be edited.
OUTFILE	[OUTFILE]	Name of the output file after applying the substitution expressions.
SUBST		The substitution expression.

Example using ksc_replace

```
ksc_copy_server_tmp FILENAME="config.template" FILE_
TYPE="ASCII"

ksc_replace FILENAME="config.template" OUTFILE="config.cfg"
SUBST="s/NAME/[P.NAME]/g"

ksc_copy_tmp_server FILENAME="config.cfg"
```

ksc_set

This command sets the value of a temporary variable which may be used to manage command conditions or aid in command processing.

The following syntax is supported:

```
ksc_set VARIABLE="Value"
```

To reference the value of this variable, use the familiar token syntax without any prefix. Unlike the ‘ksc_store’ command, ‘ksc_set’ does not write values to the database. The scope of the variable that is set is valid from when the variable is defined to the end of the command steps for the entity. This make using ‘ksc_set’ more attractive than shell variables because the values are retained between separate ‘ksc_connect’ sessions. Another advantage of using ‘ksc_set’ is that the token values are visible in the logs, not just the variable names. This command may be nested within a ‘ksc_connect’ command (see the following example).

Example using ksc_set

```
# Set the value of a compile flag.
#
ksc_set COMPILE="YES"
# ksc_set nested within a ksc_connect
ksc_connect_dest_server
ksc_set REBUILD="NO"
ksc_exit
```

Later, a temporary variable can be referenced in a command condition or in another command step. For example, the command condition may look like:

```
'[COMPILE]' = 'YES'
```

ksc_set_env

Use this command to set the correct environment context of an execution in cases where the workflow source and destination environments are overridden using the DEST_ENV and SOURCE_ENV parameters. Normally it is not necessary to use this command since it is called internally from other special commands. If it is used on a stand-alone basis, it must come after any ‘ksc_ copy’ commands.

Table A-14. ksc_set_env parameters

Parameter	Default Token	Description
DEST_ENV_ID	[DEST_ENV. ENVIRONMENT_ID]	ID of the destination environment to be used instead of the destination environment on the current workflow step.
SOURCE_ENV_ID	[SOURCE_ENV. ENVIRONMENT_ID]	ID of the source environment to be used instead of the source environment on the current workflow step.
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

ksc_store

This command dynamically sets the values of fields defined for object types, request types, and report types. This command is useful to set or alter the value of fields based on the command output. This command may only be used on fields which have been custom configured. Custom configured fields are those with tokens that are evaluated using the [P.<*TOKEN*>] or [VP.<*TOKEN*>] format. After altering a token, future evaluations of the token will use the new value. The new values are written to the database, so the changes are not temporary as in ‘ksc_set’.

This command may be nested within a ‘ksc_connect’ command (as seen in the following example) and its value can be referenced in command conditions.

The following syntax is supported:

```
ksc_store TOKEN=”Value”
ksc_store TOKEN=”Hidden Value”, “Visible Value”
```

In the first case, the hidden and visible values of the field will be set to the same value. In the second case, the hidden and visible values are set independently. “Hidden Value” refers to the [P.<*TOKEN*>] format. “Visible Value” refers to the [VP.<*TOKEN*>] format.

Example using ksc_store

In the following example, it is assumed that the entity in question has the following tokens defined:

```
[P.DRIVER]
[P.REVISION]
[P.RESULT]

# Store the name of the driver file.

ksc_store DRIVER=”driver.sh”

# Capture the Revision number of a file.
#
ksc_connect_dest_server
cd “SourceCode/java”
grep ‘$Revision’ ServerAdmin.java
ksc_store REVISION=”[EXEC.OUTPUT]”
ksc_exit

# Set the hidden and visible result codes of a parameter.
#
ksc_store RESULT=”IN_PROG”,”In Progress”
```

ksc_comment

This command adds single line comments to the execution log. It can be used to indicate informational or error messages. HTML tags are supported.

The following syntax is supported:

```
ksc_comment <comment>
```

The comment text can be any text string.

ksc_concsub

This command submits Oracle Application concurrent requests from the operating system command line. It is treated as a special command because the command engine must capture the concurrent request ID, which is an output of successful submission. To work properly, this command must be called within a ‘ksc_connect - ksc_exit’ command block.

If ‘ksc_concsub’ is used to submit a concurrent request to an Oracle Applications database other than where Mercury IT Governance Center is currently installed, the ORA_APPS_DB_LINK parameter must be added to the ‘ksc_concsub’ command. Otherwise, the status of the concurrent request cannot be determined after submission.

The following syntax is supported:

```
ksc_concsub ORA_APPS_DB_LINK="DB_LINK" CONCSUB
```

DB_LINK corresponds to the database link from the Mercury IT Governance Center schema to the APPS schema of the database to which the concurrent request is submitted.

Example using ksc_concsub

```
ksc_concsub ORA_APPS_DB_LINK=[DEST_ENV.ORA_APPS_DB_LINK]
CONCSUB [DEST_ENV.APP.DB_USERNAME]/[DEST_ENV.APP.DB_
PASSWORD]@[DEST_ENV.DB_CONNECT_STRING]  FND  'Application
Developer' SYSADMIN WAIT=N CONCURRENT FND FNDFMREG [DEST_
ENV.APP_CODE] [P.P_FILENAME]
```

Note

The special command ‘ksc_concsub’ is followed by the exact CONCSUB call that will be executed directly at the command line.

The complete syntax for Oracle's CONCSUB is shown below. Optional parameters are in square brackets.

```
CONCSUB
<ORACLE ID>
<Responsibility Application Short Name>
<Responsibility Name>
<User Name>
[WAIT=N]
CONCURRENT
<Concurrent Program Application Short Name>
```

```
<Concurrent Program Name>
[START=<Requested Start Date>]
[REPEAT_DAYS=<Repeat Interval>]
[REPEAT_END=<Request Resubmission End Date>]
<Concurrent Program Arguments...>
```

For additional information on using the CONCSUB command, see the Oracle documentation.

Note

It is not possible to retrieve the concurrent request logs from a ‘ksc_concsub’ submission submitted against a remote database.

ksc_begin_script / ksc_end_script

The object command structure of Mercury IT Governance Center lends itself nicely to standard, step-by-step processes. In most cases, these commands are fully capable of automating the migration of an object. However, in some circumstances, it is necessary to add additional logic to the commands for an object. For example, perhaps a loop must be generated to repeat a command several times. This is where scripts-on-the-fly are best applied.

Scripts-on-the-fly are designed to leverage the architecture, tools, and knowledge already present in an organization. By using a script-on-the-fly, administrators can define migration logic in their preferred scripting language (such as Bourne Shell, C Shell or Perl). The scripts only need to be defined once. The execution engine copies the script wherever it needs to be executed. The execution engine can also be instructed to clean up the script after it has been executed, leaving no traces behind.

The following syntax is supported:

```
ksc_begin_script <full_path_to_file_to_be_generated>
<directives from any scripting language>
ksc_end_script
```

It is commonly used in the following format:

```
ksc_begin_script [AS.PKG_TRANSFER_PATH][P.P_SCRIPT_FILENAME]
```

Since the script will be generated into a temporary directory by use of the [AS.PKG_TRANSFER_PATH] token, this token will reference a unique temporary directory per execution and end with the proper directory slash ‘/’ or ‘\’. After generation, the script can be transferred to another machine for execution using the ‘ksc_copy_script’ commands described in ksc_copy_script Special Commands.

Example using ksc_begin_script and ksc_end_script

```
ksc_begin_script [AS.PKG_TRANSFER_PATH][P.P_SCRIPT_FILENAME]
#!/usr/bin/csh
#
# Script to lock, check in, and re-checkout the original
# file using RCS commands.
#
# Print a warning if the file does not exist.
#

if ($#argv != 2) then
        echo "$0 : wrong number of arguments"
        echo "Usage: $0 sub_path filename"
        exit 1
endif

set sub_path = $argv[1]
set filename = $argv[2]

if (-e "$sub_path/RCS/$filename,v") then
        rcs -l $sub_path/$filename
        ci -m"Before Copy." $sub_path/$filename
        co -l $sub_path/$filename
else
       echo "Warning: File $sub_path/$filename not found in RCS
repository"
endif

exit 0
ksc_end_script

# Copy the script to the destination server and excute it.
ksc_copy_script_dest_server
ksc_connect_dest_server
csh [P.P_SCRIPT_FILENAME]
rm [P.P_SCRIPT_FILENAME]
ksc_exit
```

ksc_copy_script Special Commands

Use these special commands to transfer files from the temporary file transfer directory (defined by token [AS.PKG_TRANSFER_PATH]) to other machines. These commands are typically used in conjuction with the ‘ksc_ begin_script' and 'ksc_end_script' commands, but can also be used in other ways.

ksc_copy_script_dest_client

This command copies a script contained in [AS.PKG_TRANSFER_PATH], a temporary directory located on the Mercury IT Governance Server, to the base path of the destination client environment.

Table A-15. ksc_copy_script_dest_client parameters

Parameters	Default Token	Description
SCRIPT _FILENAME	[P.P_SCRIPT_FILENAME]	The name of the script file to transfer.
DEST_BASE _PATH	[DEST_ENV. CLIENT_BASE_PATH]	The base path of the destination client environment to be used instead of what is defined for the current destination environment.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

ksc_copy_script_dest_server

This command copies a script contained in [AS.PKG_TRANSFER_PATH], a temporary directory located on the Mercury IT Governance Server, to the base path of the destination server environment.

Table A-16. ksc_copy_script_dest_server parameters

Parameters	Default Token	Description
SCRIPT _FILENAME	[P.P_SCRIPT_FILENAME]	The name of the script file to transfer.

Table A-16. ksc_copy_script_dest_server parameters

Parameters	Default Token	Description
DEST_BASE _PATH	[DEST_ENV. SERVER_BASE_PATH]	The base path of the destination server environment to be used instead of what is defined for the current destination environment.
DEST_ENV	[DEST_ENV]	Name of the destination environment to be used instead of the destination environment on the current workflow step.

ksc_copy_script_source_client

This command copies a script contained in [AS.PKG_TRANSFER_PATH], a temporary directory located on the Mercury IT Governance Server, to the base path of the source client environment.

Table A-17. ksc_copy_script_source_client parameters

Parameters	Default Token	Description
SCRIPT _FILENAME	[P.P_SCRIPT_FILENAME]	The name of the script file to transfer.
DEST_BASE _PATH	[SOURCE_ENV. CLIENT_BASE_PATH]	The base path of the source client environment to be used instead of what is defined for the current source environment.
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the destination environment on the current workflow step.

ksc_copy_script_source_server

This command copies a script contained in [AS.PKG_TRANSFER_PATH], a temporary directory located on the Mercury IT Governance Server, to the base path of the source server environment.

Table A-18. ksc_copy_script_source_client parameters

Parameters	Default Token	Description
SCRIPT _FILENAME	[P.P_SCRIPT_FILENAME]	The name of the script file to transfer.
SOURCE_BASE _PATH	[SOURCE_ENV. SERVER_BASE_PATH]	The base path of the source server environment to be used instead of what is defined for the current source environment.
SOURCE_ENV	[SOURCE_ENV]	Name of the source environment to be used instead of the source environment on the current workflow step.

ksc_om_migrate

Use this command to launch migrations supported by the Object*Migrator.

The following syntax is supported:

```
ksc_om_migrate CONC_PROGRAM=<conc_program_name>
APP_SHORT_NAME=<APP_SHORT_NAME> OM_ARCHIVE_FLAG=<Y/N>
```

The parameters CONC_PROGRAM and APP_SHORT_NAME are required. All other parameters are optional and are used to override the default behavior.

Table A-19. ksc_om_migrate parameters

Parameter	Default Token	Description
CONC _PROGRAM	None. This is a mandatory parameter.	The concurrent program name. This has been pre-configured and will not need to be modified.
OM_ARCHIVE _FLAG	[WFS. OM_ARCHIVE_FLAG]	Specifies whether the migration will store to the archive rather than using what has been specified for the current workflow step.
APP_SHORT _NAME	None. This is a required parameter.	This value is normally “CLM” but can be modified if the Object*Migrator has been installed into a custom account.

Table A-19. ksc_om_migrate parameters

Parameter	Default Token	Description
SOURCE _ENV	[SOURCE_ENV]	The environment to migrate from rather than the one defined on the workflow step.
DEST _ENV	[DEST_ENV]	The environment to migrate to rather than the one defined on the workflow step.

Example using ksc_om_migrate

```
#
#Launch an AOL Concurrent Program Migration
#
ksc_om_migrate CONC_PROGRAM="CLMRMCP1" APP_SHORT_NAME="CLM"
```

ksc_capture_output

The ‘ksc_capture_output’ special command is only used in validations. It is used to get data from an alternate source, and use that data to populate an auto-complete field. This functionality provides additional flexibility when designing auto-complete lists.

Many enterprises have found that they need to use alternate sources of data within their applications. Examples of these sources might be a flat file, an alternate database source, or output from a command line execution. The ‘ksc_capture_output’ command may be used in conjunction with these alternate data sources, in the context of a validation, to provide a list of values on the fly.

The syntax for the ‘ksc_capture_output’ is:

```
ksc_capture_output <command>
```

In the Validation Workbench, under Validated By, choose either **Command With Delimited Output** or **Command With Fixed Width Output** and input the delimiting character or field length information. Then, under New Command, enter the steps. The example below would put the validations into the address.txt file, then run the ‘ksc_capture_output’ against the address.txt file:

```
ksc_begin_script[AS.PKG_TRANSFER_PATH]address.txt
```

```
street
city
state
zipcode
ksc_end_script
ksc_capture_output cat[AS.PKG_TRANSFER_PATH]address.txt
```

In this case, the entire sequence of commands would be executed on the local machine where the Mercury IT Governance Server is running. This is the preferred method of invoking ‘ksc_capture_output’. The ‘ksc_capture_output’ command may be embedded between ‘ksc_connect’ and ‘ksc_exit’ commands, but the time delay is significant depending on network load (because the validation actually requires an entire TELNET, SSH or SSH2 session to be generated to the remote machine). It is recommended that ‘ksc_capture_output’ only be used in a local execution scenario.

‘ksc_capture_output’ may be called more than once. Each call will append the results to the previous call.

ksc_gl_migrate

Use this command to launch migrations supported by the GL *Migrator.

The following syntax is supported:

```
ksc_gl_migrate CONC_PROGRAM=<conc_program_name>
APP_SHORT_NAME=<APP_SHORT_NAME> GL_ARCHIVE_FLAG=<Y/N>
```

The parameters CONC_PROGRAM and APP_SHORT_NAME are required. All other parameters are optional and are used to override the default behavior.

Table A-20. ksc_gl_migrate parameters

Parameter	Default Token	Description
CONC _PROGRAM	None. This is a mandatory parameter.	The concurrent program name. This has been pre-configured and will not need to be modified.
GL_ARCHIVE _FLAG	[WFS. OM_ARCHIVE_FLAG]	Specify whether the migration will store to the archive rather than using what has been specified for the current workflow step.

Table A-20. ksc_gl_migrate parameters

Parameter	Default Token	Description
APP_SHORT _NAME	None. This is a required parameter.	This value is normally “CLGM” but can be modified if the GL*Migrator has been installed into a custom account.
SOURCE _ENV	[SOURCE_ENV]	The environment to migrate from rather than the one defined on the workflow step.
DEST _ENV	[DEST_ENV]	The environment to migrate to rather than the one defined on the workflow step.

Example using ksc_gl_migrate

```
#
# Launch a Budget Organization migration
#
ksc_gl_migrate CONC_PROGRAM="CLGMRBO1" APP_SHORT_NAME="CLGM"
```

ksc_parse_jcl

This command is only used by the ‘OS/390 JCL Migration’ object type to parse a JCL script using the Mainframe parameters for the specified environment.

Table A-21. ksc_parse_jcl parameters

Parameter	Default Token	Description
FILENAME	[P.P_FILENAME]	Name of the JCL source file to be edited.
OUTFILE	[OUTFILE]	Name of the output JCL file after applying the substitution expressions.
ENV_ID	[DEST_ENV. ENVIRONMENT_ID]	The ID of the environment containing the mainframe substitution expressions.

ksc_submit_job

This command is only used by the ‘OS/390 JCL Migration’ object type to submit JCL to the Mainframe JES.

Table A-22. ksc_submit_job parameters

Parameter	Default Token	Description
PATH	[AS.PKG_TRANSFER_PATH]	Path to the JCL file.
FILENAME	[P.P_FILENAME]	Name of the JCL source file to be edited.

ksc_set_exit_value

Use this command to set the exit value of the command execution to any value. When not used, the command execution engine returns standard execution results, such as FAILURE, SUCCESS, and ERROR (if an internal error occurred) that the workflow engine can transition on. Using ‘ksc_set_exit_value’ allows for the flexibility to set any exit value and enables custom workflow transitions.

The following formats are supported:

```
# Sets the hidden and visible value to <value>.
ksc_set_exit_value "<value>"

# Sets both the hidden and visible values independently.
ksc_set_exit_value "<hidden_value>", "<visible_value">
```

The workflow engine will key off of the hidden value to determine if a transition should be made. The visible value is for display purposes.

‘ksc_set_exit_value’ is ideal for situations where there could be a number of different execution results, not just Success or Failure. Using ‘ksc_set_exit_value’ allows the workflow engine to transition on any number of execution outcomes.

ksc_clear_exit_value

Use this command to clear the exit value set by ‘ksc_set_exit_value’. When cleared, the execution engine will return its standard results, SUCCESS, FAILURE, or ERROR.

ksc_run_sql

This command runs a SQL query against the chosen environment. The result of the last row queried is returned in the [SQL_OUTPUT] token. The result of the entire query is placed in the [AS.PKG_TRANSFER_PATH][PKGL.SEQ].txt file in <*ITG_Home*>.

To run this special command, any execution steps in the Change Management workflow must have their source environments defined in the Workflow Step window.

Table A-23 lists this special command’s parameters.

Table A-23. ksc_run_sql parameters

Parameter	Default Token	Description
QUERY_STRING	[QUERY_STRING]	A SQL select statement.
ENV_NAME	[ENV_NAME]	The name of the environment where data will be queried. The JDBC connection should be checked in the environment checker.
EXCEPTION_OPTION	[EXCEPTION_OPTION]	If no data is returned, determines if an exception should be thrown. The only available option is '-no_data_exception.'

Example using ksc_run_sql

```
ksc_run_sql QUERY_STRING="select sysdate from sys.dual" ENV_
NAME="[SOURCE_ENV.ENVIRONMENT_NAME]"
```

Note

The ‘ksc_run_sql’ special command can be used to populate a validation. This is appropriate when the validation is validated by a Command with Delimited Output. In this case, the Data Delimiter should be set to “#@#”.

The following code is an example of ksc_run_sql in a validation.

```
ksc_run_sql QUERY_STRING=”select id, name from some_table”
ENV_NAME=”[SOURCE_ENV.ENVIRONMENT_NAME]”
ksc_capture_output cat [AS.PKG_TRANSFER_PATH][PKGL.SEQ].txt
```

Summary of All Special Command Parameters

Table A-24 provides the parameters for all predefined special commands.

Table A-24. Special command parameters

Special Command	Parameters	Defaults
ksc_begin_script		
ksc_comment		
ksc_concsub		
ksc_connect_dest_client	USERNAME	[DEST_ENV.CLIENT_USERNAME]
	PASSWORD	[DEST_ENV.CLIENT_PASSWORD]
	NT_DOMAIN	[DEST_ENV.CLIENT_NT_DOMAIN]
	DEST_BASE_PATH	[DEST_ENV.CLIENT_BASE_PATH]
	CONNECTION_ PROTOCOL	[DEST_ENV.CLIENT_CON_PROTOCOL_ MEANING]
	DEST_ENV	[DEST_ENV]

Table A-24. Special command parameters [continued]

Special Command	Parameters	Defaults
ksc_connect_dest_server	USERNAME	[DEST_ENV.SERVER_USERNAME]
	PASSWORD	[DEST_ENV.SERVER_PASSWORD]
	NT_DOMAIN	[DEST_ENV.SERVER_NT_DOMAIN]
	DEST_BASE_PATH	[DEST_ENV.SERVER_BASE_PATH]
	CONNECTION_ PROTOCOL	[DEST_ENV.SERVER_CON_ PROTOCOL_MEANING]
	DEST_ENV	[DEST_ENV]
ksc_connect_source_client	USERNAME	[SOURCE_ENV.CLIENT_USERNAME]
	PASSWORD	[SOURCE_ENV.CLIENT_PASSWORD]
	NT_DOMAIN	[SOURCE_ENV.CLIENT_NT_DOMAIN]
	SOURCE_BASE_ PATH	[SOURCE_ENV.CLIENT_BASE_PATH]
	CONNECTION_ PROTOCOL	[SOURCE_ENV.CLIENT_CON_ PROTOCOL_MEANING]
	SOURCE_ENV	[SOURCE_ENV]
ksc_connect_source_server	USERNAME	[SOURCE_ENV.SERVER_USERNAME]
	PASSWORD	[SOURCE_ENV.SERVER_PASSWORD]
	NT_DOMAIN	[SOURCE_ENV.SERVER_NT_DOMAIN]
	SOURCE_BASE_ PATH	[SOURCE_ENV.SERVER_BASE_PATH]
	CONNECTION_ PROTOCOL	[SOURCE_ENV.SERVER_CON_ PROTOCOL_MEANING]
	SOURCE_ENV	[SOURCE_ENV]
ksc_copy_client_client	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_ PATH	[SOURCE_ENV.CLIENT_BASE_PATH]
	DEST_BASE_PATH	[DEST_ENV.CLIENT_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]

Table A-24. Special command parameters [continued]

Special Command	Parameters	Defaults
ksc_copy_client_server	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_ PATH	[SOURCE_ENV.CLIENT_BASE_PATH]
	DEST_BASE_PATH	[DEST_ENV.SERVER_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
ksc_copy_server_client	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_PATH	[SOURCE_ENV.SERVER_BASE_PATH]
	DEST_BASE_PATH	[DEST_ENV.CLIENT_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
ksc_copy_server_server	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_ PATH	[SOURCE_ENV.SERVER_BASE_PATH]
	DEST_BASE_PATH	[DEST_ENV.SERVER_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
ksc_copy_client_tmp	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_ PATH	[SOURCE_ENV.CLIENT_BASE_PATH]
	DEST_BASE_PATH	[DEST_ENV.CLIENT_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]

Table A-24. Special command parameters [continued]

Special Command	Parameters	Defaults
ksc_copy_server_tmp	SUB_PATH	[P.P_SUB_PATH]
	SOURCE_BASE_ PATH	[SOURCE_ENV.SERVER_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	SOURCE_ENV	[SOURCE_ENV]
ksc_copy_tmp_client	SUB_PATH	[P.P_SUB_PATH]
	DEST_BASE_ PATH	[DEST_ENV.CLIENT_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	DEST_ENV	[DEST_ENV]
ksc_copy_tmp_server	SUB_PATH	[P.P_SUB_PATH]
	DEST_BASE_ PATH	[DEST_ENV.SERVER_BASE_PATH]
	FILENAME	[P.P_FILENAME]
	FILE_TYPE	[P.P_FILE_TYPE]
	DEST_ENV	[DEST_ENV]
ksc_copy_script_dest_client	SCRIPT_FILENAME	[P.P_SCRIPT_FILENAME]
	DEST_BASE_PATH	[DEST_ENV.CLIENT.BASE_PATH]
	DEST_ENV	[DEST_ENV]
ksc_copy_script_dest_server	SCRIPT_FILENAME	[P.P_SCRIPT_FILENAME]
	DEST_BASE_PATH	[DEST_ENV.SERVER_BASE_PATH]
	DEST_ENV	[DEST_ENV]
ksc_copy_script_source_client	SCRIPT_FILENAME	[P.P_SCRIPT_FILENAME]
	SOURCE_BASE_ PATH	[SOURCE_ENV.CLIENT_BASE_PATH]
	SOURCE_ENV	[SOURCE_ENV]

Table A-24. Special command parameters [continued]

Special Command	Parameters	Defaults
ksc_copy_script_source_ server	SCRIPT_FILENAME	[P.P_SCRIPT_FILENAME]
	SOURCE_BASE_ PATH	[SOURCE_ENV.SERVER_BASE_PATH]
	SOURCE_ENV	[SOURCE_ENV]
ksc_clear_exit_value		
ksc_end_script		
ksc_exit		
ksc_gl_migrate	CONC_PROGRAM	
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
	GL_ARCHIVE_FLAG	[WFS.GL_ARCHIVE_FLAG]
	APP_SHORT_NAME	
ksc_local_exec		
ksc_om_migrate	CONC_PROGRAM	
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
	OM_ARCHIVE_FLAG	[WFS.OM_ARCHIVE_FLAG]
	APP_SHORT_NAME	
ksc_parse_jcl	FILENAME	[P.P_FILENAME]
	OUTFILE	
	ENV_ID	[DEST_ENV.ENVIRONMENT_ID]
ksc_replace	FILENAME	[P.P_FILENAME]
	OUTFILE	
	SUBST	
ksc_respond		
ksc_run_sql		
ksc_set	Custom Token	

Table A-24. Special command parameters [continued]

Special Command	Parameters	Defaults
ksc_set_env	DEST_ENV_ID	[DEST_ENV.ENVIRONMENT_ID]
	SOURCE_ENV_ID	[SOURCE_ENV.ENVIRONMENT_ID]
	SOURCE_ENV	[SOURCE_ENV]
	DEST_ENV	[DEST_ENV]
ksc_set_exit_value		
ksc_simple_respond		
ksc_store	Custom Token	
ksc_submit_job	PATH	[AS.PKG_TRANSFER_PATH]
	FILENAME	[P.P_FILENAME]

Appendix B Tokens

In This Appendix:

- *Overview of Tokens*
- *System Tokens*
- *Field Group Tokens*

Overview of Tokens

This appendix provides a list of all entity tokens. Use *Table B-1* as a quick reference guide to jump to the desired location.

Table B-1. Token tables

Table	Page
Table B-2, App server properties	177
Table B-3, Budget	178
Table B-4, Contacts	178
Table B-5, Distribution	179
Table B-6, Document Management	180
Table B-7, Environments	180
Table B-8, Environment applications	183
Table B-9, Command execution	185
Table B-10, Notifications	185
Table B-11, Organization unit	186
Table B-12, Packages	187
Table B-13, Package lines	189
Table B-14, Package pending	190
Table B-15, Program	191
Table B-16, Project plans	191
Table B-17, Project plan details	194
Table B-18, Releases	194
Table B-19, Requests	195
Table B-20, Request details	198
Table B-21, Request pending	198
Table B-22, Report submissions	199
Table B-23, Resource pools	200
Table B-24, Security groups	201
Table B-25, Skill	201

Table B-1. Token tables [continued]

Table	Page
Table B-26, Staffing profile	202
Table B-27, System	202
Table B-28, Tasks	203
Table B-29, Tasks pending	205
Table B-30, Users	206
Table B-31, Validations	208
Table B-32, Validation values	208
Table B-33, Workflows	209
Table B-34, Workflow steps	210
Table B-35, Workflow step transaction	212

For a list of the tokens that are associated with field groups, see *Field Group Tokens* on page 213.

System Tokens

Table B-2. App server properties

Prefix	Token	Description
AS	PKG_TRANSFER_PATH	Temporary directory used for files during command executions.

Note

Other app server properties tokens are generated from the parameters in the server.conf file. For a description of each server parameter, see the *System Administration Guide and Reference.*

Table B-3. Budget

Prefix	Token	Description
BGT	ACTIVE_FLAG	The active flag of the budget.
BGT	BUDGET_ID	The ID of the budget (defined in the table KCST_ BUDGETS).
BGT	BUDGET_IS_FOR_ENTITY_NAME	The entity name (project plan, program, or org unit) to which the budget is linked.
BGT	BUDGET_IS_FOR_ID	The ID of the project plan/program/org unit to which the budget is linked.
BGT	BUDGET_IS_FOR_NAME	The name of the project plan/program/org unit to which the budget is linked.
BGT	BUDGET_NAME	The name of the budget.
BGT	BUDGET_ROLLS_UP_TO_ID	The ID of the budget to which this budget rolls up to.
BGT	BUDGET_ROLLS_UP_TO_NAME	The name of the budget to which this budget rolls up to.
BGT	BUDGET_URL	The URL to view this budget.
BGT	CREATED_BY	The username of the user who created the budget.
BGT	CREATION_DATE	The date when the budget was created.
BGT	DESCRIPTION	The description of the budget.
BGT	END_PERIOD	The end period of the budget.
BGT	INITIATION_REQ	The initiation request ID of the budget.
BGT	PERIOD_SIZE	The period size of the budget.
BGT	START_PERIOD	The start period of the budget.
BGT	STATUS_CODE	The status code of the budget.
BGT	STATUS_NAME	The status name of the budget.

Table B-4. Contacts

Prefix	Token	Description
CON	COMPANY	The company the Contact works for.
CON	COMPANY_NAME	The name of the company the Contact works for.
CON	CONTACT_ID	The ID of the Contact (defined in the table KCRT_CONTACTS).
CON	CREATED_BY	The ID of the user that created the Contact.

Table B-4. Contacts [continued]

Prefix	Token	Description
CON	CREATION_DATE	The date the Contact was created.
CON	EMAIL_ADDRESS	The email address of the Contact.
CON	FIRST_NAME	The first name of the Contact.
CON	FULL_NAME	The full name of the Contact.
CON	LAST_NAME	The last name of the Contact.
CON	LAST_UPDATED_BY	The ID of the user that last updated the Contact.
CON	LAST_UPDATE_DATE	The date the Contact was last updated.
CON	PHONE_NUMBER	The phone number of the Contact.
CON	USER_ID	The userID of the Contact, if the Contact is a Mercury IT Governance Center user.
CON	USERNAME	The username of the Contact (if applicable). This may be a username for an external system, not necessarily Mercury IT Governance Center.

Table B-5. Distribution

Prefix	Tokens	Description
DIST	CREATED_BY	The ID of the user that created the distribution.
DIST	CREATED_BY_USERNAME	The Mercury IT Governance Center username of the user that created the distribution.
DIST	DESCRIPTION	The description of the release.
DIST	DISTRIBUTION_ID	The ID of the distribution (defined in table KREL_ DIESTRIBUTION).
DIST	DISTRIBUTION_NAME	The name of the distribution.
DIST	DISTRIBUTION_STATUS	The workflow status of the distribution workflow.
DIST	FEEDBACK_FLAG	Whether the distribution has fed back a specified value to the package lines being distributed.
DIST	FEEDBACK_VALUE	The value to be returned to the original package lines.
DIST	LAST_UPDATED_BY	The ID of the user that last updated the distribution.
DIST	LAST_UPDATED_BY_ USERNAME	The Mercury IT Governance Center username of the user that last updated the distribution.
DIST	LAST_UPDATE_DATE	The date the distribution was last updated.

Table B-5. Distribution [continued]

Prefix	Tokens	Description
DIST	RELEASE_ID	The ID of the release that created this distribution.
DIST	RELEASE_NAME	The name of the release that created this distribution.
DIST	WORKFLOW	The workflow used to process the distribution.

Table B-6. Document Management

Prefix	Token	Description
DMS	DOC_LINK	Resolves to a URL which, when clicked, opens the latest version of the document. Forces user authentication prior to delivering the document.
DMS	DOC_HISTORY	Resolves to a URL which, when clicked, displays a view of the document’s version history Forces user authentication prior to delivering the information.
DMS	AUTHOR	Resolves to the author descriptive field stored with the document.
DMS	DESCRIPTION	Resolves to the descriptive field stored with the document.
DMS	LAST_CHECK_IN_DATE	Resolves to the timestamp of the last check-in.
DMS	LAST_CHECKED_IN_BY_NAME	Resolves to the full name of the Mercury IT Governance Center user who added or last checked in the document.
DMS	LAST_CHECKED_IN_BY	Resolves tothe ID of the Mercury IT Governance Center user who added or last checked in the document.

Table B-7. Environments

Prefix	Token	Description
ENV	CLIENT_BASE_PATH	The base (root) path of the client.
ENV	CLIENT_CON_PROTOCOL	The protocol used to connect to this client.
ENV	CLIENT_CON_PROTOCOL_MEANING	The visible value of the client connect protocol.
ENV	CLIENT_NAME	The DNS name or IP address of the client computer.

Table B-7. Environments [continued]

Prefix	Token	Description
ENV	CLIENT_NT_DOMAIN	The domain name for the client, if the client machine type is Windows.
ENV	CLIENT_ENABLED_FLAG	The flag indicating whether the client portion of the environment is enabled.
ENV	CLIENT_PASSWORD	The password Mercury IT Governance Center uses to log onto or access the client. This value is encrypted.
ENV	CLIENT_TYPE_CODE	The validation value code of the client machine type.
ENV	CLIENT_USERNAME	The username Mercury IT Governance Center uses to log onto or access the client.
ENV	CLIENT_TRANSFER_PROTOCOL	The protocol used to transfer files to or from this client.
ENV	CLIENT_TRANSFER_PROTOCOL_ MEANING	The visible value of the client transfer protocol.
ENV	CREATED_BY	The ID of the user that created the environment.
ENV	CREATION_DATE	The date the environment was created.
ENV	DATABASE_ENABLED_FLAG	The flag indicating whether the database portion of the environment is enabled.
ENV	DATABASE_TYPE	The validation value code of the database type.
ENV	DB_CONNECT_STRING	For Oracle database type, the connect string used to access the database from the command line.
ENV	DB_LINK	For Oracle database type, the database link from the Mercury IT Governance Center schema to the environment’s database schema.
ENV	DB_NAME	The DNS name or IP address of the database server.
ENV	DB_ORACLE_SID	For Oracle database type, the SID of the database (often the same as the DB_CONNECT_STRING).
ENV	DB_PASSWORD	The password Mercury IT Governance Center uses to log onto or access the database. This value is encrypted.
ENV	DB_PORT_NUMBER	For Oracle database type, the port number on which SQL*Net is listening for remote SQL connections on the database server.

Table B-7. Environments [continued]

Prefix	Token	Description
ENV	DB_USERNAME	The username or schema name Mercury IT Governance Center uses to log onto or access the database.
ENV	DB_VERSION	The version of the database (such as 8.1.7).
ENV	DESCRIPTION	The description of the environment.
ENV	ENABLED_FLAG	The flag indicating whether the environment is enabled and available for use in workflows.
ENV	ENVIRONMENT_ID	The ID of the environment in the table KENV_ ENVIRONMENTS.
ENV	ENVIRONMENT_NAME	The name of the environment.
ENV	LAST_UPDATED_BY	The ID of the user that last updated the environment.
ENV	LAST_UPDATE_DATE	The date the environment was last updated.
ENV	LOCATION	The location of the environment.
ENV	MSSQL_DB_NAME	For MS SQL Server database type, the database name used to access the database from the command line.
ENV	SERVER_BASE_PATH	The base (root) path of the server.
ENV	SERVER_CON_PROTOCOL	The protocol used to connect to this server.
ENV	SERVER_CON_PROTOCOL_ MEANING	The visible value of the server connection protocol.
ENV	SERVER_TRANSFER_PROTOCOL	The protocol used to transfer files to or from this server.
ENV	SERVER_TRANSFER_PROTOCOL_ MEANING	The visible value of the server transfer protocol.
ENV	SERVER_ENABLED_FLAG	The flag indicating whether the server portion of the environment is enabled.
ENV	SERVER_NAME	The DNS name or IP address of the server computer.
ENV	SERVER_NT_DOMAIN	The domain name for the server, if the server machine type is Windows.
ENV	SERVER_PASSWORD	The password Mercury IT Governance Center uses to log onto or access the server. This value is encrypted.

Table B-7. Environments [continued]

Prefix	Token	Description
ENV	SERVER_TYPE_CODE	The validation value code of the server machine type.
ENV	SERVER_USERNAME	The username Mercury IT Governance Center uses to log onto or access the server.

Note

If any Mercury IT Governance Center Extensions have been installed, there will be more environment tokens with the prefix ‘AC.’ For more detailed information on these tokens, see the Mercury IT Governance Center Extensions documentation.

Table B-8. Environment applications

Prefix	Token	Description
ENV.APP	APP_CODE	The short name (code) for the application.
ENV.APP	APP_NAME	The descriptive name for the application.
ENV.APP	CLIENT_BASE_PATH	The application-specific base (root) path of the client.
ENV.APP	CLIENT_PASSWORD	The application-specific password Mercury IT Governance Center uses to log onto or access the client. This value is encrypted.
ENV.APP	CLIENT_USERNAME	The application-specific username Mercury IT Governance Center uses to log onto or access the client.
ENV.APP	CLIENT_CON_PROTOCOL	The application-specific protocol used to connect to this client.
ENV.APP	CLIENT_CON_PROTOCOL_ MEANING	The visible value of the client connection protocol.
ENV.APP	CLIENT_TRANSFER_ PROTOCOL	The application-specific protocol used to transfer files to and from this client.
ENV.APP	CLIENT_TRANSFER_ PROTOCOL_MEANING	The visible value of the client transfer protocol.
ENV.APP	CREATED_BY	The ID of the user that created the application.
ENV.APP	CREATION_DATE	The date the application was created.

Table B-8. Environment applications [continued]

Prefix	Token	Description
ENV.APP	DB_LINK	For Oracle database type, the application-specific database link from the Mercury IT Governance Center schema to the environment's database schema.
ENV.APP	DB_NAME	For MS SQL Server database type, the application-specific database name used to access the database from the command line.
ENV.APP	DB_PASSWORD	The application-specific password Mercury IT Governance Center uses to log onto or access the database. This value is encrypted.
ENV.APP	DB_USERNAME	The application-specific username or schema name Mercury IT Governance Center uses to log onto or access the database.
ENV.APP	DESCRIPTION	The description of the application.
ENV.APP	ENABLED_FLAG	The flag indicating whether the application is enabled and available for selection in package lines.
ENV.APP	ENVIRONMENT_APP_ID	The ID of the application in the table KENV_ ENVIRONMENT_APPS.
ENV.APP	ENVIRONMENT_ID	The ID of the environment the application is associated with.
ENV.APP	ENVIRONMENT_NAME	The name of the environment the application is associated with.
ENV.APP	LAST_UPDATED_BY	The ID of the user that last updated the application.
ENV.APP	LAST_UPDATE_DATE	The date the application was last updated.
ENV.APP	SERVER_CON_PROTOCOL	The application-specific protocol used to connect to this server.
ENV.APP	SERVER_CON_PROTOCOL_ MEANING	The visible value of the server connection protocol.
ENV.APP	SERVER_TRANSFER_ PROTOCOL	The application-specific protocol used to transfer files to and from this server.
ENV.APP	SERVER_TRANSFER_ PROTOCOL_MEANING	The visible value of the server transfer protocol.
ENV.APP	SERVER_BASE_PATH	The application-specific base (root) path of the server

Table B-8. Environment applications [continued]

Prefix	Token	Description
ENV.APP	SERVER_PASSWORD	The application-specific password Mercury IT Governance Center uses to log onto or access the server. This value is encrypted.
ENV.APP	SERVER_USERNAME	The application-specific username Mercury IT Governance Center uses to log onto or access the server.
ENV.APP	WORKBENCH_ENVIRONMENT_URL	The URL of the environment window in the Workbench.

Table B-9. Command execution

Prefix	Token	Description
EXEC	EXIT_CODE	The exit code of a command execution.
EXEC	OUTPUT	The last line of output from a command execution.

Note

The command execution tokens, [EXEC.OUTPUT] and [EXEC.EXIT_CODE], can be used in the following contexts:

- Inside command step segments that use the ksc_connect and ksc_exit special commands.
- Immediately after command step segments that use the ksc_local_exec special command.

For example, the following code segment demonstrates how to use both command execution tokens to retrieve the output and exit code immediately upon execution. The tokens are used immediately after the ksc_local_exec special command.

```
ksc_local_exec pwd
ksc_set MY_PATH="[EXEC.OUTPUT]"
ksc_set MY_EXIT_CODE="[EXEC.EXIT_CODE]"
ksc_local_exec echo '[MY_PATH]/bin'
ksc_local_exec echo '[MY_EXIT_CODE]'
```

Table B-10. Notifications

Prefix	Tokens	Description
NOTIF	CC_USERS	The list of users on the Cc: header of the notification.
NOTIF	CHANGED_FIELD	The field that changed to trigger a notification.

Table B-10. Notifications [continued]

Prefix	Tokens	Description
NOTIF	EXCEPTION_RULE	The exception rule that was met by the task exception that caused the notification to be sent.
NOTIF	EXCEPTION_RULE_NAME	The name of the task exception that caused the notification to be sent.
NOTIF	EXCEPTION_VIOLATION	The specific violation of the exception that caused the notification to be sent.
NOTIF	NEW_VALUE	The new value of the changed field.
NOTIF	NOTIFICATION_DETAILS	Notification details for linked tokens.
NOTIF	OLD_VALUE	The previous value of the changed field.
NOTIF	TO_USERS	The list of users on the To: header of the notification.

Table B-11. Organization unit

Prefix	Tokens	Description
ORG	BUDGET_ID	The ID of the budget linked to this org unit.
ORG	BUDGET_NAME	The name of the budget linked to this org unit.
ORG	CATEGORY_CODE	The lookup code of the org unit category (lookup type = RSC - org unit Category)
ORG	CATEGORY_NAME	The category name of the org unit.
ORG	CREATED_BY	The ID of the user that created the org unit.
ORG	CREATED_BY_USERNAME	The name of the user that created the org unit.
ORG	CREATION_DATE	The date that the org unit was created.
ORG	DEPARTMENT_CODE	The lookup code of the org unit department (lookup type = DEPT)
ORG	DEPARTMENT_NAME	The department name of the org unit.
ORG	LOCATION_CODE	The lookup code of the org unit location (lookup type = RSC - Location)
ORG	LOCATION_NAME	The location name of the org unit.
ORG	MANAGER_ID	The ID of the manager of the org unit.
ORG	MANAGER_USERNAME	The name of the manager of the org unit.
ORG	ORG_UNIT_ID	The ID of the org unit (defined in table KRSC_ORG_UNITS).

Table B-11. Organization unit [continued]

Prefix	Tokens	Description
ORG	ORG_UNIT_NAME	The name of the org unit.
ORG	PARENT_ORG_UNIT_ID	The ID of the parent org unit.
ORG	PARENT_ORG_UNIT_NAME	The name of the parent org unit.
ORG	REGIONAL_CALENDAR	The name of the regional calendar for the org unit.

Table B-12. Packages

Prefix	Token	Description
PKG	ASSIGNED_TO_EMAIL	The email address of the user that the package is assigned to.
PKG	ASSIGNED_TO_GROUP_ID	The ID of the security group that the package has been assigned to.
PKG	ASSIGNED_TO_GROUP_NAME	The security group that the package has been assigned to.
PKG	ASSIGNED_TO_USERNAME	The name of the user that the package has been assigned to.
PKG	ASSIGNED_TO_USER_ID	The ID of the user that the package has been assigned to.
PKG	CREATED_BY	The ID of the user that created the package.
PKG	CREATED_BY_EMAIL	The email address of the user that created the package.
PKG	CREATED_BY_USERNAME	The Mercury IT Governance Center username of the user that created the package.
PKG	CREATION_DATE	The date the package was created.
PKG	DESCRIPTION	The description of the package.
PKG	ID	The ID of the package in the table KDLV_PACKAGES.
PKG	LAST_UPDATED_BY	The ID of the user that last updated the package.
PKG	LAST_UPDATED_BY_EMAIL	The email address of the user that last updated the package.
PKG	LAST_UPDATED_BY_ USERNAME	The Mercury IT Governance Center username of the user that last updated the package.
PKG	LAST_UPDATE_DATE	The date the package was last updated.
PKG	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.

Table B-12. Packages [continued]

Prefix	Token	Description
PKG	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.
PKG	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
PKG	MOST_RECENT_NOTE_TEXT	Text of the most recent (chronological) note.
PKG	NOTES	All notes for the package.
PKG	NUMBER	The name/number of the package.
PKG	PACKAGE_GROUP_CODE	The package group code.
PKG	PACKAGE_GROUP_NAME	The name of the package group.
PKG	PARENT_REQUEST_ID	The ID of the request that created this package (if applicable).
PKG	PRIORITY	The priority of the package.
PKG	PRIORITY_CODE	The validation value code of the package priority.
PKG	PRIORITY_NAME	The validation value meaning of the package priority.
PKG	PRIORITY_SEQ	The priority sequence of the package.
PKG	PROJECT_CODE	The validation value code of the project plan the package belongs to.
PKG	PROJECT_NAME	The validation value meaning of the project plan the package belongs to.
PKG	SUBMIT_DATE	The date that the package was submitted.
PKG	REQUESTED_BY_EMAIL	The email address of the user who requested the package.
PKG	REQUESTED_BY_USERNAME	The Mercury IT Governance Center username of the user who requested the package.
PKG	REQUESTED_BY_USER_ID	The ID of the user that requested the package.
PKG	PACKAGE_ID	The ID of the package in the table KDLV_PACKAGES.
PKG	PACKAGE_NO_LINK	Shows up as a standard hyperlink to the package in HTML-format notifications.
PKG	PACKAGE_TYPE	The validation value meaning of the package type.
PKG	PACKAGE_TYPE_CODE	The validation value code of the package type.
PKG	PACKAGE_URL	The URL of the package in the standard interface.
PKG	PERCENT_COMPLETE	Percent complete of the package.

Table B-12. Packages [continued]

Prefix	Token	Description
PKG	RUN_GROUP	The run group of the package.
PKG	STATUS	The validation value meaning for the status of the package.
PKG	STATUS_CODE	The validation value code for the status of the package.
PKG	WORKBENCH_PACKAGE_NO_ LINK	The URL of the package in the Workbench.
PKG	WORKBENCH_PACKAGE_URL	The URL of the package screen in the Workbench.
PKG	WORKFLOW_ID	The ID of the workflow used by the package.
PKG	WORKFLOW_NAME	The name of the workflow used by the package.

Table B-13. Package lines

Prefix	Token	Description
PKGL	APP_CODE	The app code for the package line.
PKGL	APP_NAME	The name of the application for the package line.
PKGL	ID	The ID of the package line in the table KDLV_PACKAGE_ LINES.
PKGL	OBJECT_CATEGORY_CODE	The validation value code of the object type category of the line.
PKGL	OBJECT_CATEGORY_NAME	The validation value meaning of the object type category of the line.
PKGL	OBJECT_NAME	The object name of the package line.
PKG	OBJECT_REVISION	The value of the object revision column (if any) as specified by the object type of the package line.
PKGL	OBJECT_TYPE	The object type of the package line.
PKGL	OBJECT_TYPE_ID	The ID of the object type of the package line.
PKGL	PACKAGE_LINE_ID	The ID of the package line.
PKGL	SEQ	The sequence of the package line (relative to other lines in the same package).
PKGL	WORKBENCH_OBJECT_ TYPE_URL	URL to access the object type window for this object type in the Workbench.

Table B-14. Package pending

Prefix	Tokens	Description
PKG.PEND	ID	The ID of the entity that is being blocked by the package.
PKG.PEND	NAME	The name of the entity that is being blocked by the package.
PKG.PEND	DETAIL	Detail information for the entity that is being blocked by the package.
PKG.PEND	DESCRIPTION	The description of the entity that is being blocked by the package.
PKG.PEND	STATUS_ID	The ID of the state or code of the status of the entity that is being blocked by the package.
PKG.PEND	STATUS_NAME	The name of the status (or state) of the entity that is being blocked by the package.
PKG.PEND	STATE	The name of the state of the entity of the request that is being blocked by the package.
PKG.PEND	ASSIGNED_TO_USERNAME	The name of the assigned user (or resource) of the entity that is being blocked by the package.
PKG.PEND	ASSIGNED_TO_USER_ID	The username of the assigned user (or resource) of the entity that is being blocked by the package.
PKG.PEND	ASSIGNED_TO_GROUP_NAME	The name of the assigned group (or resource group) of the entity that is being blocked by the package.
PKG.PEND	ASSIGNED_TO_GROUP_ID	The ID of the assigned group (or resource group) of the entity that is being blocked by the package.
PKG.PEND	RESOURCE_USERNAME	The name of the resource associated with the entity that is being blocked by the package.
PKG.PEND	RESOURCE_ID	The username of the assigned user (or resource) associated with the entity that is being blocked by the package.
PKG.PEND	RESOURCE_GROUP_NAME	The name of the assigned group (or resource group) associated with the entity that is being blocked by the package.
PKG.PEND	RESOURCE_GROUP_ID	The ID of the assigned group (or resource group) associated with the entity that is being blocked by the package.
PKG.PEND	PERCENT_COMPLETE	The current percent complete value associated with the entity that is being blocked by the package.

Table B-14. Package pending [continued]

Prefix	Tokens	Description
PKG.PEND	ENTITY_TYPE_ID	The ID of the type of entity that is being blocked by the package.
PKG.PEND	ENTITY_TYPE_NAME	The name of the type of entity that is being blocked by the package.

Table B-15. Program

Prefix	Token	Description
PRG	CREATED_BY	The ID of the user that created the program.
PRG	CREATED_BY_USERNAME	The name of the user that created the program.
PRG	LAST_UPDATED_BY	The ID of the user that last updated the program.
PRG	LAST_UPDATED_BY_ USERNAME	The name of the user that last updated the program.
PRG	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.
PRG	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.
PRG	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
PRG	MOST_RECENT_NOTE_TEXT	Text of the most recent (chronological) note.
PRG	PROGRAM_MANAGER	The ID(s) of the user(s) assigned to manage this program.

Table B-16. Project plans

Prefix	Tokens	Description
PRJ	ACTUAL_DURATION	The actual duration of the project plan.
PRJ	ACTUAL_EFFORT	The actual effort associated with the project plan.
PRJ	ACTUAL_FINISH_DATE	The actual finish date of the project plan.
PRJ	ACTUAL_START_DATE	The actual start date of the project plan.
PRJ	BUDGET_ID	The ID of the budget linked to the project plan.
PRJ	BUDGET_NAME	The name of the budget linked to the project plan.
PRJ	CONFIDENCE_CODE	The code of the confidence value entered by the user.
PRJ	CONFIDENCE_NAME	The name of the confidence value entered by the user.

Table B-16. Project plans [continued]

Prefix	Tokens	Description
PRJ	CREATED_BY	The user who created the project plan.
PRJ	CREATED_BY_EMAIL	The email address of the user who created the project plan.
PRJ	CREATED_BY_USERNAME	The username of the person who created the project plan.
PRJ	CREATION_DATE	The creation date of the project plan.
PRJ	DEPARTMENT_CODE	The code of the department value entered by the user.
PRJ	DEPARTMENT_NAME	The name of the department value entered by the user.
PRJ	DESCRIPTION	The description of the project plan.
PRJ	ESTIMATED_REMAINING_ DURATION	The estimated remaining duration of the project plan.
PRJ	ESTIMATED_REMAINING_ EFFORT	The estimated remaining effort involved in the project plan.
PRJ	ESTIMATED_FINISH_DATE	The estimated finish date of the project plan.
PRJ	LAST_UPDATE_DATE	The date the project plan was last updated.
PRJ	LAST_UPDATED_BY	The last person to update the project plan.
PRJ	LAST_UPDATED_BY_EMAIL	The email address of the last person to update the project plan.
PRJ	LAST_UPDATED_BY_ USERNAME	The username of the last person to update the project plan.
PRJ	MASTER_PROJECT_ID	The ID of the master project.
PRJ	MASTER_PROJECT_NAME	The name of the master project.
PRJ	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.
PRJ	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.
PRJ	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
PRJ	MOST_RECENT_NOTE_TEXT	Text of the most recent (chronological) note.
PRJ	MOST_RECENT_NOTE_TYPE	Type of the most recent (chronological) note (USER or FIELD CHANGE).
PRJ	MOST_RECENT_USER_ NOTE_AUTHOR_FULL_NAME	First and last name of the author of the most recent user note.

Table B-16. Project plans [continued]

Prefix	Tokens	Description
PRJ	MOST_RECENT_USER_ NOTE_AUTHOR_USERNAME	Username of the author of the most recent user note.
PRJ	MOST_RECENT_USER_ NOTE_AUTHORED_DATE	Date of the most recent user note.
PRJ	MOST_RECENT_USER_ NOTE_TEXT	Text of the most recent user note.
PRJ	MOST_RECENT_USER_ NOTE_TYPE	Type of the most recent user note (USER or FIELD CHANGE).
PRJ	PARENT_PROJECT_ID	The ID of the parent project plan.
PRJ	PARENT_PROJECT_NAME	The name of the parent project plan.
PRJ	PERCENT_COMPLETE	The project plan's completed percentage.
PRJ	PRIORITY	The priority of the project plan.
PRJ	PROJECT_ID	The number that uniquely identifies the project plan (same as PROJECT_NUMBER) in the table KDRV_PROJECTS.
PRJ	PROJECT_MANAGER	The manager of the project plan.
PRJ	PROJECT_MANAGER_EMAIL	The email address of the project manager.
PRJ	PROJECT_MANAGER_ USERNAME	The username of the project manager.
PRJ	PROJECT_NAME	The name of the project plan.
PRJ	PROJECT_NAME_LINK	Shows up as a standard hyperlink to the project plan in HTML-format notifications.
PRJ	PROJECT_NUMBER	The number that uniquely identifies the project plan (same as PROJECT_ID).
PRJ	PROJECT_PATH	The project plan path. This is a hierarchy of parent project plans that contain this project plan.
PRJ	PROJECT_STATE	The project plan state.
PRJ	PROJECT_TEMPLATE	The name of the project template used to create the project plan.
PRJ	PROJECT_TYPE_CODE	Returns TASK for tasks and PROJECT for project plans.
PRJ	PROJECT_URL	The URL for the project plan's Project Overview page.
PRJ	REGIONAL_CALENDAR	The name of the regional calendar for the project plan
PRJ	SCHEDULED_EFFORT	The scheduled effort defined in the project plan.

Table B-16. Project plans [continued]

Prefix	Tokens	Description
PRJ	SCHEDULED_DURATION	The project plan's scheduled duration.
PRJ	SCHEDULED_FINISH_DATE	The project plan's scheduled finish date.
PRJ	SCHEDULED_START_DATE	The project plan's scheduled start date.
PRJ	SUMMARY_CONDITION	The project plan's summary condition.
PRJ	WORKBENCH_PROJECT_URL	The URL to access this project plan in the Workbench.

Table B-17. Project plan details

Prefix	Tokens*	Description
PRJD	PROJECT_DETAIL_ID	The ID of the project plan detail in the table KDRV_PROJECT_ DETAILS.
PRJD	PROJECT_ID	The ID of the project plan detail in the table KDRV_PROJECT_ DETAILS.

* Parameters are accessible with this prefix (similar to request detail): [PRJD.P.CUSOM_TOKEN].

Table B-18. Releases

Prefix	Tokens	Description
REL	RELEASE_ID	The ID of the release in the table KREL_RELEASES.
REL	RELEASE_NAME	The name of the release.
REL	RELEASE_STATUS	The status of the release.
REL	CREATED_BY	The ID of the user who created the release.
REL	CREATED_BY_USERNAME	The Mercury IT Governance Center username of the user who created the release.
REL	LAST_UPDATED_BY	The ID of the user who last updated the release.
REL	LAST_UPDATED_BY_ USERNAME	The Mercury IT Governance Center username of the user who last updated the release.
REL	LAST_UPDATE_DATE	The date that the release was last updated.
REL	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.
REL	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.

Table B-18. Releases [continued]

Prefix	Tokens	Description
REL	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
REL	MOST_RECENT_NOTE_ TEXT	Text of the most recent (chronological) note.
REL	RELEASE_MANAGER	The Mercury IT Governance Center user who is designated the release manager.
REL	RELEASE_TEAM	The group of Mercury IT Governance Center users associated with the release.
REL	RELEASE_GROUP	The high level categorization of the release.
REL	DESCRIPTION	The description of the release.
REL	NOTES	The notes contained within the release.

Table B-19. Requests

Prefix	Token	Description
REQ	APPLICATION_CODE	The validation value code for the application that the request is assigned to.
REQ	APPLICATION_NAME	The validation value meaning of the application that the request is assigned to.
REQ	ASSIGNED_TO_EMAIL	The email address of the user the request has been assigned to.
REQ	ASSIGNED_TO_GROUP_ID	The ID of the security group that the request has been assigned to.
REQ	ASSIGNED_TO_GROUP_NAME	The name of the security group that the request has been assigned to.
REQ	ASSIGNED_TO_USERNAME	The Mercury IT Governance Center username of the user that the request has been assigned to.
REQ	ASSIGNED_TO_USER_ID	The ID of the user that the request has been assigned to.
REQ	COMPANY	The Company employing the user that created the request.
REQ	COMPANY_NAME	The name of the Company employing the user that created the request.
REQ	CONTACT_EMAIL	The email address of the Contact for the request.
REQ	CONTACT_NAME	The full name of the Contact for the request.
REQ	CONTACT_PHONE_NUMBER	The phone number of the Contact for the request.

Table B-19. Requests [continued]

Prefix	Token	Description
REQ	CREATED_BY	The ID of the user that created the request.
REQ	CREATED_BY_EMAIL	The email address of the user that created the request.
REQ	CREATED_BY_USERNAME	The Mercury IT Governance Center username of the user that created the request.
REQ	CREATION_DATE	The date the request was created.
REQ	DEPARTMENT_CODE	The validation value code of the department for the request.
REQ	DEPARTMENT_NAME	The validation value meaning of the department for the request.
REQ	DESCRIPTION	The description of the request.
REQ	LAST_UPDATED_BY	The ID of the user that last updated the request.
REQ	LAST_UPDATED_BY_EMAIL	The email address of the user that last updated the request.
REQ	LAST_UPDATED_BY_ USERNAME	The Mercury IT Governance Center username of the user that last updated the request.
REQ	LAST_UPDATE_DATE	The date the request was last updated.
REQ	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.
REQ	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.
REQ	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
REQ	MOST_RECENT_NOTE_TEXT	Text of the most recent (chronological) note.
REQ	MOST_RECENT_NOTE_TYPE	Type of the most recent (chronological) note (USER or FIELD CHANGE).
REQ	MOST_RECENT_NOTE_ CONTEXT	In the case of requests, this is the request status; blank in all other cases.
REQ	MOST_RECENT_USER_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent user note.
REQ	MOST_RECENT_USER_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent user note.
REQ	MOST_RECENT_USER_NOTE_ AUTHORED_DATE	Date of the most recent user note.

Table B-19. Requests [continued]

Prefix	Token	Description
REQ	MOST_RECENT_USER_NOTE_TEXT	Text of the most recent user note.
REQ	MOST_RECENT_USER_NOTE_TYPE	Type of the most recent user note (USER or FIELD CHANGE).
REQ	MOST_RECENT_USER_NOTE_CONTEXT	Status of the request.
REQ	NOTES	All notes for the request.
REQ	PERCENT_COMPLETE	The percent complete of the request.
REQ	PRIORITY_CODE	The validation value code of the request priority.
REQ	PRIORITY_NAME	The validation value meaning of the request priority.
REQ	PROJECT_CODE	The validation value code of the project plan the request belongs to.
REQ	PROJECT_NAME	The validation value meaning of the project plan the request belongs to.
REQ	SUBMIT_DATE	The date that the request was submitted.
REQ	REQUEST_GROUP_CODE	The code for the request group.
REQ	REQUEST_GROUP_NAME	The name of the request group.
REQ	REQUEST_ID	The ID of the request in the table KCRT_REQUESTS.
REQ	REQUEST_ID_LINK	Shows up as a standard hyperlink to the request in HTML-format notifications.
REQ	REQUEST_SUB_TYPE_ID	The ID of the sub-type for the request.
REQ	REQUEST_SUB_TYPE_NAME	The name of the sub-type for the request.
REQ	REQUEST_TYPE_ID	The ID of the request type of the request.
REQ	REQUEST_TYPE_NAME	The name of the request type of the request.
REQ	REQUEST_URL	URL of the request in standard interface.
REQ	STATUS_ID	The ID of the status of the request.
REQ	STATUS_NAME	The status of the request.
REQ	WORKBENCH_REQUEST_TYPE_URL	The URL of the request type in the Workbench.
REQ	WORKFLOW_ID	The ID of the workflow used by the request.
REQ	WORKFLOW_NAME	The name of the workflow used by the request.

Table B-20. Request details

Prefix*	Tokens	Description
REQD	CREATED_BY	The ID of the user who created the request detail.
REQD	CREATION_DATE	The date the request detail was created.
REQD	LAST_UPDATED_BY	The ID of the user that last updated the request detail.
REQD	LAST_UPDATE_DATE	The date the request detail was last updated.
REQD	REQUEST_DETAIL_ID	The ID for the request detail in the table KCRT_REQUEST_ DETAILS.
REQD	REQUEST_ID	The ID of the request for the request detail.
REQD	REQUEST_TYPE_ID	The ID of the request type for the request detail.

* Prefix is mainly used for accessing custom fields: `[REQD.P.CUSTOM_TOKEN]`

Table B-21. Request pending

Prefix	Tokens	Description
REQ.PEND	ID	The ID of the entity that is being blocked by the request.
REQ.PEND	NAME	The name of the entity that is being blocked by the request.
REQ.PEND	DETAIL	Detail information for the entity that is being blocked by the request.
REQ.PEND	DESCRIPTION	The description of the entity that is being blocked by the request.
REQ.PEND	STATUS_ID	The ID of the state or code of the status of the entity that is being blocked by the request.
REQ.PEND	STATUS_NAME	The name of the status (or state) of the entity that is being blocked by the request.
REQ.PEND	STATE	The name of the state of the entity of the request that is being blocked by the request.
REQ.PEND	ASSIGNED_TO_USERNAME	The name of the assigned user (or resource) of the entity that is being blocked by the request.
REQ.PEND	ASSIGNED_TO_USER_ID	The username of the assigned user (or resource) of the entity that is being blocked by the request.

Table B-21. Request pending [continued]

Prefix	Tokens	Description
REQ.PEND	ASSIGNED_TO_GROUP_NAME	The name of the assigned group (or resource group) of the entity that is being blocked by the request.
REQ.PEND	ASSIGNED_TO_GROUP_ID	The ID of the assigned group (or resource group) of the entity that is being blocked by the request.
REQ.PEND	RESOURCE_USERNAME	The name of the resource associated with the entity that is being blocked by the request.
REQ.PEND	RESOURCE_ID	The username of the assigned user (or resource) associated with the entity that is being blocked by the request.
REQ.PEND	RESOURCE_GROUP_NAME	The name of the assigned group (or resource group) associated with the entity that is being blocked by the request.
REQ.PEND	RESOURCE_GROUP_ID	The ID of the assigned group (or resource group) associated with the entity that is being blocked by the request.
REQ.PEND	PERCENT_COMPLETE	The current percent complete value associated with the entity that is being blocked by the request.
REQ.PEND	ENTITY_TYPE_ID	The ID of the type of entity that is being blocked by the request.
REQ.PEND	ENTITY_TYPE_NAME	The name of the type of entity that is being blocked by the request.

Table B-22. Report submissions

Prefix	Tokens	Description
RP	CREATED_BY	The ID of the user who submitted the report.
RP	CREATION_DATE	The date the report was submitted.
RP	FILENAME	The filename for the report. This file name is found in the REPORT_URL.
RP	LAST_UPDATED_BY	The ID of the user that last updated the report submission.
RP	LAST_UPDATE_DATE	The date the report submission was last updated.
RP	NEW_STATUS	The visible value for the report’s new status.
RP	NEW_STATUS_CODE	The code for the report’s new status.
RP	OLD_STATUS	The visible value for the report’s old status.

Table B-22. Report submissions [continued]

Prefix	Tokens	Description
RP	OLD_STATUS_CODE	The code for the report's old status.
RP	REPORT_LOG_URL	The Web address where the report log is located.
RP	REPORT_SUBMISSION_ID	The ID of the report submission in the table KNTA_ REPORT_SUBMISSIONS.
RP	REPORT_TYPE_NAME	The name of the report type of the report submission.
RP	REPORT_TYPE_ID	The ID of the report type of the report submission.
RP	REPORT_URL	The Web address where the report output is located.
RP	STATUS	The status of the report submission.
RP	STATUS_CODE	The validation value code for the status of the report submission.
RP	WORKBENCH_REPORT_TYPE_URL	The URL of the report type in the Workbench.

Table B-23. Resource pools

Prefix	Tokens	Description
RSCP	ACTIVE_FLAG	The active flag of the resource pool.
RSCP	CREATED_BY	The username of the user who created the resource pool.
RSCP	CREATION_DATE	The date that the resource pool was created.
RSCP	DESCRIPTION	The description of the resource pool.
RSCP	END_PERIOD	The end period of the resource pool.
RSCP	INITIATION_REQ	The initiation request ID of the resource pool.
RSCP	PERIOD_SIZE	The period size of the resource pool.
RSCP	RESOURCE_POOL_URL	The URL to view this resource pool.
RSCP	RSC_POOL_ID	The ID of the resource pool in table KRSC_RSC_ POOLS.
RSCP	RSC_POOL_IS_FOR_ENTITY_NAME	The entity name to which the resource pool is linked (program or org unit).
RSCP	RSC_POOL_IS_FOR_ID	The ID of the program or org unit to which the resource pool is linked.

Table B-23. Resource pools [continued]

Prefix	Tokens	Description
RSCP	RSC_POOL_IS_FOR_NAME	The name of the program or org unit to which the resource pool is linked.
RSCP	RSC_POOL_NAME	The name of the resource pool.
RSCP	START_PERIOD	The start period of the resource pool.
RSCP	STATUS_CODE	The status code of the resource pool.
RSCP	STATUS_NAME	The status name of the resource pool.

Table B-24. Security groups

Prefix	Tokens	Description
SG	CREATED_BY	The ID of the user who created the security group.
SG	CREATION_DATE	The date the security group was created.
SG	DESCRIPTION	The description for the security group.
SG	LAST_UPDATED_BY	The ID of the user that last updated the security group.
SG	LAST_UPDATE_DATE	The date the security group was last updated.
SG	SECURITY_GROUP_ID	The ID of the security group in the table KNTA_SECURITY_ GROUPS.
SG	SECURITY_GROUP_NAME	The name of the security group.

Table B-25. Skill

Prefix	Tokens	Description
SKL	AVERAGE_COST_RATE	The average cost rate associated with the skill.
SKL	CREATED_BY	The user ID that created the skill.
SKL	CREATED_BY_USERNAME	The name of the user that created the skill.
SKL	CREATION_DATE	The date that the skill was created.
SKL	SKILL_CATEGORY_CODE	The lookup code of skill Category (lookup type = RSC - skill Category).
SKL	SKILL_CATEGORY_NAME	The name of the skill category.
SKL	SKILL_ID	The ID of the skill in table KRSC_SKILLS.
SKL	SKILL_NAME	The name of the skill.

Table B-26. Staffing profile

Prefix	Tokens	Description
STFP	ACTIVE_FLAG	The active flag of the staffing profile.
STFP	CREATED_BY	The username of the user who created the staffing profile.
STFP	CREATION_DATE	The date that the staffing profile was created.
STFP	DESCRIPTION	The description of the staffing profile.
STFP	END_PERIOD	The end period of the staffing profile.
STFP	INITIATION_REQ	The initiation request ID of the staffing profile.
STFP	PERIOD_SIZE	The period size of the staffing profile.
STFP	STAFFING_PROFILE_URL	The URL to view this staffing profile.
STFP	STAFF_PROF_ID	The ID of the staffing profile in table KRSC_STAFF_ PROFS.
STFP	STAFF_PROF_IS_FOR_ ENTITY_NAME	The entity name to which the staffing profile is linked.
STFP	STAFF_PROF_IS_FOR_ID	The ID of the project plan, program or org unit to which the staffing profile is linked.
STFP	STAFF_PROFL_IS_FOR_ NAME	The name of the project plan, program or org unit to which the staffing profile is linked (project plan, program, or org unit).
STFP	STAFF_PROF_NAME	The name of the staffing profile.
STFP	START_PERIOD	The start period of the staffing profile.
STFP	STATUS_CODE	The status code of the staffing profile.
STFP	STATUS_NAME	The status name of the staffing profile.

Table B-27. System

Prefix	Tokens	Description
SYS	DATE	The date at the time the token is parsed.
SYS	NEWLINE	A new line character.
SYS	TIME_STAMP	A date and time stamp at the time the token is parsed.
SYS	UNIQUE_IDENTIFIER	Used to obtain a unique number from the database. It can be used to generate unique filenames, etc. It is often necessary to use with the 'ksc_set' special command.

Table B-27. System [continued]

Prefix	Tokens	Description
SYS	UNIX_NEWLINE	The UNIX new line character.
SYS	USERNAME	The Mercury IT Governance Center username of the user currently logged onto Mercury IT Governance Center.
SYS	USER_ID	The ID of the user currently logged onto Mercury IT Governance Center.

Table B-28. Tasks

Prefix	Tokens	Description
TSK	ACTUAL_DURATION	The actual duration of the task.
TSK	ACTUAL_EFFORT	The actual effort associated with the task.
TSK	ACTUAL_FINISH_DATE	The actual finish date of the task.
TSK	ACTUAL_START_DATE	The actual start date of the task.
TSK	CONFIDENCE_CODE	The code of the confidence value entered by the user.
TSK	CONFIDENCE_NAME	The name of the confidence value entered by the user.
TSK	CONSTRAINT_DATE	The task’s constraint date.
TSK	CREATED_BY	The user who created the task.
TSK	CREATED_BY_EMAIL	The email address of the user who created the task.
TSK	CREATED_BY_USERNAME	The username of the person who created the task.
TSK	CREATION_DATE	The creation date of the task.
TSK	DEPARTMENT_CODE	The code of the department value entered by the user.
TSK	DEPARTMENT_NAME	The name of the department value entered by the user.
TSK	DESCRIPTION	The description of the task.
TSK	ESTIMATED_REMAINING_ DURATION	The estimated remaining duration of the task.
TSK	ESTIMATED_REMAINING_EFFORT	The estimated remaining effort involved in the task.
TSK	ESTIMATED_FINISH_DATE	The estimated finish date of the task.
TSK	HAS_EXCEPTIONS	The flag to show whether or not the task has exceptions.
TSK	LAST_UPDATE_DATE	The date the task was last updated.

Table B-28. Tasks [continued]

Prefix	Tokens	Description
TSK	LAST_UPDATED_BY	The last person to update the task.
TSK	LAST_UPDATED_BY_EMAIL	The email address of the last person to update the task.
TSK	LAST_UPDATED_BY_USERNAME	The username of the last person to update the task.
TSK	MASTER_PROJECT_ID	The ID of the master project.
TSK	MASTER_PROJECT_NAME	The name of the master project.
TSK	MOST_RECENT_NOTE_ AUTHOR_FULL_NAME	First and last name of the author of the most recent (chronological) note.
TSK	MOST_RECENT_NOTE_ AUTHOR_USERNAME	Username of the author of the most recent (chronological) note.
TSK	MOST_RECENT_NOTE_ AUTHORED_DATE	Date of the most recent (chronological) note.
TSK	MOST_RECENT_NOTE_TEXT	Text of the most recent (chronological) note.
TSK	PARENT_PROJECT_ID	The ID of the parent project plan.
TSK	PARENT_PROJECT_NAME	The name of the parent project plan.
TSK	PERCENT_COMPLETE	The task's completed percentage.
TSK	PRIORITY	The priority of the task.
TSK	PROJECT_PATH	The project plan path. Hierarchy of parent project plans that contain this task.
TSK	PROJECT_TEMPLATE	The name of the project template used to create the project plan containing the task.
TSK	PROJECT_TYPE_CODE	Returns TASK for tasks and PROJECT for project plans.
TSK	RESOURCE_ID	The ID of the resource assigned to the task.
TSK	RESOURCE_EMAIL	The email address of the resource.
TSK	RESOURCE_GROUP_ID	The ID of the resource group assigned to the task.
TSK	RESOURCE_GROUP_NAME	The name of the resource group assigned to the task.
TSK	RESOURCE_USERNAME	The username of the resource.
TSK	SCHEDULED_EFFORT	The scheduled effort involved in the task.
TSK	SCHEDULED_DURATION	The task's scheduled duration.
TSK	SCHEDULED_FINISH_DATE	The task's scheduled finish date.

Table B-28. Tasks [continued]

Prefix	Tokens	Description
TSK	SCHEDULED_START_DATE	The task's scheduled start date.
TSK	SCHEDULING CONSTRAINT	The task's scheduling constraint.
TSK	TASK_CATEGORY	The predefined category the task belongs to.
TSK	TASK_ID	The number that uniquely identifies the task (same as TASK_NUMBER). This corresponds to the PROJECT_ID column in table KDRV_PROJECTS.
TSK	TASK_NAME	The name of the task.
TSK	TASK_NAME_LINK	Standard hyperlink to the task in HTML-format notifications.
TSK	TASK_NUMBER	The number that uniquely identifies the task (same as TASK_ID).
TSK	TASK_STATE	The task state.
TSK	TASK_URL	The URL for the task Detail page.
TSK	WORKBENCH_TASK_URL	The URL to access this task in the Workbench.

Table B-29. Tasks pending

Prefix	Tokens	Description
TSK.PEND	ID	The ID of the entity that is being blocked by the task.
TSK.PEND	NAME	The name of the entity that is being blocked by the task.
TSK.PEND	DETAIL	Detail information for the entity that is being blocked by the task as shown in the References panel.
TSK.PEND	DESCRIPTION	The description of the entity that is being blocked by the task.
TSK.PEND	STATUS_ID	The ID of the state or the code of the status of the entity that is being blocked by the task.
TSK.PEND	STATUS_NAME	The name of the status (or state) of the entity that is being blocked by the task.
TSK.PEND	STATE	The name of the state of the entity that is being blocked by the task.
TSK.PEND	ASSIGNED_TO_USERNAME	The name of the assigned user (or resource) of the entity that is being blocked by the task.

Table B-29. Tasks pending [continued]

Prefix	Tokens	Description
TSK.PEND	ASSIGNED_TO_USER_ID	The username of the assigned user (or resource) of the entity that is being blocked by the task.
TSK.PEND	ASSIGNED_TO_GROUP_NAME	The name of the assigned group (or resource group) of the entity that is being blocked by the task.
TSK.PEND	ASSIGNED_TO_GROUP_ID	The ID of the assigned group (or resource group) of the entity that is being blocked by the task.
TSK.PEND	RESOURCE_USERNAME	The name of the resource associated with the entity that is being blocked by the task.
TSK.PEND	RESOURCE_ID	The username of the resource (or assigned user) associated with the entity that is being blocked by the task.
TSK.PEND	RESOURCE_GROUP_NAME	The name of the resource group (or assigned user) associated with the entity that is being blocked by the task.
TSK.PEND	RESOURCE_GROUP_ID	The ID of the resource group (or assigned group) associated with the entity that is being blocked by the task.
TSK.PEND	PERCENT_COMPLETE	The current percent complete value associated with the entity that is being blocked by the task.
TSK.PEND	ENTITY_TYPE_ID	The ID of the type of entity that is being blocked by the task.
TSK.PEND	ENTITY_TYPE_NAME	The name of the type of entity that is being blocked by the task.

Table B-30. Users

Prefix	Tokens	Description
USR	AUTHENTICATION_MODE_CODE	The authentication mode for the user (such as LDAP).
USR	AUTHENTICATION_MODE_NAME	The authentication mode for the user (such as LDAP).
USR	COMPANY	The Company employing the user.
USR	COMPANY_NAME	The name of the Company employing the user.
USR	COST_RATE	The cost rate of the user ($/hour - subject to security of user evaluating the token).
USR	CREATED_BY	The ID of the user that created the user.

Table B-30. Users [continued]

Prefix	Tokens	Description
USR	CREATED_BY_USERNAME	The Mercury IT Governance Center username of the user that created the user.
USR	CREATION_DATE	The date the user was created.
USR	DEPARTMENT_CODE	The lookup code of the department the user belongs to (lookup type = DEPT).
USR	DEPARTMENT_NAME	The name of the department that the user belongs to.
USR	EMAIL_ADDRESS	The email address of the user.
USR	END_DATE	The date the user is made inactive in the application.
USR	FIRST_NAME	The first name of the user.
USR	LAST_NAME	The last name of the user.
USR	LAST_UPDATED_BY	The ID of the user that last updated the user.
USR	LAST_UPDATED_BY_USERNAME	The Mercury IT Governance Center username of the user that last updated the user.
USR	LAST_UPDATE_DATE	The date the user was last updated.
USR	LOCATION_CODE	The lookup code of the user's location (lookup type = RSC - Location).
USR	LOCATION_NAME	The name of the user's location.
USR	MANAGER_USERNAME	The username of the user's manager.
USR	MANAGER_USER_ID	The ID of the user's manager.
USR	PASSWORD	The password for the user to use to log onto Mercury IT Governance Center. This value is encrypted.
USR	PASSWORD_EXPIRATION_DATE	The date the password needs to be reset for the user.
USR	PASSWORD_EXPIRATION_DAYS	The number of days until the password must be reset for the user.
USR	PHONE_NUMBER	The phone number of the user.
USR	PRIMARY_SKILL_ID	The ID of the primary skill associated with the user.
USR	PRIMARY_SKILL_NAME	The name of the primary skill associated with the user.
USR	REGIONAL_CALENDAR	The name of the regional calendar for the user.
USR	RESOURCE_CATEGORY_CODE	The lookup code of resource category (lookup type = RSC - Category) to which the user belongs.
USR	RESOURCE_CATEGORY_NAME	The name of the category to which the user belongs.

Table B-30. Users [continued]

Prefix	Tokens	Description
USR	RESOURCE_TITLE_CODE	the lookup code of the user's resource title (lookup type = RSC - Resource Title).
USR	RESOURCE_TITLE_NAME	The name of the user's resource title.
USR	START_DATE	The date the user is made active in the application.
USR	USERNAME	The username for the user to use to log onto Mercury IT Governance Center.
USR	USER_ID	The ID of the user in the table KNTA_USERS.
USR	WORKLOAD_CAPACITY	The workload capacity of the user (% of FTE)

Table B-31. Validations

Prefix	Tokens	Description
VAL	COMPONENT_TYPE	The component type associated with the validation.
VAL	CREATED_BY	The ID of the user that created the validation.
VAL	CREATION_DATE	The date the validation was created.
VAL	DESCRIPTION	The description of the validation.
VAL	LAST_UPDATED_BY	The ID of the user that last updated the validation.
VAL	LAST_UPDATE_DATE	The date the validation was last updated.
VAL	LOOKUP_TYPE	The lookup type associated with the validation (if applicable).
VAL	VALIDATION_ID	The ID of the validation in the table KNTA_ VALIDATIONS.
VAL	VALIDATION_NAME	The name of the validation.
VAL	VALIDATION_SQL	The SQL statement associated with the validation (if applicable).
VAL	WORKBENCH_VALIDATION_URL	The URL for the validation in the Workbench.

Table B-32. Validation values

Prefix	Tokens	Description
VALUE	CREATED_BY	The ID of the user that created the value.
VALUE	CREATION_DATE	The date the value was created.

Table B-32. Validation values [continued]

Prefix	Tokens	Description
VALUE	DEFAULT_FLAG	The flag to indicate whether the value is the default value for the associated lookup type.
VALUE	DESCRIPTION	The description of the value.
VALUE	ENABLED_FLAG	The flag to indicate whether the value is enabled for selection in a drop-down list.
VALUE	LAST_UPDATED_BY	The ID of the user that last updated the value.
VALUE	LAST_UPDATE_DATE	The date the value was last updated.
VALUE	LOOKUP_CODE	The code associated with the value.
VALUE	LOOKUP_TYPE	The lookup type the value belongs to.
VALUE	MEANING	The meaning associated with the value.
VALUE	SEQ	The sequence relative to other values in the associated lookup type in which this value will be displayed in a drop-down list.

Table B-33. Workflows

Prefix	Tokens	Description
WF	CREATED_BY	The ID of the user that created the workflow.
WF	CREATION_DATE	The date the workflow was created.
WF	DESCRIPTION	The description of the workflow.
WF	ENABLED_FLAG	The flag indicating whether the workflow is enabled and available to use in packages and/or requests.
WF	FIRST_WORKFLOW_STEP_ID	The ID of the first workflow step in the workflow.
WF	FIRST_WORKFLOW_STEP_NAME	The name of the first workflow step in the workflow.
WF	ICON_NAME	The name of the workflow step icon.
WF	LAST_UPDATED_BY	The ID of the user that last updated the workflow.
WF	LAST_UPDATE_DATE	The date the workflow was last updated.
WF	PRODUCT_SCOPE_CODE	The validation value code for the product scope of the workflow.
WF	REOPEN_WORKFLOW_STEP_ID	The ID of the reopened workflow step.
WF	REOPEN_WORKFLOW_STEP_NAME	The name of the reopened workflow step.
WF	SUBWORKFLOW_FLAG	An indicator that specifies whether this workflow can be used as a Subworkflow.

Table B-33. Workflows

Prefix	Tokens	Description
WF	WORKFLOW_ID	The ID of the workflow defined in the table KWFL_ WORKFLOWS.
WF	WORKFLOW_NAME	The name of the workflow.
WF	WORKBENCH_WORKFLOW_URL	The URL to open the workflow in the Workbench.

Table B-34. Workflow steps

Prefix	Tokens	Description
WFS	ACTION_BUTTON_LABEL	The label displayed on the package or request action button for the workflow step.
WFS	AVERAGE_LEAD_TIME	The average lead time in days defined for the workflow step.
WFS	CREATED_BY	The ID of the user that created the workflow step.
WFS	CREATION_DATE	The date the workflow step was created.
WFS	DESCRIPTION	The description of the workflow step.
WFS	DEST_ENV_GROUP_ID	The ID of the destination environment group for the workflow step.
WFS	DEST_ENV_GROUP_NAME	The name of the destination environment group for the workflow step.
WFS	DEST_ENVIRONMENT_ID	The ID of destination environment for the workflow step.
WFS	DEST_ENVIRONMENT_NAME	The name of the destination environment for the workflow step.
WFS	ENABLED_FLAG	The flag indicating whether the workflow step is enabled and able to be traversed in a package or request.
WFS	GL_ARCHIVE_FLAG	For GL object migration, the flag indicating whether to save the GL object being migrated to the GL*Migrator archive.
WFS	INFORMATION_URL	The workflow step's information URL.
WFS	JUMP_RECEIVE_LABEL_CODE	The code for a Jump/Receive workflow step.
WFS	JUMP_RECEIVE_LABEL_NAME	The name of a Jump/Receive workflow step.
WFS	LAST_UPDATED_BY	The ID of the user that last updated the workflow step.

Table B-34. Workflow steps [continued]

Prefix	Tokens	Description
WFS	LAST_UPDATE_DATE	The date the workflow step was last updated.
WFS	OM_ARCHIVE_FLAG	For AOL object migration, the flag indicating whether to save the AOL object being migrated to the Object*Migrator archive.
WFS	PARENT_ASSIGNED_TO_GROUP_ID	The ID of the security group that the current package or request is assigned to (determined by context at time of evaluation).
WFS	PARENT_ASSIGNED_TO_GROUP_NAME	The security group that the current package or request is assigned to (determined by context at time of evaluation).
WFS	PARENT_ASSIGNED_TO_USERNAME	The name of the user that the current package or request is assigned to (determined by context at time of evaluation).
WFS	PARENT_ASSIGNED_TO_USER_ID	The ID of the user that the current package or request is assigned to (determined by context at time of evaluation).
WFS	PARENT_STATUS	The validation value code of the status of the request that is using the workflow step.
WFS	PARENT_STATUS_NAME	The validation value meaning of the status of the request that is using the workflow step.
WFS	PRODUCT_SCOPE_CODE	The validation value code for the product scope of the workflow containing the workflow step.
WFS	RESULT_WORKFLOW_PARAMETER_ID	The ID of the workflow parameter that the result of the workflow step is written to.
WFS	RESULT_WORKFLOW_PARAMETER_NAME	The name of the workflow parameter that the result of the workflow step is written to.
WFS	SORT_ORDER	The display sequence of the workflow step relative to all other steps in the workflow.
WFS	SOURCE_ENV_GROUP_ID	The ID of the source environment group for the workflow step.
WFS	SOURCE_ENV_GROUP_NAME	The name of the source environment group for the workflow step.
WFS	SOURCE_ENVIRONMENT_ID	The ID of the source environment for the workflow step.
WFS	SOURCE_ENVIRONMENT_NAME	The name of the source environment for the workflow step.

Table B-34. Workflow steps [continued]

Prefix	Tokens	Description
WFS	STEP_NAME	The name of the workflow step.
WFS	STEP_NO	The display sequence of the workflow step relative to all other steps in the workflow.
WFS	STEP_SOURCE_NAME	The name of the workflow step source.
WFS	STEP_TYPE_NAME	The name of the workflow step source type.
WFS	WORKFLOW_ID	The ID of the workflow containing the workflow step.
WFS	WORKFLOW_NAME	The name of the workflow containing the workflow step.
WFS	WORKFLOW_STEP_ID	The ID of the workflow step in the table KWFL_ WORKFLOW_STEPS.

Table B-35. Workflow step transaction

Prefix	Tokens	Description
WST	CONCURRENT_REQUEST_ID	The ID of the concurrent request that was launched in Oracle Applications.
WST	CREATED_BY	The ID of the user that created the step transaction.
WST	CREATION_DATE	The date the step transaction was created.
WST	ERROR_MESSAGE	The error message for the step transaction.
WST	EXECUTION_BATCH_ID	The ID of the execution batch for the workflow step.
WST	HIDDEN_STATUS	The hidden value for the status of the step transaction.
WST	LAST_UPDATED_BY	The ID of the user that last updated the step transaction.
WST	LAST_UPDATED_BY_EMAIL	The email address of the user that last updated the step transaction.
WST	LAST_UPDATED_BY_USERNAME	The Mercury IT Governance Center username of the user that last updated the step transaction.
WST	LAST_UPDATE_DATE	The date the step transaction was last updated.
WST	NEW_HIDDEN_STATUS	The new hidden value for the status of the step transaction.
WST	NEW_STATUS	The new status of the step transaction.

Table B-35. Workflow step transaction [continued]

Prefix	Tokens	Description
WST	OLD_HIDDEN_STATUS	The old hidden value for the status of the step transaction.
WST	OLD_STATUS	The old status of the step transaction.
WST	STATUS	The status of the step transaction.
WST	STEP_TRANSACTION_ID	The ID of the step transaction in the table KWFL_ STEP_TRANSACTIONS.
WST	TIMEOUT_DATE	The date that the step transaction times out.
WST	USER_COMMENT	The user comment for the step transaction.
WST	WORKFLOW_ID	The ID of the workflow for the step transaction.
WST	WORKFLOW_STEP_ID	The ID of the workflow step for the step transaction.

Field Group Tokens

Field groups can be attached to request header types to enable additional pre-configured fields on requests. Field groups are often delivered as a part of Mercury IT Governance Center best practice functionality. You will only have access to field groups associated with products that are licensed at your site.

Use *Table B-36* as a quick reference guide to jump to the desired location.

Table B-36. Field group token tables

Table	Page
Table B-37, Demand Management field group tokens	214
Table B-38, Master project reference on request field group tokens	214
Table B-39, PFM asset field group tokens	214
Table B-40, PFM project field group tokens	215
Table B-41, PFM proposal field group tokens	216
Table B-42, PMO field group tokens	216

Table B-36. Field group token tables

Table	Page
Table B-43, Program reference on request field group tokens	217
Table B-44, Work item field group tokens	217

Table B-37. Demand Management field group tokens

Field	Token
SLA Level	KNTA_SLA_LEVEL
SLA Violation Data	KNTA_SLA_VIOLATION_DATE
Service Request Date	KNTA_SLA_SERV_REQUESTED_ON
Service Satisfied Date	KNTA_SLA_SERV_SATISFIED_ON
Estimated Start Date	KNTA_EST_START_DATE
Estimated Effort	KNTA_EFFORT
Reject Date	KNTA_REJECTED_DATE
Demand Satisfied Date	KNTA_DEMAND_SATISFIED_DATE

Table B-38. Master project reference on request field group tokens

Field	Token
Master Project	KNTA_MASTER_PROJ_REF

Table B-39. PFM asset field group tokens

Field	Token
Business Unit	KNTA_BUSINESS_UNIT
Asset Name	KNTA_PROJECT_NAME
Asset Health	KNTA_PROJECT_HEALTH
Project Class	KNTA_PROJECT_CLASS
Asset Class	KNTA_ASSET_CLASS
Business Objective	KNTA_BUSINESS_OBJECTIVE
Project Plan	KNTA_PROJECT_PLAN
Budget	KNTA_BUDGET

Table B-39. PFM asset field group tokens [continued]

Field	Token
Financial Benefit	KNTA_FINANCIAL_BENEFIT
Staffing Profile	KNTA_STAFFING_PROFILE
Net Present Value	KNTA_NET_PRESENT_VALUE
Value Rating	KNTA_VALUE_RATING
Risk Rating	KNTA_RISK_RATING
Return on Investment	KNTA_RETURN_ON_INVESTMENT
Custom Field Value	KNTA_CUSTOM_FIELD_VALUE
Total Score	KNTA_TOTAL_SCORE
Discount Rate	KNTA_DISCOUNT_RATE

Table B-40. PFM project field group tokens

Field	Token
Business Unit	KNTA_BUSINESS_UNIT
Project Name	KNTA_PROJECT_NAME
Project Health	KNTA_PROJECT_HEALTH
Project Class	KNTA_PROJECT_CLASS
Asset Class	KNTA_ASSET_CLASS
Business Objective	KNTA_BUSINESS_OBJECTIVE
Project Plan	KNTA_PROJECT_PLAN
Project Manager	KNTA_PROJECT_MANAGER
Budget	KNTA_BUDGET
Financial Benefit	KNTA_FINANCIAL_BENEFIT
Staffing Profile	KNTA_STAFFING_PROFILE
Net Present Value	KNTA_NET_PRESENT_VALUE
Value Rating	KNTA_VALUE_RATING
Risk Rating	KNTA_RISK_RATING
Custom Field Value	KNTA_CUSTOM_FIELD_VALUE
Return on Investment	KNTA_RETURN_ON_INVESTMENT

Table B-40. PFM project field group tokens

Field	Token
Total Score	KNTA_TOTAL_SCORE
Discount Rate	KNTA_DISCOUNT_RATE

Table B-41. PFM proposal field group tokens

Field	Token
Business Unit	KNTA_BUSINESS_UNIT
Project Name	KNTA_PROJECT_NAME
Project Class	KNTA_PROJECT_CLASS
Asset Class	KNTA_ASSET_CLASS
Business Objective	KNTA_BUSINESS_OBJECTIVE
Project Template	KNTA_PROJECT_TEMPLATE
Project Manager	KNTA_PROJECT_MANAGER
Budget	KNTA_BUDGET
Expected Benefit	KNTA_FINANCIAL_BENEFIT
Staffing Profile	KNTA_STAFFING_PROFILE
Net Present Value	KNTA_NET_PRESENT_VALUE
Value Rating	KNTA_VALUE_RATING
Risk Rating	KNTA_RISK_RATING
Return on Investment	KNTA_RETURN_ON_INVESTMENT
Custom Field Value	KNTA_CUSTOM_FIELD_VALUE
Total Score	KNTA_TOTAL_SCORE
Discount Rate	KNTA_DISCOUNT_RATE

Table B-42. PMO field group tokens

Field	Token
Escalation Level	KNTA_ESCALATION_LEVEL
Role Description	KNTA_ROLE_DESCRIPTION
Risk Impact Level	KNTA_RISK_IMPACT_LEVEL

Table B-42. PMO field group tokens

Field	Token
Probability	KNTA_PROBABILITY
CR Level	KNTA_CR_LEVEL
Business Impact Severity	KNTA_IMPACT_SEVERITY

Table B-43. Program reference on request field group tokens

Field	Token
Program	KNTA_PROGRAM_REFERENCE

Table B-44. Work item field group tokens

Field	Token
Scheduled Start Date	KNTA_USR_SCHED_START_DATE
Actual Start Date	KNTA_USR_ACTUAL_START_DATE
Scheduled Finish Date	KNTA_USR_SCHED_FINISH_DATE
Actual Finish Date	KNTA_USR_ACTUAL_FINISH_DATE
Scheduled Duration	KNTA_SCHED_DURATION
Actual Duration	KNTA_ACTUAL_DURATION
Scheduled Effort	KNTA_SCHED_EFFORT
Actual Effort	KNTA_ACTUAL_EFFORT
Workload?	KNTA_WORKLOAD
Workload Category	KNTA_WORKLOAD_CATEGORY
Skill	KNTA_SKILL
Allow External Update of Actual Effort	KNTA_ALLOW_EXTERNAL_UPDATE
_Scheduled Start Date	KNTA_SCHED_START_DATE
_Actual Start Date	KNTA_ACTUAL_START_DATE
_Scheduled Finish Date	KNTA_SCHED_FINISH_DATE
_Actual Finish Date	KNTA_ACTUAL_FINISH_DATE
_Scheduled Effort Over Duration	KNTA_SCHED_EFF_OVER_DUR

Index

A

app server properties 177

auto-complete
- command with delimited output 95
- command with fixed width output 97
- configuring general behavior 82
- configuring the values 94
- example 100
- list of users 93
- long lists 84
- search fields 88
- short lists 83
- user-defined multi-select 99

auto-complete validations 82

C

command conditions 25, 34
- examples 26

command language 25

command steps 25

command with delimited output 95

command with fixed width output 97

commands
- triggering from workflow 23
- validation 82

component types 71
- auto-complete 99
- directory chooser 116
- file chooser 116
- file chooser (static environment override) 117
- file chooser (token-based environment override) 118

contact 178

currency data mask 108

custom data mask 115

D

date field
- valid format 120

directory chooser 116

dynamic list validations 79
- command 82
- SQL 79

E

entity token
- app server properties 177
- command execution 185
- contacts 178
- demand management fields 214
- distributions 179

document management 180
environment applications 183
environments 180
extension 183
notifications 185
organization units 186
package lines 189
package pending 190
program 191
project field group 214
project plan 191
project plan details 194
releases 194
report submissions 199
request details 198
requests 195
requests pending 198
resource pools 200
security groups 201
skills 201
staffing profile 202
tasks 203
tasks pending 205
users 206
workflow steps 210
workflows 209

entity tokens
validation values 208
validations 208

F

field group tokens 213
asset 214
demand management 214
master project reference 214
PMO 216
program reference 217
project 215
proposal 216
work item 217

fields
preview layout 93

file chooser 116
static environment override 117
token-based environment override 118

format masks
for text fields 104

K

ksc_begin_script 159
example 160

ksc_capture_output 164

ksc_clear_exit_value 168

ksc_comment 157

ksc_conc_sub 158
example 158

ksc_connect 138

ksc_connect_dest_client 139
example 139

ksc_connect_dest_server 140
example 140

ksc_connect_source_client 141
example 141

ksc_connect_source_server 142
examples 142

ksc_copy 143

ksc_copy_client_client 143
example 144

ksc_copy_client_server 144
example 145

ksc_copy_client_tmp 148

ksc_copy_script 160

ksc_copy_script_dest_client 161

ksc_copy_script_dest_server 161

ksc_copy_script_source_client 162

ksc_copy_script_source_server 162

ksc_copy_server_client 145
example 146

ksc_copy_server_server 146
example 147

ksc_copy_server_tmp 148

ksc_copy_tmp_client 149

ksc_copy_tmp_server 150
ksc_end_script 159
    example 160
ksc_exit 143
ksc_gl_migrate 165
    example 166
ksc_local_exec 153
ksc_om_migrate 163
    example 164
ksc_parse_jcl 166
ksc_replace 154
ksc_respond 151
ksc_set 154, 166, 167
    example 153, 154, 155
ksc_set_env 155
ksc_set_exit_value 167
ksc_simple_respond 151
    examples 152
ksc_store 154, 156, 166, 167
    example 157
ksc_submit_job 167

N

numeric data mask 106

O

object types
    commands and workflow 23
ownership
    setting for special commands 43

P

percentage data mask 110

R

request field tokens 61
    prefixes 61
request tield fokens
    table components 61

S

special command
    parameters tab 32
special command builder 37
    using to build steps 46
special commands
    adding parameters 33, 40
    building steps with command builder 46
    commands tab 33
    creating new 38
    deleting parameters 43
    editing parameters 42
    header fields 32
    ksc_begin_script 159
    ksc_capture_output 164
    ksc_clear_exit_value 168
    ksc_comment 157
    ksc_conc_sub 158
    ksc_connect 138
    ksc_copy 143
    ksc_copy_script 160
    ksc_end_script 159
    ksc_exit 143
    ksc_gl_migrate 165
    ksc_local_exec 153
    ksc_om_migrate 163
    ksc_parse_jcl 166
    ksc_respond 151
    ksc_run_sql 168
    ksc_set 154, 166, 167
    ksc_set_env 155
    ksc_set_exit_value 167
    ksc_simple_respond 151
    ksc_store 154, 156, 166, 167
    ksc_submit_job 167
    nesting 47
    ownership tab 37
    parameters 169
    setting ownership 43
    used by tab 38
    user interface 30
    using 45
    window 31

- workbench 31

SQL validations 79
- tips 81

static list validations 77

swap mode 92

T

table component validations 121
- adding to request type 132
- column totals 130
- creating rules 126
- defining 122
- rules example 126
- tokens 130

table components
- using tokens in 61

telephone data mask 112

text fields
- configuring 104
- currency 105
- custom format 106
- customizing the data masks 106
- format masks 104
- numeric 105
- percentage 106
- telephone 106

token
- evaluation example 100

Token Builder window 51

token evaluation 66

tokens
- building 56
- default format 55
- environment tokens 64
- explicit entity format 56
- field groups 213
- formats 53
- overview 50
- parameter format 60
- request fields 61
- sub-entity format 63
- user data format 59
- within tokens 57

U

URL to validation 76

user data
- context sensitive in validation 74

V

validations
- auto-complete 82
- command 82
- command with delimited output 95
- command with fixed width output 97
- context sensitive user data and 74
- creating 73
- date format 120
- defined 70
- deleting 77
- directory chooser 116
- dynamic list 79
- editing 75
- file chooser 116
- file chooser (static environment override) 117
- file chooser (token-based environment override) 118
- overview 71
- package and request group 134
- quick link 76
- request type category 135
- seeded 136
- special characters and 135
- SQL 79
- SQL tips 81
- static lists 77
- system 136
- table component 121
- text area 1800 120