Siebel Business Process Framework: Task UI Guide

Version 8.0

December 2006

ORACLE

Contents

Index

1 What's New in This Release

What's New in Siebel Business Process Framework: Task UI Guide, Version 8.0

The *Siebel Business Process Framework: Task UI Guide* is a new book title for 8.0 that covers how to configure a task-based user interface (Task UI) for Oracle's Siebel Business Applications. It discusses the concepts and procedures associated with configuring, deploying, and debugging Task UI applications.

2 Overview of the Business Process Framework

An important business benefit provided by Oracle's Siebel Business Applications is the ability to effect the consistent and efficient execution of an organization's customer-centric business processes. Adherence to rapidly changing processes is critically important to both business agility and compliance with an increasing number of regulations.

The extensive business process automation capabilities of Siebel business applications—including Siebel Workflow, the State Model, and iHelp—are enhanced (in version 8.0) by the inclusion of two new modules: Task UI and Business Rules.

Siebel business rules allow you to maintain business process logic declaratively and in a location external to your Siebel applications. For information on implementing rules in Task UI, see the *Siebel Business Rules Administration Guide*.

An overview of the task-based user interface (Task UI) is provided in the following topics:

For information on the State Model and Siebel Worklow, see the *Siebel Business Process Framework: Workflow Guide*.

About Siebel Task UI

Task UI extends business process automation to the level of user interaction. Tasks are multiple-step, interactive operations that can include branching and decision logic. Task UI's wizard-like user interface guides the end user through task execution, allows navigation both forward and backward within task execution, and allows task execution to be paused and resumed as needed. This combination of features helps tasks to increase the efficiency of users by guiding them through the execution of unfamiliar tasks. Task UI can also increase the efficiency of busy veteran users, especially those working in environments that are prone to interruption, because it allows for easy switching between multiple tasks throughout the working day.

Figure 1 shows an example of a task view in a Task UI application.

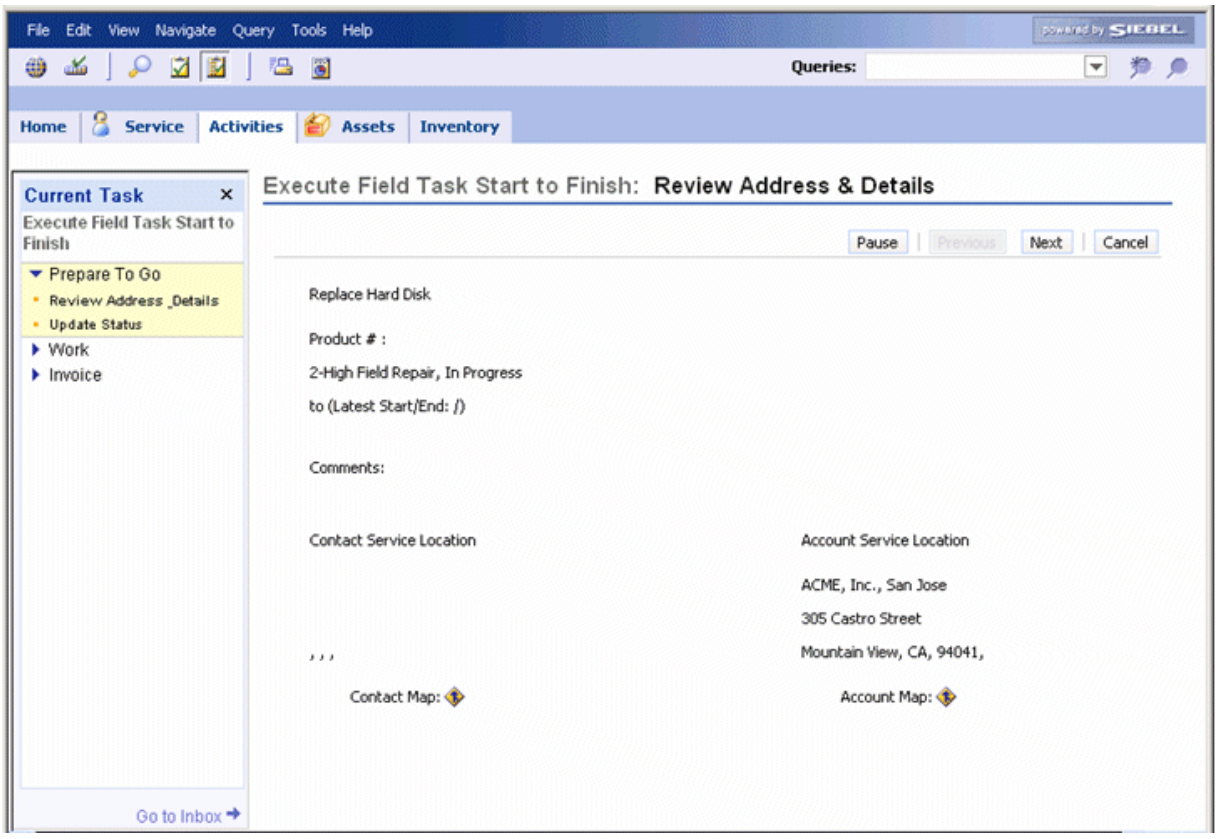


Figure 1. Example of a Task View in Task UI

A task comprises a distinct set of operations performed by a single end user, such as filling out an expense report. A task can also be incorporated as a step within one or more broader-based Siebel workflows. In this way, a task can be part of the defining of end-to-end business processes spanning multiple roles, such as the process for routing an expense report through multiple levels of review and approval. A task can also help define integration with external systems, such as for account setup and provisioning.

Task UI's features include:

- Directing forward and backward navigation through multiple pages.
- Providing extensive user guidance and supporting information to effect the accurate execution of complex processes, complemented by effective, on-point communication. For example, Task UI can guide the process of gathering all of the information and applying the logic to complete a financial needs analysis, the recommendations from which must then be sold back to the customer.

- Incorporating decision processes requiring complex business logic to determine the appropriate sequence of activities and content at each step. For example, Task UI can be configured to present appropriate upgrade or upsell products based on the customer's geography and current products.
- Applying sophisticated validation to enforce business rules in the execution of a business process, such as making sure that the customer provides a written statement of fact within 14 days of opening a credit card dispute (otherwise, the case will be closed automatically).
- Incorporating integration to external data or services into the processing of a task—such as invoking an external credit engine to determine an applicant's creditworthiness when taking an application, and then submitting identification information to the customer master database for validation.
- Coordinating multiple actions comprising a logical transaction that must either finish successfully or be completely rolled back, such as when executing a transfer of funds between financial accounts.
- Improving efficiency through next steps capabilities and tight integration with analytics.
- Allowing standardization of corporate business procecess.

About Designing Tasks

The Task Designer allows Task UI developers to configure tasks using a visual programming language, which consists of different types of steps connected to a directed, cyclic graph. Task Designer is tightly integrated with Siebel Workflow's Process Designer, so that it simplifies the process of configuring complex, long-lived business processes in Siebel business applications.

Figure 2 shows a task flow displayed in the Task Designer, located in Siebel Tools.

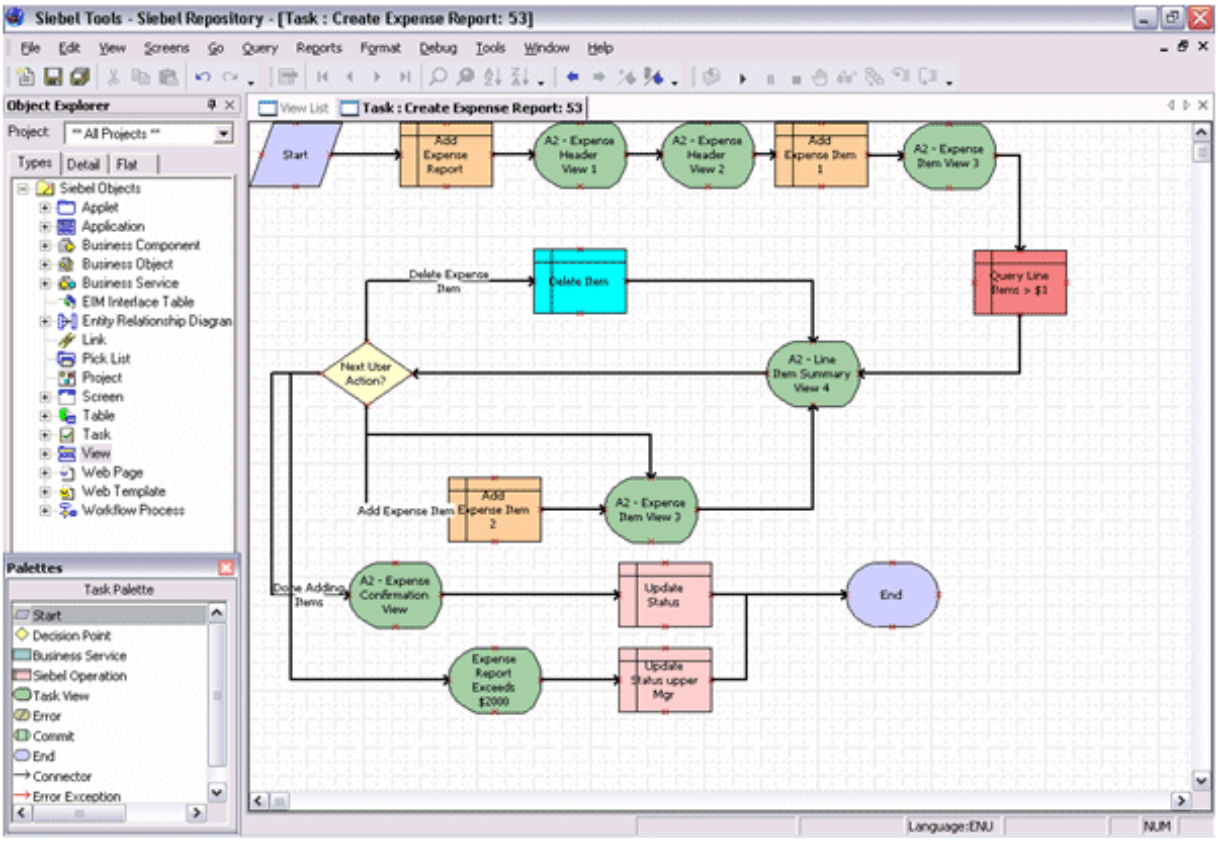


Figure 2. Task Designer in Siebel Tools

Comparing Task UI with Alternative UI Technologies

When deciding whether Task UI is the appropriate technology for your business needs, consider Table 1, which provides a summary of the advantages and constraints of the various UI options.

Table 1. Comparing Task UI with Alternative Technologies

Features	Ad-hoc UI (HI)	Ad-hoc UI (SI)	Task UI	iHelp	Smart Script	Custom UI (Web Channel)
Employee-facing	Yes	Not recom-mended	Yes	Yes	Yes	Not recom-mended
Customer-facing	No	Yes	No	No	Yes	Yes
Encapsulates business logic	Some	Some	All	None	All	All
Integration with Siebel Workflow	Some	Some	Full	None	Some	Limited
Integration with Universal Inbox	Some	None	Good	None	Best	Some
Support for long-running transaction	No	No	Yes	No	No	No
Performance and scalability overhead	None	None	Some	None	Large	Some

As shown in Table 1, Task UI is the only technology that supports long-running transactions, and the one that best integrates with business processes (Siebel Workflow). Task UI cannot be used for customer-facing applications, however, because it requires a high-interactivity (HI) client with support for ActiveX.

The power of encapsulating business logic and a guided user interface also comes with a higher price in terms of performance and scalability overhead, when compared with a well-designed ad-hoc UI for power users. As such, Task UI is typically the best technology to use for tasks that are employee-facing, nontrivial, transactional, and which may require tight integration with business processes.

Just like Task UI, other Siebel UI technologies also provide trade-offs between the important features listed in Table 1. For example, SmartScript provides the best integration with Universal Inbox, at the cost of significant performance overhead, as well as the cost of developing and maintaining a scripted solution. For this reason, SmartScript might be a more appropriate technology than Task UI if used to implement simple, nontransactional tasks that require strong integration with Universal Inbox and that are performed rarely enough so as not to jeopardize the scalability of the whole system (for example, a task for review of expense reports might be a good candidate for SmartScript).

On the other end of the spectrum, the ad-hoc UI is the best technology for tasks that are frequently performed, but simpler in nature and performed primarily by power users. iHelp complements the ad-hoc UI nicely for frequently performed and simpler tasks worked on by novice and intermittent users. Web Channel is the preferred technology for implementations that require a customer-facing UI with a specific look and behavior.

Is Siebel Task UI Right for Your Implementation?

Siebel Task UI can provide a quick return on investment when applied to the appropriate business-use cases. Task UI is not a universal answer to every possible problem that your business faces. For this reason, be selective about when to implement it, considering all the trade-offs it entails, as well as the benefits it provides.

3 Task UI Concepts

This chapter discusses the concepts and terminology related to the task-based user interface (Task UI). It includes the following topics:

Language Concepts for Task UI

This topic describes the following terms and concepts employed in a Task UI implementation:

- “Transient Business Component” on page 30
- “UI Task” on page 33
- “Workflow Task Step” on page 33

Business Task

A logical unit of work to be accomplished. This is the business user’s perception of a task (for example, filing an expense report or entering a new prospect in the system), rather than the programmatic or user interface perception.

See also:

- “Task” on page 21
- “UI Task” on page 33

Error Handling

You can implement error handling to gracefully intercept and handle processing errors that occur during the execution of a task. The following topics describe how errors are handled in Task UI:

- “Handled Errors” on page 18
- “Unhandled Errors during Task Navigation” on page 18
- “Errors during Task Cancellation” on page 19
- “Errors during Task Pause” on page 19

Handled Errors

Explicit error handling is implemented using a combination of exception branches and error steps. Exception branches may be terminated with an error step. The semantics of the Error step are similar to the default exception handling. When the Error step is executed, the existing error message is replaced with the error message configured for the Error step and returned to the user at the current task view. In this way, the user is presented with a more informative and contextual error message.

Unhandled Errors during Task Navigation

During task execution, if an exception occurs in a task step where no exception branch is defined, a generic error message is displayed to the user in a message box on top of the current task view. After acknowledging the error message, the user may correct the data on the task view or may navigate to a previous task view to correct the data. This is also known as default exception handling.

Errors during Task Cancellation

When the end user clicks Cancel, a dialog box prompts for confirmation, for example:

> You are about to delete or undo changes to this task. Click OK to continue.

Errors during Task Pause

There is no exception handling when a task is being paused. If pausing a task fails, the end user receives a standard error message, not a special pause error. The end user can cancel the task, or continue with the task and submit it.

Event Handling

Task event handlers allow you to configure the processing that occurs when Pause, Resume, Pre-Cancel, or Delete events are triggered by the end user. You can add event handlers to your task using the Task Designer in Siebel Tools. To configure a task flow for event handling, you must create a Task Event object under the parent Task object, and specify a business service or a service workflow to call when the corresponding task event occurs.

Task events can be triggered by the following end-user operations:

- **Resume.** Reinstantiates a task instance by restoring the task's state and history. This event resumes a paused task when the owner clicks an inbox item of the type Task in the inbox view.
- **Pause.** Causes the task state and the history stack to be serialized and stored on the disk. An inbox item, which may be used to resume the task, is created in the inbox on the disk.
- **Cancel.** Either deletes the task or rolls it back to its prior state. If the task has a parent flow, the parent flow receives an Abort status from the task.
- **Delete.** Removes the task from the inbox, and removes the task instance from the database. If the task is initiated by its parent workflow process, the parent flow receives an Abort status from the task. This event is created by the task utility service.

The event handlers are executed before the Resume, Pause, Cancel, or Delete end-user operation is carried out. Therefore, the event handlers are considered pre-event handlers. If the event handler execution succeeds, the task controller proceeds to complete the end-user operation. If an error occurs during event handler execution, the end-user operation is halted. Errors that occur during event handler execution are returned to the end-user. For information on how to configure event handlers, see "Configuring Task Event Handlers" on page 84.

Long-Running Workflows

The Task UI framework allows for integration with long-running workflows. Long-running workflows are used to create assigned tasks. Typically, this occurs upon a task completing and waking up a workflow that creates an inbox record for the next user who needs to process some information.

For information on configuring long-running workflows to invoke tasks, see *Siebel Business Process Framework: Workflow Guide*. For a description of how to configure the initiation of a long-running workflow within the Task UI framework, see "Initiating a Long-Running Workflow" on page 88.

Subtasks

Using subtasks, you can make a task flow modular. If your task flow diagrams become so large that readability is compromised, use subtasks to break up the work. The primary reasons to use subtasks are the following:

- **Task readability.** Improve the use of space on the canvas by breaking large tasks into separate modules.
- **Reuse.** Decrease development and maintenance costs by reusing common Task UI sequences.
- **Consistency of design.** Maintain a clean, consistent, and intuitive programming model.

While subtasks correspond to tasks in the same way that subprocess steps correspond to workflows, the subtask programming model varies from that of workflow subprocesses. The major differences are the following:

- A subtask and its parent task share the same process instance. In Siebel Workflow, a main process and its subprocess run in separate instances.
- A subtask and its parent task should have the same business object type, and share the same business object instance. In Workflow, the main process and the subprocess can be of two different business object types, and can have two different business object instances.
- Both subtasks and workflow subprocesses support the notation of local process properties. Each workflow subprocess invocation creates a process instance and has its own process properties. Each subtask invocation does not create a new instance, but rather a new context is created for each invocation for storing local task properties. A subtask and its parent task do not share task properties.
- A subtask's boundaries are not visible to end users. This means that if end users click Previous this action might cross the subtask boundaries in either direction.

Other characteristics of subtasks that require special attention include the following:

- Event handlers can be defined only for the main task, and not the subtasks. If an event is triggered by a subtask, it is propagated back to the main task, and handled by the event handlers defined in the main task. The event handler runs under the context of the main task.
- A task must be explicitly defined as the main task or subtask at design time:
  - Each main task has a task state. The task state stores information essential to the task run-time instance. This information includes the current step pointer, task object Id, and the navigation path. Subtasks do not have individual task states, but they do have their own individual local task properties.
  - A subtask cannot be used as a main task.
  - The parent task and its subtask communicate through input and output argument passing.
- Subtasks can be nested (contain more subtasks).

For information on how to configure subtask steps when building task flows, see "Configuring Subtask Steps" on page 71.

Transient Business Component in a Subtask

A transient business component (TBC) can have only one record for each context. In Task UI, TBC context is at the level of the subtask, which means that even if a subtask uses the same TBC with its parent task, the two do not share the same TBC record. If there is no record for a TBC in the current context, a new record is created by the framework when the TBC is queried.

Task

A *task* is a logical unit of work performed by an end user to complete a business operation. This logical unit of work might be completed using different mechanisms, one of which is Task UI.

In the context of the Task UI framework, this work is performed through a series of user interface views that guide the user towards completion of the task.

Throughout this guide and in other related documentation, the term task can refer either to a task definition or task instance.

See also:

- "Task Definition" on page 22
- "Task Instance" on page 23

Task Branch

A *task branch* is a child object of a task that facilitates the flow of execution for a task to go from one step to another. There are different types of task branches, just as there are different types of branches in workflow processes. The two types of standard task branches are conditional and nonconditional (default branch). They are used to direct the execution of the flow between task steps. Conditional task branches are used in conjunction with decision steps, which determine the appropriate path to take in the flow, based on some criteria. Exception task branches are used to direct the flow of execution in case of errors. Error steps are used to customize error messages returned to the user.

See also:

- "About Decision Points" on page 26
- "About Error Steps" on page 27

Task Chapter

A *task chapter* allows you to logically group particular steps in a task and show them in the task pane through a series of bold headers that inform the user of what lies ahead in the processing of this task. For more information about the task chapter user interface element, see "Task Chapter" on page 40.

Task Definition

The *task definition* is the metadata component of a UI task. This definition includes the task flow and the associated attributes of the flow (for example, view names). In a broader sense, the task definition refers to the many pieces required to run a task instance, including the definitions of referenced task views, applets, business components, LOVs, and other related metadata.

See also:

- "Task Flow" on page 22
- "Task Instance" on page 23
- "Task View" on page 46
- "Transient Business Component" on page 30
- "UI Task" on page 33

Task Event

When a task is paused, resumed, canceled, or deleted, events are fired to handle these operations. The event handler for each operation performs the required actions according to the semantics of each operation.

NOTE: Event Handlers cannot be associated with subtasks. If a task event is triggered when the user is in a subtask, the corresponding event handler associated with the main task is executed. The event handler is executed in the context of the main task.

See also:

- "Event Handling" on page 19

Task Flow

A *task flow* is the portion of a task definition that shows the flow of the task; that is, the diagram of the steps in a task. A task flow is similar to a workflow, as are their respective design tools. However, the task flow and workflow are used to perform different kinds of processes.

A task flow involves at least one task view step, in which a user enters data into fields and controls. A task flow is meant to guide the user experience to accomplish a business function. A task flow is synchronous and is tightly bound to a UI experience. The interactions in a task are the result of a user explicitly navigating through the flow using the actions provided by the navigation buttons (Next, Previous, Pause, Cancel).

In contrast, a workflow is a business process that encapsulates a series of tasks. A workflow does not need to be synchronous and does not need to involve the UI. A workflow can wake up and go to sleep in response to calls from other systems.

See also:

- "Task Definition" on page 22

- “Task Step” on page 25

Task Group

The *task group* is a grouping mechanism that controls the UI tasks displayed in the Context pane. The task group also allows you to specify where the task must take the record context when the task is invoked, or whether the task can be invoked standalone.

Within a task group, you can specify the application to which the task is available. When a task group is declared, the task group can be individually added to be available in a view-by-view basis, or it can be added to the Task pane view to be available globally throughout the application. The task group also allows you to control the order of the tasks.

See also:

- “Context Pane” on page 37
- “UI Task” on page 33

Task Instance

Task instance refers to a single instance of a particular task definition. A new task instance is created every time a particular kind of task is started, either by the end user, or by a long-running business process. The relationship between a task instance and a task definition is similar to the relationship between an object and a class in an object-oriented programming language.

Each task instance owns its own state, consisting of task properties and navigation history. Data in the associated business object is generally shared, except for the data in transient business components, which are unique to a specific task instance.

The life span of a task instance can often coincide with the life span of the associated task transaction, but that is not necessarily the case. The task instance might not even use a task transaction, or it might use a series of task transactions (by way of intermediate commits).

Tasks can be launched in ways other than a call through the task pane. A task can be started in the following ways:

- **From an ad-hoc view.** See “Configuring the Ad-Hoc UI for Task Instantiation” on page 102.

  You can do this using:

  - The Context pane. See “Context Pane” on page 37.
  - A push button. See “Configuring Buttons for Task Instantiation” on page 104.
  - Application and applet menu items. See “Configuring Menus for Task Instantiation” on page 104.
  - A link.
- **From iHelp.** See “Configuring iHelp Links for Task Instantiation” on page 104.

- **From a long-running workflow.** See the topic on configuring long-running workflows to invoke tasks in *Siebel Business Process Framework: Workflow Guide*.
- **From a script.** See "Writing Scripts to Instantiate Tasks" on page 109.

NOTE: When you are configuring task initiation, it is important to consider access control. For an end user to launch a task, that user must have the appropriate responsibility associated with the task.

See also:

- "UI Task" on page 33
- "Task Session" on page 34

Task Metrics

Task metrics collect and store various task data that is regularly loaded into a data warehouse and analyzed using an OLAP tool like Oracle Business Intelligence. Two supported types of metrics are: timestamp metrics and property metrics.

Timestamps

One of the centerpieces of information in process measurement is the time when a certain event occurs. For example, to know the amount of time a user spends on a task, the time the task starts and ends is needed.

If task metrics are enabled at task deployment, the Task UI framework stores timestamps for the following events:

- Task Started
- Task Paused
- Task Resumed
- Task Cancelled
- Task Completed

The collection and persistence of these timestamps incur a minimal, but still finite performance and scalability overhead.

NOTE: Step-level timestamps are not collected, because the impact on performance is unacceptable.

Property Metrics

Property metrics allow you to collect task data whose sources are task properties. You can configure property metrics to satisfy the requirements for measuring business performance.

For more information, see "Configuring Task Metrics" on page 79.

Task Properties

Task properties are data objects used to store data that is local to a task or a subtask, which is analogous to local variables in a programming language. The task properties define the characteristics of the task. Task properties can also be used to pass data into and out of the task, analogous to input and output arguments in a programming language. For more information, see "System Task Properties" on page 77.

Task Step

A *task step* is a child object of a task. Each step represents a particular action performed as part of a task. Each task contains several steps. Similar to a workflow process, a task can contain the following types of steps:

- Business Service step
- Decision Point step
- End step
- Siebel Operation step
- Start step

A task can also contain steps that are unique to the Task UI application, and that are not part of workflow processes, as follows:

- Commit step
- Error step
- Subtask step
- Task View step

As in the Process Designer, the Task Designer provides connectors (branches) and error exceptions. As with designing workflow processes in the Process Designer, in the Task Designer you define conditions and values associated with steps.

An average task flow might have ten to fifteen steps, although the tasks you implement may vary greatly in length.

For general information on how to configure task steps, see "Editing Task Flows" on page 63.

Task Step Descriptions

The Task UI framework provides the following types of task steps:

- **Business service step.** A step that calls a business service. See "About Business Service Steps" on page 26.
- **Commit step.** A step that explicitly commits the temporary task data to the Siebel database. See "About Commit Steps" on page 26.

- **Decision point.** A branch step that determines the direction through the task based on input. See "About Decision Points" on page 26.
- **End step.** Completes the task and commits the temporary task data. See "About End Steps" on page 26.
- **Error step.** A step that presents a custom error message to the end-user. This step provides the same function as the Stop step used in the Siebel Workflow application. See "About Error Steps" on page 27. (For more information about Siebel Workflow, see the *Siebel Business Process Framework: Workflow Guide*.)
- **Siebel Operation step.** A step that performs a Siebel Operation. See "About Siebel Operation Steps" on page 27.
- **Start step.** The step that begins a task. See "About Start Steps" on page 27.
- **Subtask step.** A step that calls another task. See "About Subtask Steps" on page 27.

  NOTE: For information on subtasks, see "Subtasks" on page 20.
- **Task View step.** A step in which the user interface interacts with the end user. See "About Task View Steps" on page 27.

About Business Service Steps

A *business service step* allows you to call a business service, which executes predefined or custom actions in a task flow. See "Best Practices for Invoking Business Services" on page 144 for more information.

About Commit Steps

A *commit step* is a step that explicitly commits the task data stored in temporary storage to the Siebel database. You can use commit steps to transfer all data stored in the temporary storage to the Siebel database while you are in the task, before the task is completed. When the task is completed, the End step commits the temporary data to the Siebel database. For more information on the conflict detection and resolution scheme during the commit, see "Conflict Detection and Resolution" on page 29. For information on how to configure commit steps, see "Configuring Commit Steps" on page 67.

About Decision Points

A *decision point* is a step that evaluates conditions on outgoing condition branches to determine the next step to be executed. For more information, see "About Branching" on page 73.

About End Steps

An *end step* instructs the Task UI run-time framework to end the task instance, and commit all temporary data to the Siebel database. Each task flow must contain only one end step.

For more information on the conflict detection and resolution scheme during the commit, see "Conflict Detection and Resolution" on page 29. For information on how to configure end steps, see "Configuring Start Steps and End Steps" on page 68.

About Error Steps

The *error step* provides a mechanism for the taskflow developer to instruct the Task UI engine to display a localized error message to the end user. This functionality is typically needed when an error message returned from a business service might not be clear enough for the end user. The error message is displayed in a modal pop-up window and the last-displayed task view becomes the active step. Execution of an Error step before the first View step in a task is successfully displayed causes cancellation of the task.

An error step can be used in an exception branch, or as part of the normal task flow processing logic to handle expected errors. For instructions on how to work with Error steps, see "Configuring Error Steps" on page 82. For more information on error handling, see "Error Handling" on page 18.

About Siebel Operation Steps

A *Siebel Operation step* performs operations such as Insert, Update, and Delete on a business component in a task's instance of a business object. See "Configuring Siebel Operation Steps" on page 68 for more information.

About Start Steps

A *start step* is the initial step in a task. For more information, see "Configuring Start Steps and End Steps" on page 68.

About Subtask Steps

A *subtask step* allows you to invoke a separate task within a task. In contrast, the task invoked by a subtask step is a task in which the Is Subtask property is set to TRUE. Generally, you create the subtask first before adding the corresponding subtask step to the main task flow. A task definition can have one or more subtask steps.

Similarly to the Workflow subprocess step, you can pass information into and out of a subtask through the input and output arguments. Input arguments allow you to populate task properties in the subtask with information from the parent task. For instructions on how to work with subtask steps, see "Configuring Subtask Steps" on page 71. For information about subtasks, see "Subtasks" on page 20.

About Task View Steps

A *task view step* presents a user interface view to the end user. It allows the end user to view the application data that the task is working on, and provide user input when necessary. It also allows the end user to control task execution and navigation by providing a task playbar applet with navigation buttons (for example, Next and Previous). For more information, see "Task View Step" on page 47 and "Configuring Task View Steps" on page 65.

Task Transaction

Task transaction allows transactions to last for arbitrarily long periods of time while maintaining good reliability, performance, and scalability characteristics. Task transaction can be short-lived, or it can last for days or even months. It might even span multiple user sessions, process boundaries, and database or application server restarts.

Task transaction is dependent on a mechanism implemented in the Siebel application server, external to the underlying relational database management system (RDBMS).

NOTE: The task transaction mechanism can be turned off at both the business component level and the task level.

The following topics describe the characteristics of task transaction:

Atomicity

Operations performed in the task are either committed together, or aborted. Atomicity is traditionally achieved using the transactional mechanism embedded in the RDBMS. However, that mechanism is not appropriate for task transactions because it is limited to a single database connection, which then either limits the transaction's duration, or jeopardizes the system's reliability, performance, and scalability.

Isolation

A change made within a task transaction can be seen only within that task transaction until it is committed (for example, with an insert, update, or delete operation). On the other hand, a change committed outside of the task transaction is also seen within the task transaction as soon as the changed record is queried again after the external change is committed.

Transparent Storage

A set of generic tables is dedicated to storing data changes within the task transaction. The Task UI framework maps columns in the generic tables to columns in the Siebel database tables, completely hiding the complexity of the storage details from the Task UI developer.

However, the need to clean up task transaction storage after transactions are committed and aborted is not transparent to Siebel administrator. The Siebel administrator must make sure that the Task UI server component group and its component Task Log Cleanup are enabled. When enabled, this component automatically cleans up the task transaction storage as a background process.

Transparent Data Retrieval

The Task UI framework merges the data in task transaction storage with the data in Siebel database tables, with full support for data filtering (search specification). Declarative data ordering (sort specification) is fully supported only for hierarchical business components; for others, ordering functions as expected only when no data is changed within the task transaction.

Conflict Detection and Resolution

Three major types of conflicts can occur within a task transaction:

- **Duplicate conflict.** Is detected by unique key violation in the RDBMS during the commit phase.

  The business component's Dup Conflict In Task user property allows you to specify the desired duplicate conflict resolution behavior by setting one of three values:

  - **Resolve**. The duplicate record is written with its Conflict Id field being set to the value of Id.
  - **Fail**. The task transaction is aborted.
  - **Ignore new record**. The duplicate record is skipped, but the rest of the task transaction is committed.
- **Update conflict.** Occurs when the same record is modified both within and outside the task transaction. Similar to ad-hoc UI, the update conflict is detected based on the Modification Id system field.
- **Delete conflict.** Is caused by either of the following two reasons:
  - A record is deleted within and updated outside the task transaction
  - A record is updated within and deleted outside the task transaction

Resolution Behavior in Update and Delete Conflicts

Update and delete conflicts are detected either upon data retrieval within the task transaction, or at the commit phase. You can configure their resolution behavior by setting the On Conflict task property to one of the following values:

- **Continue operation**. Changes made within the task transaction overwrite changes committed outside the task transaction. This does not apply to delete conflict where the record is deleted outside of the task transaction, which always returns an error message.
- **Cancel operation**. An error message pops up to indicate the conflict. Data in conflicting fields within the task transaction are lost, and the end user must manually resolve field-level conflicts by reentering them.

Task UI

Task UI is a method of presenting the user interface of your Siebel business application using a wizard-like user interface that guides the end user through the execution of a series of tasks, allowing bidirectional navigation, branching, and the ability to pause and resume task execution as needed.

See also:

- “User Experience Concepts for Task UI” on page 33
- “About Siebel Task UI” on page 11

Task UI Framework

The *Task UI framework* consists of the development, run-time, and administrative features that allow you to implement Task UI. See also:

- “Task UI” on page 29
- “About Siebel Task UI” on page 11

Temporary Storage

During task execution, transient data is stored in a temporary storage and not committed to the Siebel database unless a commit step is used. When the task is cancelled, the transient data related to the task is removed from temporary storage. When the task is submitted, the transient data is committed to the Siebel database. See also “Task Transaction” on page 28.

Transient Business Component

Transient business components (TBCs) provide a way to create task-specific data that can be displayed and edited in the user interface and accessed from within the task-flow logic.

TBCs are used to house data to control the flow or logic (but not stored in the database) and are cleared when a task is completed. They are created similarly to other business components in the repository except that they are based on the S_TU_LOG table and are of the type transient.

TBCs are used for data that is needed for the lifetime of the task, but can be discarded when the task is done. This might include intermediate calculations, end-user decisions made in the user interface, or other data needed during the task that need not be transferred to long-term storage in the database.

TBCs are also used for data that may be later incorporated into longer-term storage, but the initial views of which do not capture enough information to build a complete record. For example, the initial view of a task might ask for a customer's postal code. The postal code is used to make task-flow decisions and to query data. The postal code may also get included in transactional records generated by the task. Using a standard BC to enter the postal code would require entry of a complete record all at once. Using a TBC allows the postal code to be entered alone and written to a standard BC later in the flow.

About Transient Data

Transient data is data that is relevant within a limited time period. In Task UI, that time period is the duration of a task instance. A typical example of transient data is an end user's answers to questions such as "What would you like to do next?"

In Task UI, data that is transient — dynamic data — is tied to a special type of business component called a transient business component (TBC). The TBC is a business component for which the records span the lifetime of a task instance. The records of the TBC can be accessed only within the context of a particular task or subtask. A user property in the top-level Task object, Transient BC - Property, specifies the TBC used in the task to store temporary data that is only necessary throughout the life of the task.

The TBC is used to make sure that only one record exists for each context. Context may be the main task or a subtask. There can be only one TBC with only one record for each context.

NOTE: Task properties also satisfy the definition of transient data above. The main difference between TBC data and task properties is that while TBC data can be directly presented in the UI, task properties cannot. This difference should be a primary factor when deciding whether to store transient data in a TBC or in a task property.

How Transient BCs Differ from Standard BCs

You distinguish between transient business components and standard (nontransient) business components using the Type property. A TBC's type is Transient, while the type of a standard business component is nontransient.

NOTE: When a business component's type has been set to Transient or Nontransient, it cannot be changed, even through a copy operation.

A TBC uses either the CSSBCTaskTransient class, or a class derived from CSSBCTaskTransient. The CSSBCTaskTransient class enforces that there is only one row for each context for each TBC. In Task UI, TBC context is at the level of subtask, which means that even if a subtask uses the same TBC with its parent task, the two do not share the same TBC record.

This specialized class also filters the single record for the current context and current business component. It executes a default query on the first Get/Set function, and creates a new record if no such record exists (Execute () returns no rows).

Further characteristics of TBCs are the following:

- A transient business component is always based on the S_TU_LOG table.
- Changes to transient business components are immediately committed.
- Transient business components do not allow multi-value fields.
- The Column and Join properties of a TBC are auto-populated and are not editable:
  - Explicit joins are not allowed within a TBC.
  - All columns are forced active at run time, to avoid field activation problems.
- LOV-bound data is stored using language-independent code, if the associated LOV type is marked Multilingual.

Managing Transient Data

Transient business components are deployed just like standard business components, through the repository (SRF).

NOTE: Unlike tasks, TBCs are not deployed in the run-time deployment schema.

TBCs allow you to display transient data. TBC data can be manipulated both by the end user (through UI controls), as well as by business logic embedded in the task definition (through the business component update operation).

NOTE: Insert, delete, and copy operations on TBCs are not supported.

TBC data can be referenced directly from a task, just as standard business component data can. For example, it can directly drive branching decisions, or it can be assigned to a step's input method arguments. If necessary, TBC data can even be assigned to task properties through a step's output argument assignments.

TBC data is stored in the S_TU_LOG table and, if necessary, in S_TU_LOG_X_* extension tables. The TASK_ID_VAL column stores the Task Instance ID, while the column BC_NAME stores the business component name. After use, the table is wiped clean by a background service, based on TASK_ID_VAL. If Task UI is running on a mobile client connected to a local database, TBC data is cleaned up as soon as the task finishes.

The primary uses of transient data include the following:

- Flow control through radio buttons (in the UI) and decision steps (in the task definition). The selection of the end user determines where the Task UI moves next.
- Control over search specifications. User input based on choices that include a pre-default, such as in response to the question, “In which month did your transaction take place?”
- Data manipulation:
  - Merging fields from different business components in a single applet, for example, to allow creating a new account and entry of contact data through a single task applet.
  - Splitting the entry of required fields across views. The record is not inserted into the business component until all field values are collected.
  - Polymorphic user interface. Based on a selection at run-time, UI controls map to different business component fields. For example, if the user chooses Credit Card as a payment method, then fields for credit card type, number, and expiration date are enabled at run-time.
  - Checking for the existence of a record before committing a new record that might be a duplicate.

Getting Data Out of a TBC

As with standard business components, TBC data can be assigned to task properties and also mapped to other business component fields.

Updating Data in a TBC

Transferring data between business components happens in the same way, regardless of whether one is a TBC. You can transfer data between business components of either type, transient or nontransient.

To transfer data from a transient business component to a task property

1. While positioned on a step after which the assignment should happen (typically, this might be a task view step that allows the end user to manipulate TBC data):
   a. Assign a TBC field to a task property in the Output Arguments tab of the Multi Value Property Window.
   b. For the assignment type, use Business Component.

To transfer data between a transient business component and another business component

1. Use the Siebel operation of type Update to update the target business component.
2. In the Fields tab of the Multi Value Property Window, specify the Business Component type.
3. Pick BC and BC Field properties to select the source.

UI Task

A specific task built on top of the Task UI framework (see also "Task UI Framework" on page 30).

Workflow Task Step

A task creation and assignment step in Workflow (see also "Task Step" on page 25).

User Experience Concepts for Task UI

This topic provides a description of key user experience concepts employed in a Task UI application. It contains the following topics:

See Chapter 2, "Overview of the Business Process Framework" for more information about Task UI applications, and "UI Elements Used in Task UI" on page 35 for more information about the user interface features of Task UI.

Task

In Task UI, the term *task* can refer to two distinct concepts, as follows:

- Task is a logical unit of work performed by an end user to complete a business operation. This logical unit of work might be completed by multiple mechanisms, one of which is Task UI.
- Task is a configuration built using the Task UI framework. In this usage, a task is a specific implementation, rather than a logical concept.

From the perspective of the end user, a task is the view representation of a logical unit of work to be performed. The task is presented in the application as a line that can be clicked in the Task pane and the corresponding view (or series of views) where the user performs this unit of work.

See also:

- "Task" on page 21
- "Task Definition" on page 22
- "Task Session" on page 34

Task Session

A *task session* is the mechanism through which an end user experiences a specific task instance. A task session begins when the end user launches or resumes a task and ends when the task is paused, completed, or canceled. This implies that a task instance may span zero, one, or more task sessions.

Universal Inbox

Universal inbox is a general framework for presenting end users with units of work (called inbox items) assigned to them for execution. It allows assigning only a single owner to each inbox item. Tasks can be transferred between users for reassignment, approval, or consultation through the universal inbox. Tasks are stored and accessed from the owner's inbox. The role of the inbox for Task UI is that is gives a single location for users to find and retrieve their tasks that have been paused or tasks that have been assigned to them in a long running process.

The Task UI framework is dependent on universal inbox to allow end users to start, resume, transfer, or delete tasks that were assigned or transferred to them. To make navigation easier, the bottom of Task Pane contains a hyperlink to the universal inbox.

Each universal inbox item of type Task references only one task instance. Throughout its life span, the task instance and its associated inbox item change through multiple states, indicated by the status field of the inbox item. Since the name of the inbox item is the name of the task definition, the additional inbox item context field allows distinguishing between multiple instances of the same task.

A user can resume a paused task in the inbox by clicking its hyperlinked name field (similar to starting a newly created task). The state of the task instance being resumed is recreated, and the view that was being displayed when the task was paused is shown. Note that if the data shown was changed and committed outside of the task, those changes show up when the task is resumed. For more information on this topic, see "Task Transaction" on page 28.

While generic universal inbox views show all inbox items assigned to current user, inbox items of paused tasks can be also be seen through their associations with particular business data such as accounts, service requests, or contacts. This allows end users to resume other user's tasks without jeopardizing integrity of their inbox. For more details, see "Associating Tasks with a Business Object Instance" on page 87.

UI Elements Used in Task UI

This topic describes the following user-interface elements within the Task UI application:

Action Pane

The *Action pane* is the pane at the left-most part of the task view. It is shared by the three distinct frameworks: Task UI, iHelp and Search Center. When the Action pane is displayed, the toggle button for the framework that controls its appearance is displayed as inset. The Action pane can be closed (useful when screen space is critical) either by clicking the toggle button, or by clicking the close (x) icon in the upper right corner of the Action pane.

Applet Message

An *applet message* is a free-flowing text control which presents a mix of hard-coded, static text strings and dynamic data from the Siebel database, filled in at run time. An applet message is presented as a continuous, wrapped line of text. Applet messages are similar to the personalization text which appears on home pages. You can use applet messages to provide instructions to the end user performing the task. For example, in the Siebel Call Center application, an applet message may represent a block of text to be read by a customer service representative (the end user), when addressing a customer.

You specify the static text at design time. The dynamic data is derived from business component fields and, at run time, it becomes part of the applet message the end user sees while moving through the task.

NOTE: The format of the text in the applet message is not editable.

An applet message, as a user interface control, is statically positioned within the task applet. It differs from other control types (such as Text or Field) by always displaying its content as read-only, and not inside a box or border in the user interface.

Figure 3 shows an example of an applet message. In this example, the values for status, account, and the name and number of the agreement are dynamic.

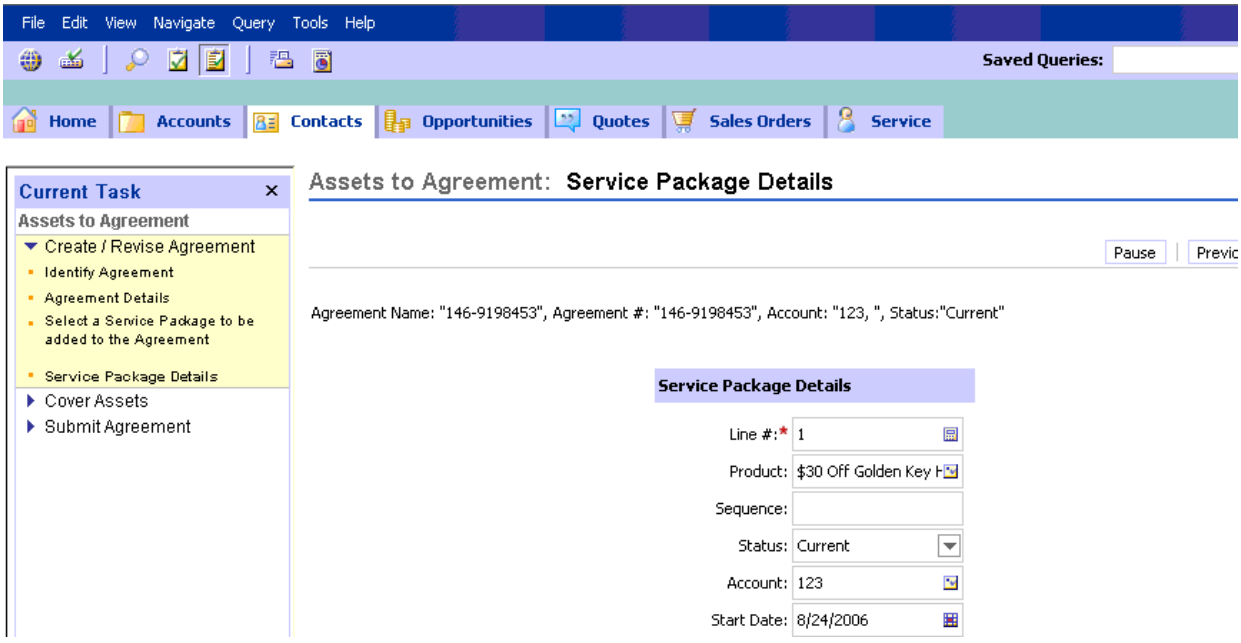


Figure 3. Sample Applet Message

For more information on how to configure an applet message, see "Configuring Applet Messages" on page 100.

Context Pane

The *Context pane* is displayed in the task pane when the traditional Siebel (ad-hoc) UI is in use. The Context pane shows the tasks that can be launched within the current context. You can use task groups to configure the content of the Context pane.

As shown in Figure 4, the header of the Context pane is labeled Tasks, while the body displays appropriate task groups and the associated tasks. As recommended in Chapter 9, "Best Practices for Working with Tasks," the task group containing the most frequently used tasks is displayed first, followed by the task group containing tasks that are related to the current content presented in the ad-hoc view.

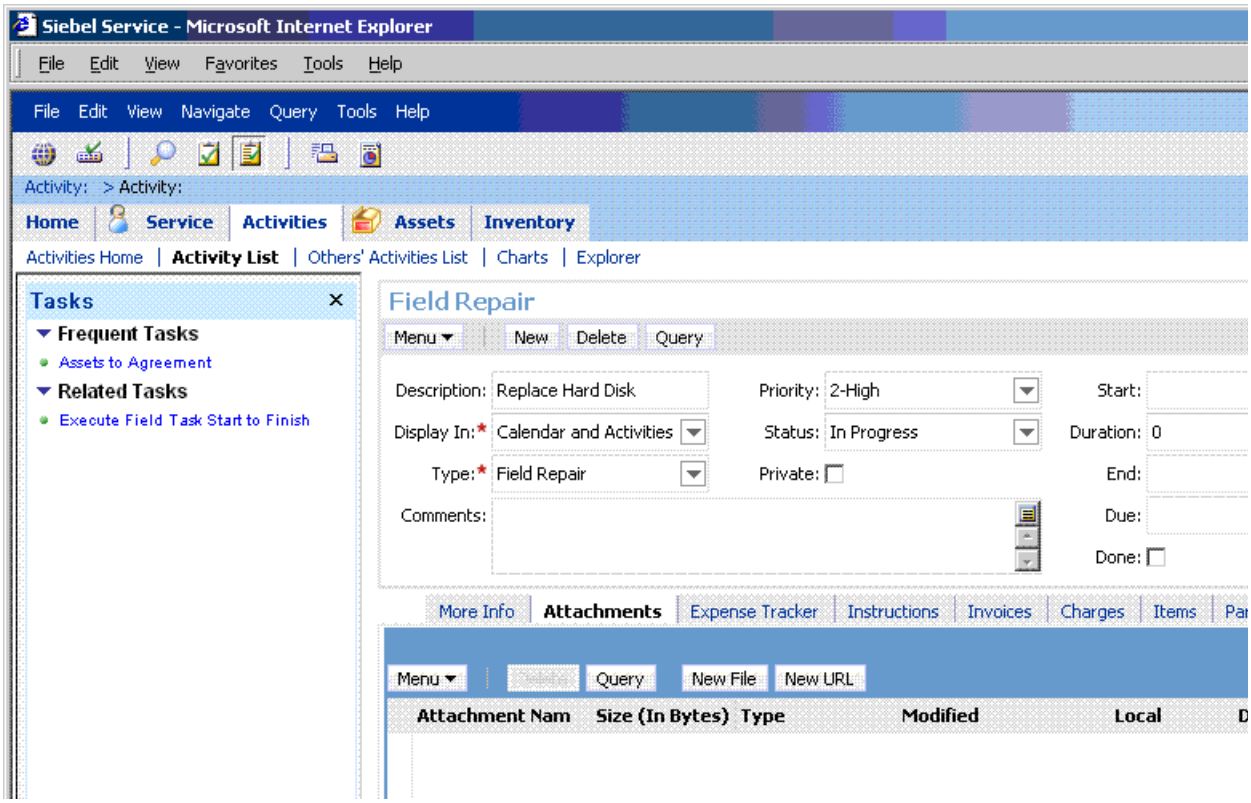


Figure 4. Context Pane Example

In this example, executing a field activity is a task related to a field repair activity shown. The end user can manage Context pane screen area by collapsing and expanding task groups.

When end user clicks on a task name in the Context pane, new instance of that task is started and the first view in that task is displayed. At that time, Task pane turns into Current Task pane.

See also:

- "Current Task Pane" on page 38
- "Task Group" on page 23

■ “Task Pane” on page 43

Current Task Pane

The *Current Task pane* is displayed in task pane when a task is running (as part of a task view). It is a read-only pane that helps the end user navigate through the task by showing all chapters, with the current chapter highlighted. Within the current chapter, the Current Task pane displays visited view steps in the order of visitation, with current view step displayed in bold.

If no chapters are defined in the task, the Current Task pane displays only view steps. Upon backward navigation, view steps that follow the current step on forward navigation are still displayed. If, on subsequent forward navigation, the end user enters a different flow branch, the view steps after the current steps are cleared from the Current Task pane.

See also:

■ “Task Chapter” on page 21

■ “Task Pane” on page 43

■ “UI Task” on page 33

■ “Task View Step” on page 47

Persistent Dashboard

The dashboard, a standard Siebel user interface component, is available in both the ad-hoc UI and the Task UI. The dashboard displays global information, such as contact information of the caller (for Siebel Call Center). This information stays on the screen as the end user switches between different ad-hoc views.

This concept of a persistent dashboard plays a part in a Task UI implementation. The dashboard shown to the end user in a Task UI implementation can include screen updates to data modified by the running task.

Radio Button Group

Along with combo boxes and drop-down lists, radio buttons are controls used by end users to make choices that drive the Task UI application. You configure radio buttons on task applets to register these user choices and decisions. The radio buttons you configure present a predefined list of mutually exclusive options, one of which the user selects.

The input received by a radio button may determine the next step of a task, or it may determine which data to retrieve for display in the next view of the task. For example, a user who is a customer service representative may face multiple options in response to a question he poses to a caller regarding the nature of the call. The CSR asks the customer, “How can I help you today?” (This question is shown in the UI as a prompt to the CSR, as in Figure 5.) If the response the CSR registers on behalf of the caller is that the call is a balance inquiry, then the next view Task UI presents to the CSR will be different than the view presented if the caller’s issue is a credit card dispute.

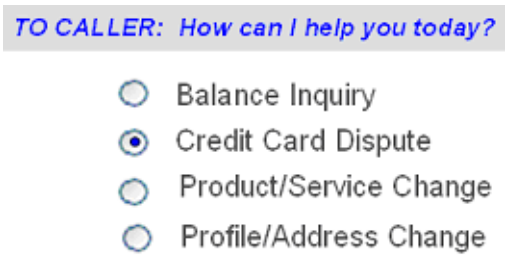


Figure 5. Example of a RadioButton Group

Task applets contain radio buttons that are populated by an underlying business component field. A set of radio button controls is not a group of separate items on the applet Web template; the multiple choices that make up a radio button comprise a single control item.

You can map radio buttons to both transient and nontransient business component fields, as long as the fields are single-value fields.

A primary difference between a radio button group and a bounded picklist is how it appears in the application. With a radio button, the user can see all choices at once, without clicking on the picklist drop-down button. When the user selects one choice, all the others are unselected.

Configuration of radio buttons requires a list of values (LOV) from which to draw the choices presented to the user. One value is the default.

The radio button control should generally not be based on hierarchical LOV. In the exceptional case where a hierarchical LOV is required as the base for radio button control, it will work at run-time, but will not correctly display values during preview in Siebel Tools.

NOTE: Radio buttons cannot appear in list applets. If you attempt to configure a radio button control within a list applet, the buttons are displayed as a text entry field.

For more information, see “Configuring Radio Button Groups” on page 99.

Standard Applet

Standard applets are traditional ad-hoc applets, as opposed to task applets which are specific to tasks and are associated with transient business components. The applets in a task view can be either standard applets or task applets.

Task Applet

A *task applet* differs from standard applet in that it is designed to interact with transient data (data that is transitory during the execution of a task) in fields of transient business components, rather than with standard fields in a regular business component.

Configuring a task applet is part of the larger process of creating tasks for implementing a business process framework. For more information about the procedures involved in configuring a task applet, see Chapter 5, "Configuring Task UI." See also "Creating Task Applets Using the Task Applet Wizard" on page 89. For more information on transient data binding, see "Transient Business Component" on page 30.

Task Applets and How They Differ from Standard Applets

A task applet differs from a standard applet in the following ways:

- A task applet is based on a transient business component. Transient business components are used to display data from a task that is discarded when the task ends — for example, data shown to allow the end user to select values that form the branching condition of a task. Depending on the transient data value selected by the end user, the next task step will vary.
- A task applet has a specialized frame class, CSSSWEFrameTask.
- A task applet is always a form applet, not a list applet.
- A task applet can be based only on grid Web templates. These grid Web templates differ from standard Web templates in that they do not display the applet title or the applet menu. A standard applet, particularly the form applet, can be based either on the available standard templates or on grid Web templates.

For more information on how task applets differ from traditional Siebel applets, see "Configuring Task Applets" on page 89.

Task Chapter

A *task chapter* is a list of task steps, grouped under a common display name (the chapter name). The task chapter allows you to define a logical grouping of task steps, and displays the chapter name alongside with the task view names in the current task pane.

Initially, only chapter names of a task are displayed in the current task pane. When the first task view step of a chapter is executed, the associated chapter name is expanded in the current task pane to show names of completed task views. When chapters are created correctly, they can give the end user an idea of what lies ahead in the processing of this task, and how far along this task has been executed.

Figure 6 shows a task named Asset to Agreement with three chapters.

Figure 6. Example of a Task Chapter

The chapter names give more details on the processing stages of this task: Create/Revise Agreement, Cover Assets, Submit Agreement. The example also shows that the first 4 steps of the Create/Revise Agreement stage have been completed.

For more information on the procedures involved in the configuration of task chapters, see "Configuring Task Chapters" on page 80.

Task Group

The *task group* is a list of hyperlinks to tasks and commands that are displayed in the Context pane.

Figure 7 shows the task group model as UML class diagram. Classes shown in yellow are configured using Siebel Tools, while those shown in orange are manipulated using the run-time client and appropriate administration views.

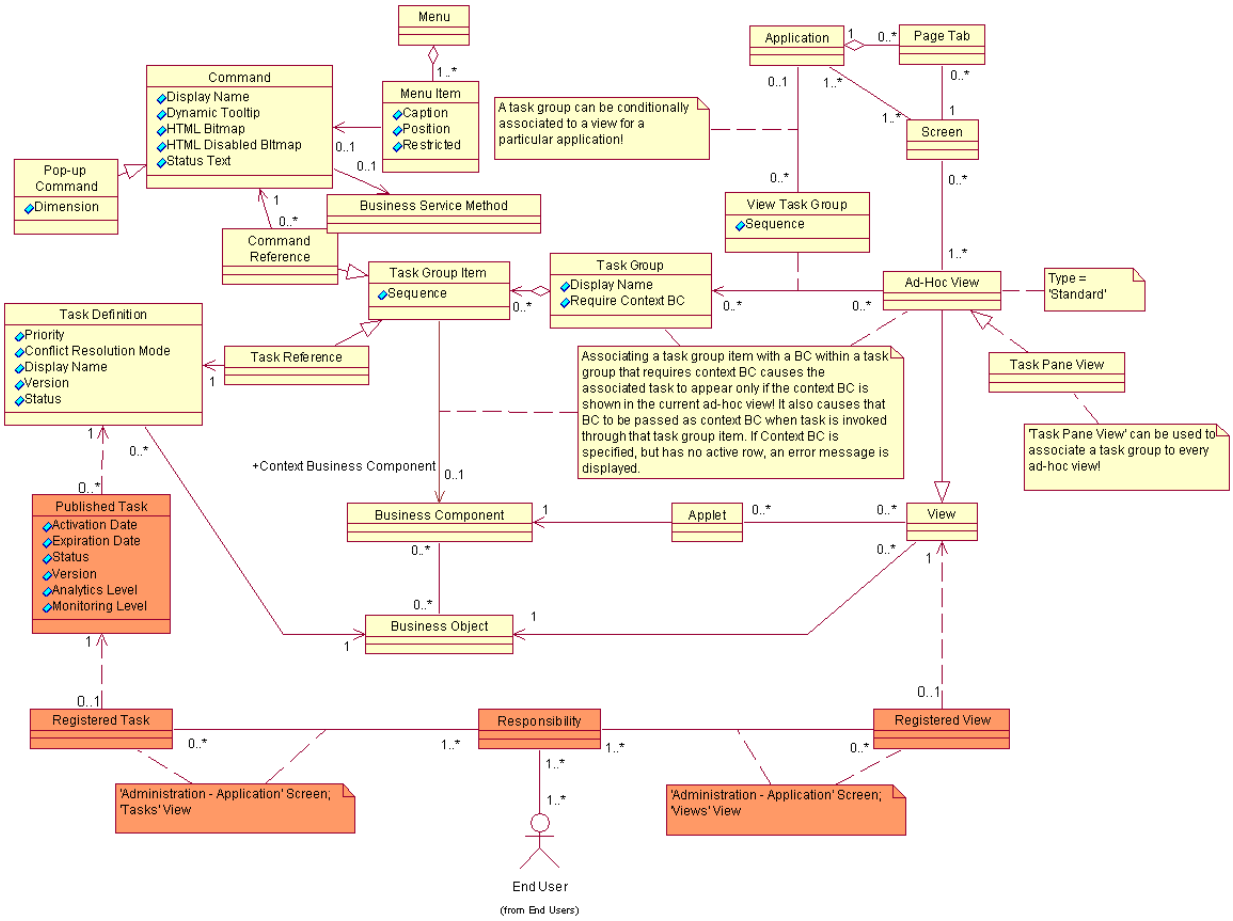


Figure 7. UML Class Diagram of a Task Group

The model gives the option of specifying a context business component for each of the task group items, but that association is meaningful only if that item is part of a task group that requires context. In other words, context-sensitive tasks need to be grouped together.

If these conditions are met, clicking on the context-sensitive task group item causes the Task UI framework to instantiate the associated task and pass it the current record of the context business component. If the task's business object matches the business object of the current ad-hoc view, the task shares the business object instance and its business components with the ad-hoc view. Otherwise, the task designer must use the Context BO Name, Context BC Name, and Context BC Id system task properties to properly initialize task's business object and context business component See "Associating Tasks with a Business Object Instance" on page 87 for more details.

Figure 7 also illustrates the following conditions that a task must satisfy to show up in the Context pane. The task must:

- Be associated with a task group that is associated with current ad-hoc view in current application, or with Task Pane View. Task Groups associated with Task Pane View are shown before any view-specific task groups, which are ordered by the sequence.
- Be published to current run-time DB.
- Have an activation date that has passed, or is unspecified.
- Have an expiration date that has not passed, or is unspecified.
- Be licensed to be used by current user.
- Be associated with at least one of the current user's responsibilities.
- Have a context business component that is either unspecified; or the specified context business component is instantiated in the current ad-hoc view and the task group requires the context business component.
- Exist in the repository. The task group and metadata definitions must exist in the repository.
- Be assigned to a view in the repository.

Task groups and the task items within the task groups are sorted primarily in ascending order of their sequence number, and secondarily (since sequence number is not forced to be unique) in ascending alphabetical order of their display names.

Task Pane

One of the three possible left-side panels that can be displayed in the Action pane, along with the iHelp pane and the Search pane.

The Task pane is displayed in the Action pane when it is controlled by the Task UI framework. If a task is currently running, the Task pane displays the Current Task pane. Otherwise, it displays the Context pane.

Task Playbar

The *task playbar* is an applet containing buttons that allow the end user to control the execution of the task.

The Task UI framework provides a standard playbar applet (shown in Figure 8) with buttons for forward and backward navigation, as well as for pausing and canceling the task. The standard playbar applet satisfies the majority of business requirements, but the framework also provides the ability to modify the look and behavior of the task playbar by using a custom playbar applet.

Figure 8. Standard Task Playbar

The task playbar should be positioned at the top right of the view on task views that don't require vertical scrolling. When vertical scrolling is required, the Playbar applet should be positioned at both the top and the bottom right of the view.

The following topics describe behavior of the buttons on the standard task playbar:

- “Navigating Forward” on page 44
- “Navigating Backward” on page 45
- “Pausing the Task” on page 45
- “Cancelling the Task” on page 46

For more information on how to use task playbar applets in your implementation using mobile clients, see “About Task Playbar Applets” on page 91.

Navigating Forward

When an end user clicks the forward navigation button (labeled Next in Figure 8), the Task UI framework first validates the data in the current view.

If there is a validation error, a pop-up error message is displayed, forward navigation is aborted, and the user is given the opportunity to fix the data.

If data in the current view is successfully validated, then the Task UI framework follows the task flow, executing all task steps until the next view step, or end step, whichever comes first. If any of the steps throws an exception not handled by the task flow, then the Task UI framework aborts forward navigation, pops up the error message, and returns control to the end-user in the same view at which forward navigation had been initiated. If the task's forward execution successfully arrived to the next task view step, the task context associated with that step is applied to the specified business components, and the associated view is presented to the end user.

The button for forward navigation can be labeled either Next, Submit, or Finish; and is controlled by the task developer using the Forward Button Type task step property. The label itself doesn't change the behavior of the framework, but is a hint intended to give the end user an idea of what happens when the forward navigation button is clicked:

- **Submit** indicates that task's data transaction is about to be committed to permanent storage for enterprise-wide consumption, and when committed, can not be rolled back. This typically occurs at the end of the task. However, for tasks that use intermediate commit points (using the Commit step), the Submit button should appear in the last task view before the commit step.
- **Finish** indicates that clicking the forward navigation button ends the task. It should be used only in tasks where task transaction is fully committed before the last view, which then typically serves as a summary view. This also applies to tasks that don't use the transaction mechanism, and instead directly work with permanent storage.
- **Next** is used in all other cases.

Navigating Backward

Clicking the backward navigation button (labeled Previous in the standard playbar applet, see Figure 8) causes Task UI framework to first validate data in the current view, and then display the last displayed view. This feature is most useful in the case where the end user makes a mistake (for example, entering erroneous data, or choosing the wrong option) during task execution, and needs to go back to fix it.

NOTE: The fix can often cause subsequent forward navigation to enter a different branch in the task flow.

Validation is done similarly to forward navigation. The exception being that records in the current view that have been created within task transaction with deferred validation option (through insert step with Defer Write Record property set to TRUE), are not validated. Consequently, data inserted in the task transaction, which has not been validated before the commit step, is not committed to permanent storage.

NOTE: Data in transient business components is validated because it is not part of the task transaction.

Before presenting the previous view, the Task UI framework attempts to reconstruct that view by applying the original search and sort specifications. It also attempts to reinstate the original current record. However, no reinstatement of original values is performed. The view presents the latest data within the transaction, so if the data presented on the view has later been changed, those changes are seen when navigating back to the target view.

Backward navigation can be continued until the first view in the task, and can freely cross subtask boundaries. At the first view shown in the task, the backward navigation button is disabled.

Backward navigation can be prevented by setting the Disable Previous task view step property to TRUE. This feature is useful for situations like a summary page where the task transaction has already been committed to permanent storage, and perhaps a business process has been initiated to start processing the transaction, so going back to change the transaction is no longer possible.

Pausing the Task

Clicking the Pause button in the task playbar applet causes the current task to be paused. This means that the Task UI framework first performs validation of the current view in the same way as in backward navigation. If validation succeeds, the framework executes the pause event handler if one is defined for current task.

Only if the event handling succeeds does the framework continue to persist to database the current state of both the task (including current view, navigation history, and local data), and its transaction. The end user is shown the ad-hoc view from which the current task was invoked. The Inbox item associated with the paused task instance is set to a paused state, which allows the end user to resume the paused task either from the Universal Inbox, or (if the task instance has been associated with any business objects) from an ad-hoc view that shows tasks associated with those business objects.

The Pause button on the playbar can be disabled by setting the Disable Pause task step property to TRUE. However, note that the task is implicitly paused when the user tries to navigate the Web browser outside of the current task (for example, by clicking on a site map, or a screen tab). The task is also implicitly paused if the user's session times out. Disabling the pause button does not prevent implicit pause, but it can be used to give a hint to end user that pausing is not recommended (for example, in the summary view).

Cancelling the Task

Clicking the cancel button in a playbar applet causes the current task to be cancelled. The impact of this action depends on the state of the task. If the task has never been paused, then cancelling it leaves no trace of it (except for the task timestamp metrics). On the other hand, if the task has been paused at least once, then cancelling it resets its state to the last paused state. The task's transaction is also rolled back. If there were any intermediate commits since the last pause, then the transaction is rolled back to the state at the last intermediate commit. Otherwise, the transaction is rolled back to the state saved during the last pause of this task instance.

Task View

The *Task View* is a type of view that is made up of either task applets and or standard ad-hoc applets that contain a playbar for a user to navigate forwards and backward throughout a task. The task view is based on transient business components.

For information about implementing a task view in your application, see “Configuring Task Views” on page 91.

Most Siebel users have become accustomed to ad-hoc views. These are traditional Siebel views that consist of the form, list, tree, or chart applet types. In an ad-hoc view, the user has the freedom to navigate through the records using a variety of navigational techniques, including the scroll bar, drilldowns, screen tabs, view drop-downs. These types of views can contain a superset of the fields and controls that the user needs to accomplish a business function.

The ad-hoc type of view gives the user increased freedom, yet requires the user to be more knowledgeable and presents a higher probability of making mistakes. In contrast, a task view is targeted toward exactly what the user needs to accomplish. The task view contains a playbar applet that allows the user to traverse through the business function. Additionally, a task view typically contains fewer fields, controls, and applets to minimize exposure and reduce the opportunity for mistakes.

See also:

- “Standard Applet” on page 39
- “Task Applet” on page 40
- “Task Playbar” on page 43
- “Transient Business Component” on page 30

Task View Step

A step in a task flow that presents a user interface view to the end user. The task view step allows you to map a business process to the user interface.

See also:

- “Task Flow” on page 22
- “Task Step” on page 25

Object Types for Task UI

There are a number of object types that are used specifically for the implementation of Task UI in Siebel business applications. The usage of many of these object types is described in this document. For a complete description of these object types, see the *Siebel Object Types Reference*. The following object types are used in Task UI:

- Task
- Task Branch
- Task Branch Criteria
- Task Branch Criteria Value
- Task Chapter
- Task Chapter Locale
- Task Event
- Task Event IO Argument
- Task Group
- Task Group Item
- Task Group Locale
- Task Locale
- Task Metric
- Task Property
- Task Step
- Task Step Context
- Task Step IO Argument
- Task Step Locale
- View Task Group
- View Task Group Locale

To control the visibility of these and other object types in the Object Explorer, you can use the Options dialog box. For details on how to do this, see *Using Siebel Tools*.

4 Implementing Task UI

This chapter presents a scenario of one way in which Task UI can be implemented. It includes the following topics:

Scenario for Iterative Development of a Task UI Implementation

This chapter is targeted at business analysts, development managers, and developers working on enterprise applications using a task-based UI framework. This chapter highlights a recommended process for the development of applications using a task-based user interface.

To simplify the explanation of this recommended process, it is explained through the example actions of four individuals who perform primary roles in the development of a Task UI implementation in a fictional organization called Acme, Inc:

- *User A* is a business analyst. She has detailed knowledge of the way Acme, Inc. conducts financial planning and reporting. She is also a sophisticated consumer of various technologies, from personal computers to wireless personal devices, but she has no experience in programming using a formal language.
- *User B* is a business process developer. He has a computer science degree and several years experience applying various business process management technologies at Acme, Inc. His experience has brought him insight into the various business processes performed in his organization.
- *User C* is a UI developer. She has a degree in art history, as well as a strong technical background, including over five years of experience in building and testing usability of various user interfaces. In the last two years, she has been configuring UI for various customizations Acme has completed to Siebel-packaged applications.
- *User D* is the IT director driving the business process management initiative at Acme aimed at optimizing and automating their business processes to provide competitive advantage.

Defining Goals for a Task UI Implementation

User D and *User A* meet for a brainstorming session in which they list on a whiteboard Acme's business processes within their domain. They then try to assess best candidates for applying BPM technology. They determine that the main criteria are return on investment (ROI) and risk.

After careful examination, *User D* and *User A* pick expense reporting and reimbursement as a candidate for the Task UI implementation. They choose expense reporting and reimbursement because it is a common task that almost every employee performs, and at the same time, most of the employees do it only irregularly.

The process seems relatively simple and well-defined, which makes both the risk and the investment required quite low. At the same time, however, *User A* and *User D* are aware that Acme's current way of handling this process—through email and spreadsheets—is labor-intensive and error-prone, which means the process is also expensive and slow. This situation makes potential ROI very large.

By using Siebel technology to automate a part of the process—with Task UI guiding the manual part of the process while enforcing business rules—*User A* and *User D* estimate they can cut costs by 30%, speed up the process by 400% (from the current median time of 12 days to 4 days from submission to reimbursement), and save the company $200,000 for each year through better enforcement of company reimbursement policies.

Modeling the Business Process

Given the low risk and the high ROI, Acme executives easily and quickly approve the proposed project. *User A* is now ready to model the business process the company is about to modernize. After some analysis, she decides to name the business process *Expense Reimbursement*. The name reflects the fact that the objective of the business process is not simply for employees to report expenses, but for the company to reimburse employees for the valid expenses they have incurred while conducting business for Acme.

The next step is to break down the business process into separate activities. An experienced business analyst, *User A* tries to answer the question: *Who does what?* Her answer to that question identifies three abstract roles involved in this business process:

- Submitter
- Reviewer
- Payer

She also identifies three kinds of activities within the business process:

- Submit expense report (a task).
- Review expense report (a task).
- Pay expense report (a business service).

Submission is difficult to fully automate, thus it is easily identified as a task. Review is a candidate for automation, but to minimize the risk of fraud, *User A* decides (at least for this iteration) to also make it task. She also makes a note that this activity may be repeated several times through an approval chain. The third activity can be automated easily through integration with Acme's Oracle accounts payable (AP) application, so it is tagged as a business service.

Reviewing the Draft Model

User A has finished the first iteration of the business process model, so she meets with *User B* to get his feedback as early as possible.

User A has used the Process Designer in Siebel Tools to model the business process at hand as a long-running workflow process. This process (shown in Figure 9) is not yet executable, but (accompanied by her notes) it serves as a clear communication vehicle for sharing her analysis with *User B*.

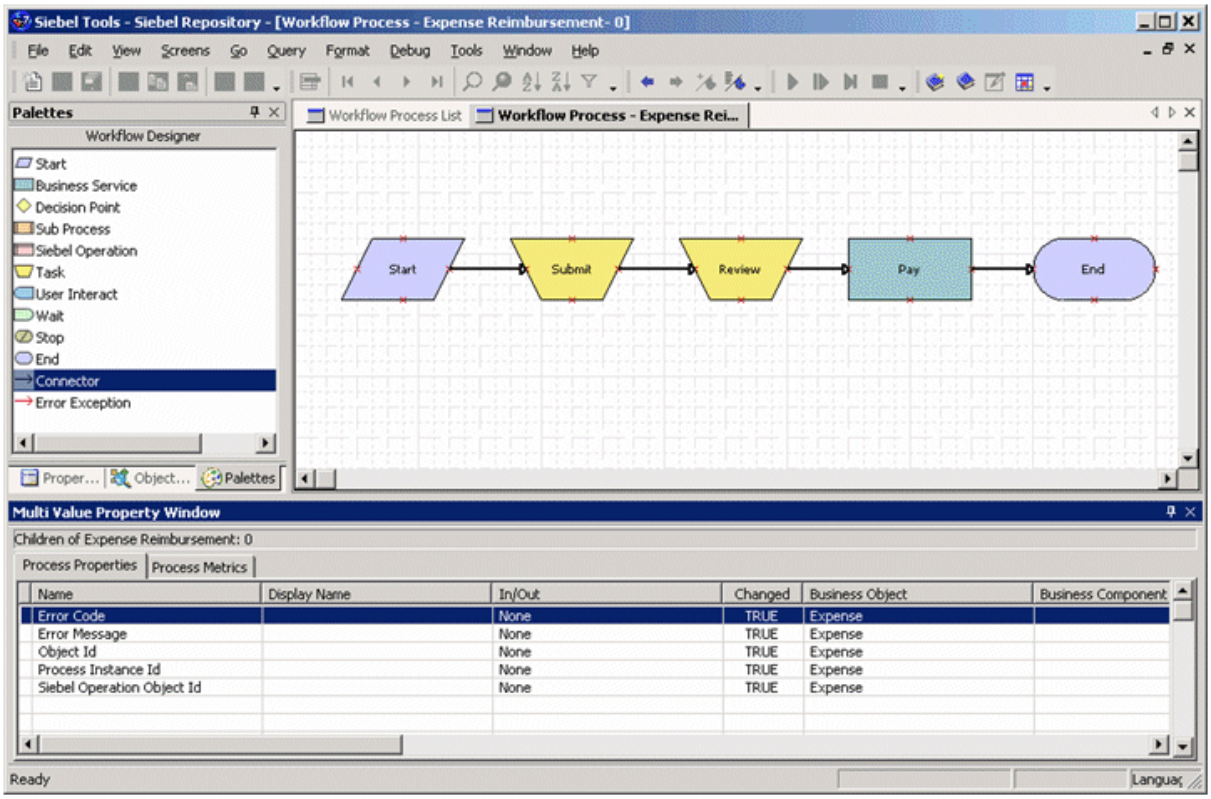


Figure 9. Draft Business Process Model in Siebel Tools

Being detail-oriented, *User B* quickly notices that *User A*'s model is incomplete, because she has not clarified the business rules that drive the approval. For example, when is a single approval enough, and when is it not?

User B also notices that the model does not cover a secondary scenario for which a review step finds that the expense report is ineligible and rejects it.

Because *User A* has modeled the process in Siebel Tools, *User B* is able to quickly help her refine the business process model during the review session. After doing this, he is ready to take over the process definition (the new version of which is shown in Figure 10) for further refinement into an executable process definition. By this point, he has developed a good understanding of the business process being automated.

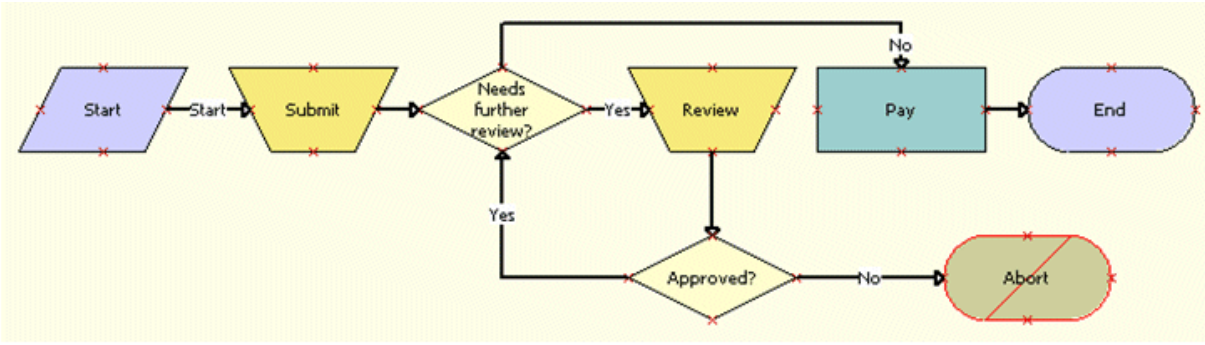


Figure 10. Refined Business Process Model

Creating an Executable Business Process Definition

By using Siebel Tools to model the business process, *User A* has saved *User B* a significant amount of time, because he is now able to use *User A*'s model as a starting point for his iterative refinement.

After some analysis, *User B* notices that—although logically it belongs to the business process model—the submission task must be taken out of the executable long-running workflow definition, because it is the submission task that must initiate the long-running workflow, and not vice versa. The submission task has to pass the expense report's ID as an input argument. For this purpose, *User B* decides to reuse the Object ID process property, which he now makes an input argument to the process. He lets *User A* know about the discovery.

Next, *User B* considers the Decision step that decides whether further review of the expense report is required. Being familiar with the full range of technologies provided by the Siebel 8 platform, *User B* determines it is best to add a call out to the business rules engine to make the further-review decision. He knows this will improve the agility of the business process by allowing the approval rules to change without requiring changes (and consequently re-testing and re-deploying) to this executable process definition.

When *User B* takes a closer look at the review task, he realizes that it must be assigned to a particular reviewer before it can be instantiated. He achieves that by calling Assignment Manager through a Business Service step that directly precedes the review task. The final flow is depicted in Figure 11.

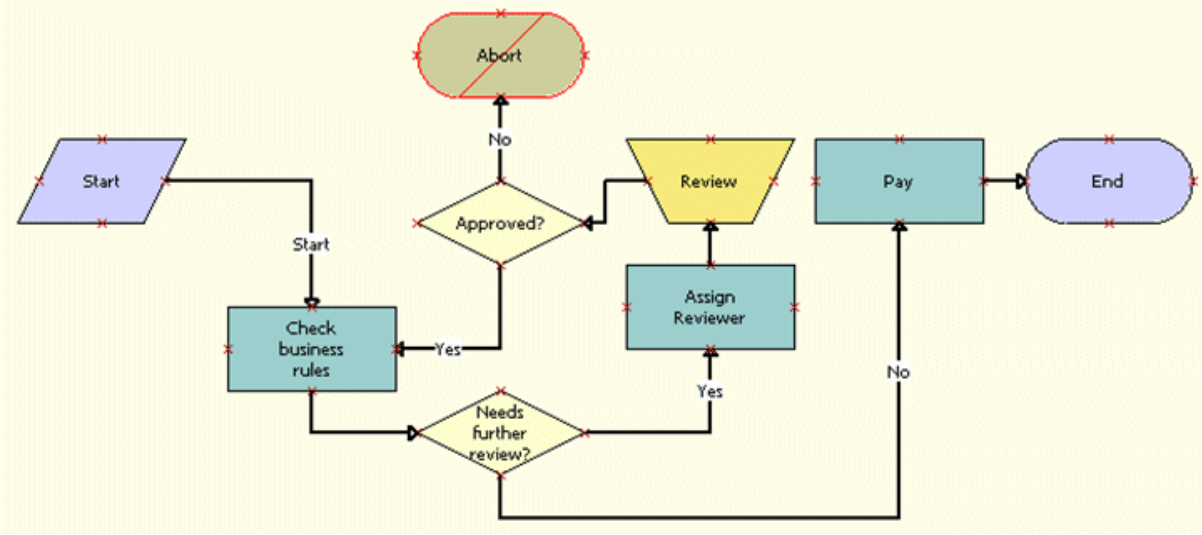


Figure 11. Executable Business Process Flow

All these revisions to the draft business model help to illustrate the differences in the level of abstraction between a business process model and an executable process definition. They also help to show why the roles of business analyst and business process developer can rarely be played by a single person.

User B continues refining (for example, by defining conditional branches, process properties, and input and output arguments) and unit-testing the long-running workflow process until it is ready for deployment, as explained in *Siebel Business Process Framework: Workflow Guide*.

Identifying Task Context

Even though she has handed over the business process definition to *User B*, *User A* still has a lot of work to do on the tasks within the process. She starts by identifying their context. Having been warned by *User B* that the submission task initiates the long-running workflow, she realizes that it is the end user who needs to manually start the submission task from the ad-hoc UI. She also realizes that this task is going to be frequently used by almost every employee of the company.

For these reasons, *User A* decides the submission task should be added to the *Common Tasks* task group. She also concludes that the submission task does not require any context passing. She considers that this task may, however, be frequently paused (for example, so the employee has a chance to find all required documents). Because of this pause requirement, *User A* decides that the task should use the expense report's description as the value of the Context field in the Universal Inbox, so as to easily distinguish between different instances of the same task.

Moving on to the review task, *User A* notes that is invoked from the long-running workflow, so it will be started through the Universal Inbox; for this reason, the review task does not need to be added to any task group. The review task does, however, require an expense report number to be passed as an input argument. A Boolean flag called *Approved* seems to *User A* to be a logical output argument. The appropriate context for the Universal Inbox would be a concatenation of the expense report's total amount and the submitter's name (for example, $450.00 from Aaron Jones).

Designing the Task Flow

User A now proceeds with identification of the activities within each task, as well as definition of the task flow.

For the task called *Submit expense report*, this yields the following activities:

1. Create expense report header.
2. Create expense items.
3. Review and submit the expense report.

Just as she did when she was drafting the business process model, *User A* creates this task flow using Siebel Tools. This time, however, she uses the Task Designer rather than the Process Designer. She creates the three identified activities as three separate task view steps. Because she is not familiar with the View and Applet editors, she enters only the view step names, without linking the view steps to actual task views, since at this stage they still do not exist.

The second task, Review expense report, is simple, requiring only a single task view step: Review expense report.

Designing the Task UI

At this point, *User A* is ready to start working with *User C*, who is an expert in human-computer interaction, but also familiar with the Expense Report business object and its underlying data model. Together, they sketch on a whiteboard the layout of the identified views. *User C* identifies business components that will be (re)used, as well as applets that will be created for the two new tasks (*Submit expense report* and *Review expense report*).

One important design decision that must be made at this point is the determination of whether all expense items should be created in the same view, or whether each item should be created in a separate view. After some brainstorming—including considering that most users will use this task quite frequently—*User A* and *User C* agree that productivity is more important than UI simplicity, and they conclude that all items should be created in the same view.

As an advantage of this top-down approach, *User C* realizes the possibility of reuse for a pair of applets in two similar views: the third view in the submission task displays the same data as the only view in the review task.

Configuring the Task UI

When consensus is reached on the layout of all four task views for the two tasks, *User C* can proceed with the UI configuration in the following sequence (imposed by Siebel Tools):

1. Making any required additions and/or changes in the business logic configuration (including business object, transient and nontransient business components, links, picklists, LOVs, and so on.)
2. Creating and/or re-using the required applets.
3. Fully configuring the applets.
4. Creating and/or re-using the required task views.
5. Fully configuring the task views.

Configuring the Task Flow

User C now updates the tasks' definition by:

- Adding links to the newly created task views.
- Adding an insert Operation step before the first view in the submission task.
- Adding an update Operation step after the only task view step in the review task.
- Reviewing and updating all properties and multi-value properties for all steps in the task definition.
- Making sure that the tasks contain no remaining validation errors or warnings. This includes setting the Instance ID task property as identified by the business analyst.

Configuring the Task Group

User C's last configuration job is to add the submission task to the Common Tasks task group, and to recompile the SRF with all the new UI configuration, including the task group update.

Deploying the Tasks

User C clicks the Publish and Activate button to deploy the submission task in the development environment.

Setting Up Access Control

At this point, *User C* wants to see the results of her work in the mobile client she started from Siebel Tools. But first she must allow the test user's responsibility to see and run the submission task.

Unit Testing

Finally, the phrase *Submit Expense Report* shows up in the Context pane, and clicking its link causes the first view of that task to be displayed. *User C* is excited, but her joy is spoiled by the fact that she forgot to include an Expense Description field in the Expense Header Applet.

Several iterations later, she is ready to demo the submission task to *User A*, who is quite pleased, but who notes there are a few details that could use some touch-up. Finally, version 16 of the submission task coupled with version 22 of the SRF seems to comprise the iteration that is ready for integration testing.

The review task is somewhat harder to test in isolation, as it requires tight integration with a long-running workflow. Since the task itself is fairly simple, *User C* decides to defer its unit testing until it is integrated with the long-running workflow.

Integration Testing

It is now time to test the whole business process as follows:

- Publishing and activating the latest versions of both tasks
- Publishing and activating the latest version of the long-running workflow
- Compiling the latest supporting UI configuration (including any business layer configuration changes) into the SRF

Testing is done in the development environment, using the Siebel application server and the thin client, since long-running workflows cannot run on the mobile client. Defects in task-flow logic are analyzed using the Task Debugger.

System Testing

When the configuration seems to satisfy the business requirements, the whole configuration is migrated to a testing environment using Application Deployment Manager (ADM). It is important to note that this typically includes migration of any new or updated LOVs, which are typically used heavily in Task UI. After functional verification, performance and scalability tests are done to assess major risk areas.

Deploying the Tasks and Workflows to Production

After the system testing phase has finished successfully, the new tasks and the long-running workflows are ready to be deployed to the production system. This activity is best done as part of a system-wide configuration upgrade, as new functionality typically requires SRF changes, and thus some server down-time.

Why It Is Best to Take an Iterative Development Approach

Following the best practices of modern software development to minimize the risk of project failure, your development process must be iterative and incremental.

Using an iterative approach means that feedback from any phase can cause reiteration of any of the previous phases. For example, a significant performance issue may require another iteration of UI design, which would naturally cause reiteration of all subsequent phases.

Using an incremental approach means that the best way of mitigating the risks of using new technology such as the Task UI framework is to start with a smaller scope, deliver it to customers, and use the feedback to incrementally build functionality.

In an effort to keep this chapter brief, iterations and incremental releases have not been fully illustrated. Within every real-life project, however, iterations and incremental releases will be a fact of life, and therefore it is best to plan for them.

5 Configuring Task UI

This chapter discusses the concepts and procedures involved in configuring applications that developers must perform in Siebel Tools to implement a task-based user interface. It includes the following topics:

NOTE: As discussed in Chapter 4, "Implementing Task UI," a prerequisite for being able to perform the configuration work described in this chapter is that you have a clear understanding of the high-level business process that you are going to implement. This includes the identification and clear delineation of all long-running workflows and tasks in that business process.

About the Task Designer

The Task UI editor (Task Designer) has much in common with the Workflow editor (the Siebel Business Process Designer), because the Task and Workflow components are configured using similar visual programming languages. But there are also significant differences. In this guide, references are made to the *Siebel Business Process Framework: Workflow Guide* to explain the functionality that Task UI shares with Workflow.

TIP: For efficient use of the limited space on your screen, you can stack the Object Explorer, Palettes, and Properties windows on top of each other (as shown in Figure 12). In this way, each window is accessible through tabs at the bottom. These three windows are commonly used during Task UI configuration, but rarely at the same time. For details on stacking windows in Siebel Tools, see *Using Siebel Tools*.

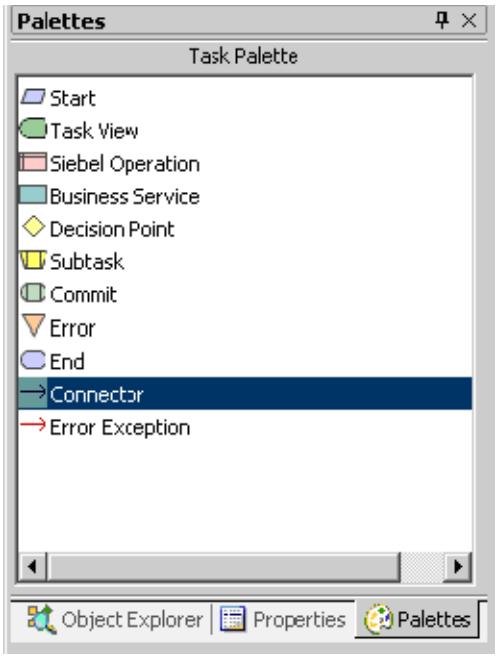


Figure 12. Stack of Palettes, Object Explorer, and Properties Windows

To launch the Task Designer

1. In Siebel Tools, click Task in the Object Explorer.

   Tools displays a list of tasks in the Tasks OBLE.

2. In the Tasks OBLE, right-click the task you want to edit, and choose Edit Task from the pop-up menu.

   This opens the selected task in the Task Designer.

NOTE: If the status of the task you selected is Completed, a new version of that task is created with the status of, In Progress.

Tasks OBLE

You can use the Tasks object list editor (OBLE) in Siebel Tools to work with tasks. For a selected task, right-click and use the pop-up menu to:

- Create a task. For more details, see "Creating New Tasks Using the Tasks OBLE" on page 62.
- Copy a task.
- Delete a task.

  NOTE: Tasks with the status Completed cannot be deleted. To delete such tasks, change their status to Not In Use first, by clicking the Expire button in the WF/Task Editor toolbar.
- Revise an existing task.
- Validate a task.
- Publish a task.
- Publish and activate a task (using a button in the WF/Task Editor toolbar).
- Archive a task definition in a .SIF file. For more information, see the topics on working with archive files in *Using Siebel Tools*.
- Reset the version number on a task to 0. The version can be reset only if the original version 0 of the task no longer exists in the repository.

NOTE: The pop-up menu can be used to perform each of the actions listed in the preceding topic, with the exception of publishing and activating a task. Instead, this is done with a button in the WF/Task Editor toolbar. For more details, see "Deploying Tasks from Siebel Tools" on page 115.

Task UI Wizards

The following wizards provide assistance in defining the components of a task:

- The Task wizard guides you through the creation of a task, including the initial steps required to run a task.
- The Task Applet wizard helps you create task applets that map to transient business components.
- The Task View wizard helps you generate the views with which users interact, making sure that the required elements are included and exposed.
- The Transient Business Component Wizard helps you create transient business components and fields.

Creating New Tasks

You can define a new task flow in two ways:

- Using the New Task Wizard. See "Creating New Tasks Using the New Task Wizard" on page 62.
- Using the Tasks object list editor. See "Creating New Tasks Using the Tasks OBLE" on page 62.

NOTE: Some task flows are subtasks. For more information, see "Subtasks" on page 20, "Creating New Subtasks" on page 63, and "Configuring Subtask Steps" on page 71.

Creating New Tasks Using the New Task Wizard

The New Task wizard consists of a single page that allows you to specify the most important task flow properties. The wizard allows you to do the following:

- Specify a project for the task.
- Give the task a name and a display name.
- Indicate whether the task is a subtask. (For information on subtasks, see "Subtasks" on page 20.)
- Select the business object on which the task operates.
- Specify the default transient business component for the task, if necessary.

NOTE: For a detailed description of the task flow properties, see the description of the Task object, in the *Siebel Object Types Reference*.

To create a new task using the New Task wizard

1. In Siebel Tools, choose New Object from the File menu.
2. In the New Object Wizards dialog box, click the Task tab.
3. Select the Task icon, and click OK.
4. Use the wizard to complete the steps of creating a new task.

NOTE: You can also access the New Task wizard through the pop-up menu available by right-clicking in the Tasks OBLE.

Creating New Tasks Using the Tasks OBLE

As an alternative to using the New Task Wizard to create a new task, you can use the Tasks OBLE.

NOTE: It is highly recommended that you use the New Task Wizard to make sure you have the necessary properties and objects associated with your task. For more information, see "Creating New Tasks Using the New Task Wizard" on page 62.

To create a new task using the Tasks OBLE

1. In Siebel Tools, lock the project for the new task, or if necessary, create a new project.
2. In the Object Explorer, select the Task object.
3. In the Tasks OBLE, right-click and choose New Record.
4. At a minimum, specify the following properties:
   - Task Name
   - Project
   - Business Object

   For more information, see "Task Properties" on page 64.

Creating New Subtasks

You create a new subtask in the same ways that you create a main task. The only difference in the task definition is the value entered for the Is Subtask flag. If the task is a subtask, make sure the Is Subtask flag is checked. When using the New Task wizard, you can create a subtask by checking the box labeled Create as a subtask. For more information on subtasks, see "Subtasks" on page 20 and "Configuring Subtask Steps" on page 71.

Editing Task Flows

In building tasks, you can use many different types of steps. For a description of the various types of tasks, see "Task Step" on page 25. You build the sequence of steps that comprise a task flow by configuring the task using the Task Designer. For more information about the Task Designer, see "About the Task Designer" on page 60.

Information on configuring tasks is organized as follows:

Task Properties

Task properties are displayed in the Properties window when no task step or connector is selected in the Task Designer. For a description of the properties of the Task object, see the *Siebel Object Types Reference*.

Task Step Properties

Task step properties are displayed in the Properties window when a task step is selected in the Task Designer. For a description of the properties of the Task Step object, see the *Siebel Object Types Reference*.

NOTE: Not every property applies to every type of step. The Properties window displays only the properties applicable to the currently selected step.

Configuring Task Step Context

You can define the context of a task step to identify the search specification that filters the data on which the step is performed. You do this by configuring a Task Step Context object (a child object of the Task Step object).

For example, if you want to update Opportunities with a lead quality of Poor, you use the update operation of a Siebel Operation step with a Task Step Context object configured accordingly. For a description of the properties of the Task Step Context object, see the *Siebel Object Types Reference*.

To configure task step context

1. In Siebel Tools, navigate to the task that contains the step you want to provide the context for.
2. Open task in the Task Designer, and select the appropriate step.

   NOTE: The task must be either a Siebel Operation step or a Task View step.
3. Click the Task Step Context tab in the Multi Value Property Window.
4. In the list area of the Multi Value Property Window, right-click and choose New Record from the pop-up menu.
5. Enter a Name for the task step context, and select a Type (Literal or Expression).
6. If the context type is Expression, enter the name of the business component to evaluate the expression for the Expression Business Component property.
7. Enter a Search Specification for the context.
   - If the Type property is set to Literal, enter a literal value in the form of an expression, (for example, = 100).
   - If the Type property is set to Expression, enter an expression (for example, [Status] LIKE '*Open*'). The expression is evaluated by the specified Expression Business Component.

   CAUTION: Define your Siebel operation search specification as efficiently as possible, so that it matches only the smallest necessary set of rows. Search specifications that select a large set of rows can cause severe performance degradation.

8 (Optional) You can specify a Filter Business Component. This business component provides the group of records on which the context search is performed.

Controlling Step Display

You can use the Display Name and Display Name Type properties of the Task View object to selectively inhibit the display of steps shown to the end user in the Current Task pane. For a description of these properties, see the *Siebel Object Types Reference*.

To control the display of steps by inhibiting selected steps from being shown to the end user, do one of the following:

- Enter a valid Display Name for the first step, and leave Display Name blank for subsequent steps.

  By setting the Display Name property to an empty string, the step is not displayed in Current Task pane. The step shown for the first view continues to be shown as the current step for the others.

  NOTE: This option is useful when there is a sequence of views performing what the end user thinks of as one logical step.

- Set the Display Name Type property to Unique.

  Using this setting means that Display Name is added to the Current Task pane the first time that the step is encountered, but it is not added subsequently.

  NOTE: This option is useful in loops (for example, a loop in which the end user enters line items). When the type is set to Unique, the step is shown in the Current Task pane the first time through the loop, but not in subsequent iterations.

  When a loop includes only a single view, then using this option alone is sufficient. When a loop includes multiple views, the Unique flag can be combined with an empty Display Name to achieve the desired effect. The first view in the loop has Display Name filled in and is marked as Unique. The other views in the loop have an empty Display Name. The result is that the Display Name of the first view is displayed in the Current Task pane for the entire loop.

Configuring Task View Steps

A Task View step is used to represent the step in the task flow where a Siebel view is presented to the end user. As an application developer, you can navigate from this Task View step in the Task Designer to the Task View UI editor. You can also bring up the run-time view from this step. See the *Siebel Object Types Reference* for descriptions of the properties of the Task View Step object.

Adding a Task View Step to a Task

The procedure for adding a task view step is similar to adding any type of step.

To add a task view step

1 Open the appropriate task in the Task Designer.

2 From the Task Palette, drag a Task View step, and drop it on the Task Designer at the desired location.

3 Bind the new task view step to a task view:

   a Click to select the task view step in the Task Designer.

   b In the Properties window, select a task view for the Task View property.

4 Select the appropriate settings for the Disable Cancel, Disable Pause, and Disable Previous properties.

   If you want to disable any of the navigation buttons set the corresponding property to TRUE.

5 Enter a Display Name, and pick a Display Name Type.

   The Display Name Type controls the display of the task steps in the Current Task pane. It is used in conjunction with the Display Name property. If set to Unique, consecutive task views with display names of the same value are displayed only once in the Current Task pane (see "Controlling Step Display" on page 65).

6 Select a Forward Button Type.

   The Forward Button Type property determines the label and type of the forward button (Next, Submit or Finish).

7 Select the appropriate settings for the Retain Applet SearchSpec, Retain Task SearchSpec, and Retain User SearchSpec properties.

   See the *Siebel Object Types Reference* for descriptions of these properties. See also "Record Context Is Lost" on page 131 to understand how these settings can affect the way business component states are preserved across task views.

8 (Optional) You can add a Task Step Context to identify the search specification that is applied to filter the data on which the view step is performed.

   For information on how to configure Task Step Context, see "Configuring Task Step Context" on page 64.

9 (Optional) After configuring the task step context, you can add output arguments as needed.

   The task view step does not return output arguments. However, you can use the Output Arguments tab (in the Multi Value Property Window) to configure updates to task properties. For example, you can copy data entered by the end user through a business component field to a task property. For more information on output arguments, see "Configuring Input and Output Arguments for Task Steps" on page 75.

10 Make sure you save your changes before leaving the Task Designer.

Drilling Down on Task View Steps

After you have bound the task view step to a task view, you can drill down on the task view layout by double-clicking on the task view step in the Task Designer. This displays the Task View UI editor, which you can use to edit the task view. You can also display the run-time appearance of the view from the Task View UI editor.

Configuring Business Service Steps

A business service step allows you to call a business service that executes predefined or custom actions in a task flow. See "Best Practices for Invoking Business Services" on page 144 for more information.

You can define business services by navigating to Site Map > Administration - Business Service, or by selecting the business service object in Siebel Tools. The methods and arguments you define in your business service appear in the Arguments list applets for the business service.

To define a business service step

1. Open the appropriate task in the Task Designer.
2. From the Task Palette, drag a Business Service step, and drop it on the Task Designer at the desired location.
3. In the Properties window, enter or modify the step name, and then enter a description of the purpose of the Business Service step.

   NOTE: If the Properties window is not showing, right-click the step, and choose View Properties from the pop-up menu.
4. For the Business Service Name property, select the name of the service to be invoked.

   The picklist contains the business services defined in Siebel Tools or the Siebel client.
5. For the Business Service Method property, select the method for invoking the service. The choices available for this field depend on the service you select.
6. Specify the input and output arguments as necessary.

   For more information on input and output arguments, see "Configuring Input and Output Arguments for Task Steps" on page 75.

Configuring Commit Steps

A commit step is a step that explicitly commits the task data stored in temporary storage to the Siebel database. You can create exception branches from the commit step to handle errors resulting from the persistence of temporary data.

To define a Commit step

1. Open the appropriate task in the Task Designer.

2 From the Task Palette, drag a Commit step, and drop it on the Task Designer at the desired location.

3 In the Properties window, enter or modify the step name, and then enter a description of the purpose of the commit step.

NOTE: If the Properties window is not showing, right-click the step, and choose View Properties from the pop-up menu.

Configuring Start Steps and End Steps

A Start step is the initial step in a task; it starts the task. An End step instructs the Task UI framework to end the task instance and transfer the data in temporary storage to the Siebel database. It also provides one last chance to change (through the output arguments) the task properties returned from the task output arguments to the caller. Each task flow must contain only one end step.

NOTE: Workflow supports condition branches for the start step. However, for Task UI, condition branches for the start step are not necessary.

To define a Start step

1 Open the appropriate task in the Task Designer.

2 From the Task Palette, drag a Start step, and drop it on the Task Designer at the desired location.

3 Enter a Name and Description for the step.

NOTE: If the Properties window is not showing, right-click the step, and choose View Properties from the pop-up menu.

To define an End step

1 Open the appropriate task in the Task Designer.

2 From the Task Palette, drag an End step, and drop it on the Task Designer at the desired location.

3 Enter a Name and Description for the step.

4 (Optional) You can define output arguments for the end step.

An output argument allows you to store a resulting value in a task property. This value can then be passed to other tasks. See “Configuring Input and Output Arguments for Task Steps” on page 75 for more information.

Configuring Siebel Operation Steps

Siebel Operation steps perform operations such as Insert, Update, Query, Delete, NextRecord, and PrevRecord on the business components in a task’s instance of a business object.

Information about Siebel Operation steps appears in this topic as well as in the following additional topics:

- "Configuring Task Step Context" on page 64
- "Specifying Task Step Output Arguments" on page 77

You can define Siebel Operation steps for any business component associated with the business object selected for the task. If you want to update a business component not associated with the business object, you must associate the business component with the business object using Siebel Tools.

All fields are available for update and insert except fields based on multi-value groups and calculated fields. If you want to update a field based on a multi-value group, you can define a business component for the field, and link the business component to the object using Siebel Tools. For example, Account Team is based on a multi-value group, so it cannot be updated by selecting the Account business component. However, you can create a business component called Account Team and then associate it with the Account business object using Siebel Tools. You can then select Account Team as the business component to update it with the Siebel Operation step.

NOTE: After executing a Siebel Operation step, the Siebel Operation Object ID process property stores the row ID of the record that was operated on.

To define a Siebel Operation step

1. Open the appropriate task in the Task Designer.
2. From the Task Palette, drag a Siebel Operation step, and drop it on the Task Designer at the desired location.
3. Enter a Name for the step, and a Description of its purpose.
4. Select the Type of operation. The available choices are:
   - Insert
   - Update
   - Query
   - QueryBiDirectional
   - Delete
   - NextRecord
   - PrevRecord

   For information on NextRecord and PrevRecord operators, see the *Siebel Business Process Framework: Workflow Guide*.

   NOTE: Verify that updates or inserts of fields that have dependencies are valid fields. For example, if you have a service request task and your task is updating the area and subarea fields, make sure that the values specified for the subarea field are valid for that associated area.
5. Select the name of the business component.
6. (Optional) You can define fields for the Siebel operation. For more information, continue to "Configuring Siebel Operation Field Lists" on page 70.

7 (Optional) You can define search specifications for the Siebel operation. For more information, see "Configuring Task Step Context" on page 64.

8 (Optional) You can define output arguments for the Siebel operation. For more information, see "Specifying Task Step Output Arguments" on page 77.

Configuring Siebel Operation Field Lists

To define the fields for the Siebel Operation step, you use the Fields tab in the Multi Value Property Window. The drop-down list for the Field Name property (on the Fields tab) lists the fields of the business component that is specified for the Siebel Operation step. You can select the field to be defined from this list.

NOTE: If you define the Siebel Operation step to perform an insert operation, make sure that the required fields have been added to the Siebel Operation step. System fields and predefaulted fields are populated when the step is created.

To define fields for a Siebel Operation step

1 In the Task Designer, select the appropriate Siebel Operation step, and open the Multi Value Property Window.

2 In the Multi Value Property Window, click the Fields tab.

3 Right-click the list area, and choose New Record.

4 For the Field Name property, select the field to be updated.

5 For the Type property, choose an input argument type. The available choices are:

- Business Component
- Expression
- Literal
- Task Property

6 Specify the appropriate input value, based on the field type:

a If the field type selected is Business Component, select the applicable values for the Business Component and Business Component Field properties.

b If the field type selected is Expression, enter an expression for the Value property. (Alternatively, you can click the drop-down button to launch the Expression Builder.)

c If the field type selected is Literal, enter a value for the Value property.

d If the field type selected is Task Property, select a task property for the Property Name property.

7 If you define multiple fields, and the field values have dependencies and must be entered in a specific order, then specify the order in the Preferred Sequence property starting with number 1.

The Preferred Sequence property allows you to specify the order in which the field values are entered in the business component.

8 Enter comments as appropriate.

For information on updating a field based on a multi-value group, see the *Siebel Business Process Framework: Workflow Guide*.

Configuring Subtask Steps

Subtasks allow you to modularize the task flow. When a task flow diagram becomes so large that readability is significantly compromised, you can break a larger task into smaller subtasks. A subtask is launched from a subtask step in the main task flow. It is treated as a special task and is versioned just like the main task. The primary goals of subtasks are to:

- Improve task flow readability by breaking large tasks into modules.
- Decrease development and maintenance costs by reusing common Task UI sequences.
- Maintain a clean, consistent, and intuitive programming model.

Similarly to the Workflow subprocess step, you can pass information into and out of a subtask through the input and output arguments. Input arguments allow you to populate task properties in the subtask with information from the parent task.

When the subtask is invoked, the task properties of the subtask are initialized with the values of the input arguments of the subtask step. Output arguments allow you to populate data from the subtask back to its parent task. When the subtask returns to its parent, the parent task can read the subtask's task properties through the output arguments of a subtask step.

You can configure the input and output arguments for a subtask through the Multi Value Property Window of the subtask. For example, if you want to pass the Object ID from the main task to the subtask, you do this through the input arguments.

NOTE: The Object ID passed to a subtask must be null, not the parent's Object ID, when the subtask creates a child object.

For example, if the subtask has a return code of SUCCESS or FAILED, you can send the return code to the parent task through output arguments. For more information, see "Configuring Input and Output Arguments for Task Steps" on page 75.

Because parent tasks and subtasks have separate local task properties, argument passing causes the data to be copied between the parent tasks and subtasks. However, for hierarchical task properties, copying hierarchical data can be expensive. For this reason, a hierarchical data type is passed by reference between parent tasks and subtasks.

Exception branches are allowed within a subtask. However, exception branches are not allowed to go into and out of a subtask step. This restriction makes sure that a subtask can only exit from its end step in the forward direction.

The subtask configuration model is somewhat different from the Workflow subprocesses. The major differences are:

- A subtask and its parent task share the same process instance; while in Workflow, the main process and its subprocesses run in separate instances.

- Task UI subtasks and Workflow subprocesses support the notation of local process properties differently. Each subtask invocation does not create a new instance; instead a new context is created for each invocation for storing local process properties. However, each Workflow subprocess invocation creates a process instance and has its own process properties.

The process of configuring a subtask step involves the following procedures:

1. Make sure that the task to be called by the subtask step is already defined as a subtask.

   The task must be defined before you create the subtask step, and it must be defined as a subtask. Do not link a nonsubtask into a subtask step.

2. Define the subtask step (see "To define a subtask step").
3. Define input and output arguments for the subtask step. For more information, see "Configuring Input and Output Arguments for Task Steps" on page 75.

To define a subtask step

1. In the Object Explorer (in Siebel Tools), click the Task object type.
2. Verify that the task to be called as a subtask is already defined (and defined as a subtask). Create it, if necessary.
3. Open the parent task in the Task Designer.
4. From the Task Palette, drag a Subtask step, and drop it on the Task Designer at the desired location.
5. In the Properties window, specify a Name for the subtask step, and enter a description of the purpose of the subtask.
6. For the Subtask Name property, select the subtask to be called by the subtask step.

If you need to define the input or output arguments for the subtask, see "Configuring Input and Output Arguments for Task Steps" on page 75.

Connecting Task Steps

Use the procedure that follows to define each branch.

NOTE: Tasks do not support parallel processing. Make sure that you define your conditions such that entry conditions are satisfied for only one branch at a time. If entry conditions are satisfied for multiple branches at once, the exact execution path is not guaranteed.

To connect two steps

1. In the Task Designer, drag a Connector arrow from the Task Palette to the design canvas, dropping the start of the arrow on the earlier of the two steps (for example, the Start step).
2. Drag the tip of the connector arrow onto the second of the two steps.

About Branching

Branching is configured through a Decision Point followed by multiple outgoing connectors, which may be associated with conditions. A decision point is a type of step that evaluates the conditions on the outgoing connectors to determine which step should be executed next. Because you cannot configure the order in which the conditions are evaluated, it is important that branching conditions are mutually exclusive.

The connectors coming out of a decision point are typically of one of the following types:

- **Condition**. The condition connector allows you to specify a condition that must be met to continue through this path. For more information, see "Specifying Branching Conditions" on page 73.
- **Default**. The Default connector does not have any associated conditions. The task flow follows the Default connector if no other connector's conditions for entry are met.

To connect steps using a branch

1. In the Task Designer, drag a Connector from the Task Palette to the canvas, dropping the start of the arrow on the earlier of the two steps.
2. Drag the end of the connector arrow to the second of the two steps.
3. Enter or modify the Name of the branch.

   NOTE: The name of the branch must be unique.
4. Select a Type for the branch. See the description of the Task Branch object in the *Siebel Object Types Reference* for more information on the types of branches.

   CAUTION: Always define a Default branch step in case some work items do not meet any of the conditions you define.
5. Enter comments as appropriate.
6. Define the conditions that apply to each branch. For information about branching conditions, see "Specifying Branching Conditions" on page 73.

Specifying Branching Conditions

You can define conditions and values for branches to control the flow of the task. For example, you can define a condition, based on the value of a priority field such that:

- If the priority is high, the task follows a branch that sends an email to a vice president.
- If the priority is medium, the email is sent to an engineer.

You specify the branching conditions for a branch using the Compose Condition Criteria dialog box. Figure 13 shows an example of the Compose Condition Criteria dialog box.

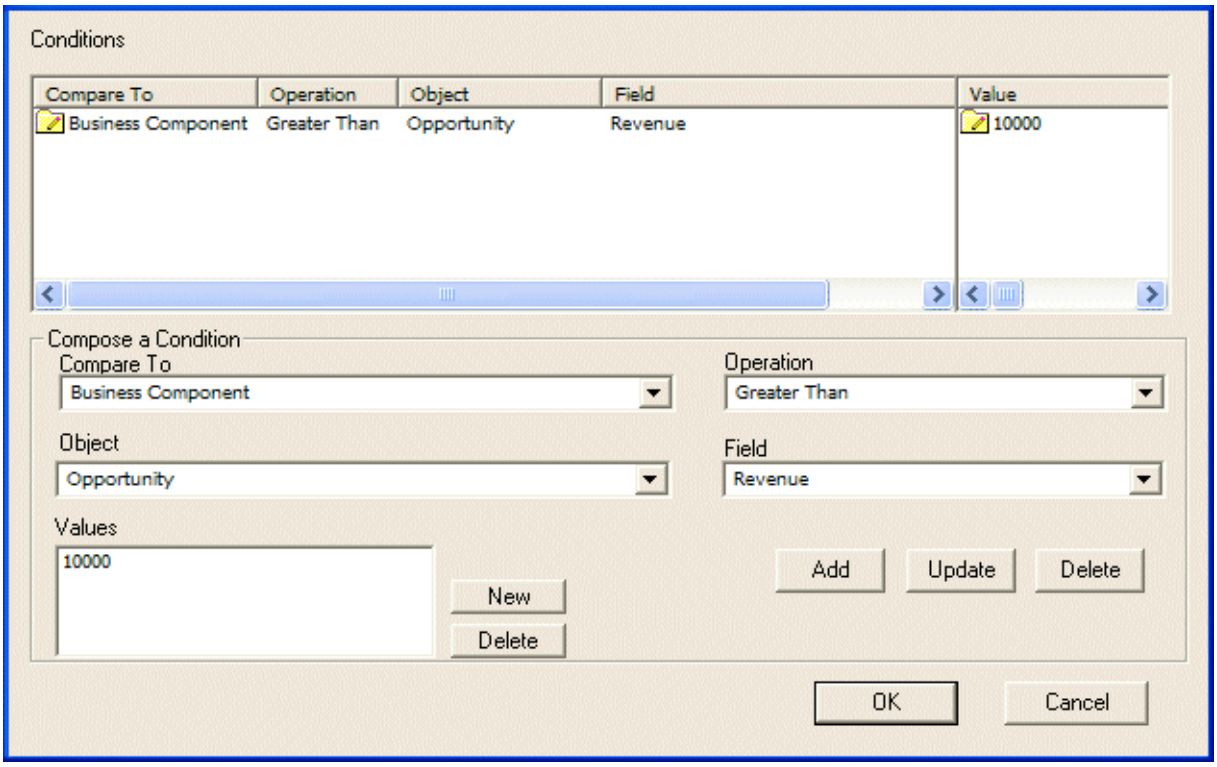


Figure 13. Compose Condition Criteria Dialog Box

To define conditions and values

1 Right-click the appropriate branch in the Task Designer, and choose Edit Conditions to display the Compose Condition Criteria dialog box.

NOTE: The values listed in the Compose Condition Criteria dialog box are constrained by the business object for the task, which is specified at the task level.

2 In the Compose Condition Criteria dialog box, select a value from the Compare To drop-down list.

- **Applet.** Uses the value in an applet field for the condition comparison.
- **Business Component.** Uses the value in a business component field for the condition comparison or when you are defining an expression.
- **Expression.** Uses an expression to evaluate a specific value.
- **Task Property.** Compares a process instance's process property value with a specified value.

3 Select an Operation to use for evaluating the values.

For a description of the available comparison operations, see the topic on building expressions in the *Siebel Business Process Framework: Workflow Guide*.

4 Enter an Object and Field, if applicable.

5 Enter any appropriate values in the Values box.

You can enter multiple records in the Values box. Task UI assumes an OR condition between values.

6 If you selected Expression in the Compare To field, enter the expression in the Values box.

For more information about expressions, see the *Siebel Developer's Reference*.

7 Click OK.

8 Make sure you save changes before closing the Task Designer.

Defining Multiple OR Conditions

You can define multiple conditions for each branch. Task UI treats multiple conditions with the AND operator. To define multiple OR conditions, use expressions.

The following example shows an expression comparing a business component field with today's date, using the OR operator, which allows you to compare multiple conditions:

```
([Close Date] <= Today()) OR ([Name] = 'Opportunity test1')
```

Configuring Input and Output Arguments for Task Steps

The Task Step IO Argument object (a child of the Task Step object) allows you to provide an input or output argument to some types of steps:

- Input arguments apply to business service steps, Siebel Operation steps, and subtask steps.
- Output Arguments apply to business service steps, Siebel Operation steps, task view steps, subtask steps, and end steps.

For a description of the Task Step IO Argument object and its properties, see the *Siebel Object Types Reference*.

NOTE: Business services, methods, and arguments can have hidden properties. For a business service, method, or argument to be displayed on a drop-down list in Siebel Tools, the Hidden property for the object must be set to FALSE. For more information, see *Siebel Business Process Framework: Workflow Guide*.

Specifying Task Step Input Arguments

Input arguments allow you to pass data to some task steps during task flow execution.

NOTE: Input arguments apply to business service steps, Siebel Operation steps, and subtask steps.

For example, if you want to pass the object ID from a main task to a subtask, you do this through input arguments. If the subtask is based on a different business object, you must pass the relevant row ID of the target object as the subtask Object ID task property. For subtask steps, the receiving end of an input argument is one of the task properties of the subtask. You can select this using the drop-down list in the Task Input field. The task properties of the subtask that are of type In/Out or In are displayed in the drop-down list.

To add an input argument to a task step

1. Open the task that contains the task step to which you want to add an input argument in the Task Designer.
2. In the Task Designer canvas, select the step for which you want to define an input argument.
3. In the Multi Value Property Window, click the Input Arguments tab.

   NOTE: For Siebel Operation steps, you use the Fields tab (rather than the Input Arguments tab) to specify input arguments.
4. Right-click and choose New Record.
5. Select an Input Argument (Field Name for a Siebel Operation step).

   This is the destination of the input argument.
6. Select an input argument Type, and specify the rest of the properties for the input argument as appropriate, based on the type selected:

   - **Business Component.** When the input argument Type is set to Business Component, you can select a business component and one of its fields in the Business Component and Business Component Field properties.
   - **Expression.** When the input argument Type is set to Expression, the Value property is used to specify an expression that is evaluated at run time to determine the input argument value.
   - **Literal.** When the input argument Type is set to Literal, the Value property of the input argument is used to specify a literal value for the input argument.
   - **Task Property.** When the input argument Type is set to Task Property, you can select a task property in the Property Name property.

   NOTE: Business component fields based on multi-value groups cannot be selected as values for input or output arguments. To use a field based on a multi-value group, you must define a business component for the field, and link it to the appropriate business object. See *Configuring Siebel Business Applications* for more information.

   NOTE: Calculated fields are not available as values for input or output arguments. If you want to use a calculated value, use an expression.

For descriptions of the properties of input arguments, see the description of the Task Step IO Argument object in the *Siebel Object Types Reference*.

Specifying Task Step Output Arguments

Output arguments allow you to store the value resulting from a Siebel operation, a business service, or a task view in a task property. This value can then be passed to a subsequent task step as an input argument.

NOTE: Output arguments apply to business service steps, Siebel Operation steps, task view steps, and end steps.

To define output arguments for a task step

1. Open the task that contains the task step to which you want to add an output argument in the Task Designer.
2. In the Task Designer canvas, select the step for which you want to define an output argument.
3. In the Multi Value Property Window, click the Output Arguments tab.
4. Right-click and choose New Record.
5. Select a Property Name.

   This is the name of a task property that is the destination of the value for the output argument.
6. Select an output argument Type, and specify the rest of the properties for the output argument as appropriate, based on the type selected:
   - **Business Component.** When the output argument Type is set to Business Component, you can select a business component and one of its fields in the Business Component and Business Component Field properties.
   - **Expression.** When the output argument Type is set to Expression, the Value property is used to specify an expression that is evaluated at run time to determine the output argument value.
   - **Literal.** When the output argument Type is set to Literal, the Value property of the output argument is used to specify a literal value for the output argument.
   - **Output Argument.** When the output argument Type is set to Output Argument, you can select a task property in the Output Argument property. The drop-down list allows you to select the task properties that are of type In/Out or Out.

For descriptions of the properties of output arguments, see the description of the Task Step IO Argument object in the *Siebel Object Types Reference*.

System Task Properties

System task properties are the same for Task UI as the system process properties are for workflow processes. There are five system task properties: Object Id, Siebel Operation Object Id, Error Code, Error Message, and Instance Identifier.

When a task is created, in addition to these five system task properties, Context BC Id, Context BO Name, and Context BC Name are also added to the list of task properties in the Multi Value Property Window. When a task is associated with an ad-hoc view by way of a context-sensitive task group, these three properties are populated by the run-time engine, as explained in "Determining Visibility of a Task Group" on page 103.

Object Id

Object Id is the Siebel Row Id of the work item being processed (the active row of the primary business component).

Siebel Operation Object Id

Siebel Operation Object Id is the object identification of an object that is updated, created, or queried during a Siebel Operation step. This system property is populated when a Siebel operation is executed.

Error Code

Error Code is an error symbol for the task instance. It is populated when a step returns an error.

Error Message

Error Message is the text describing the error, and it is populated when a step returns an error.

Instance Identifier

Instance Identifier is the object identification of the task instance. It is populated when the task is executed.

Context BO Name

Context BO Name is the name of the business object for the ad-hoc view from which this task instance was invoked.

Context BC Name

Context BC Name is the name of the business component to which the record identified context BC Id system task property belongs.

Context BC Id

Context BC Id is the id of the current record in the business component identified by Context BC Name at the time when this task instance was invoked.

Configuring Task Properties

You can use task properties to store values that you can use in task steps, either as input and output arguments, or for performing evaluations. Task Properties store values that the task retrieves from the database or derives before or during processing. You can base decision branches on the values in a task property, and pass the task properties as input and output arguments. You can also use the task property values in expressions.

There are two types of task properties: system task properties and user task properties. For a description of system task properties, see "System Task Properties" on page 77. User task properties can be of type String, Number, Binary, Date, Hierarchy, Integration Object, and Strongly Typed Integration Object. You use the Multi Value Property Window to define and configure task properties.

To define a task property

1. Open the task to which you want to add a task property in the Task Designer.
2. In the Task Designer, click the open area of the canvas (not on a task step).

   The Multi Value Property Window displays tabs associated with the task.
3. In the Multi Value Property Window, click the Task Properties tab.
4. Right-click the list area of the Task Properties tab, and choose New Record from the pop-up menu.
5. Enter a Name, and select the Data Type for the task property.
6. (Optional) You can assign a default value for the task property in the Default field if necessary.
7. (Optional) You can change the values of Access Mode and In/Out.

   By default, Access Mode and In/Out are set to R/W and None respectively. Typically, you do not need to change these values. However, you can make changes as follows:

   a. You can change Access Mode to make the task property read-only.

   b. You can change In/Out to In, Out, or In/Out, which allows you to use the task property as input, output, or both for the task. The default is None, which confines the task property to the task instance executed.
8. If you chose Integration Object as the Data Type (in Step 5), select the Integration Object to be used for the task property.
9. (Optional) You can provide Comments, as necessary.
10. Step off the record to save your changes.

Configuring Task Metrics

Task metrics collect and store various task data that is regularly loaded into a data warehouse and analyzed using an OLAP (online analytical processing) tool, such as Oracle Business Intelligence. Two supported types of metrics are timestamp and property metrics.

Configuring Task Timestamp Metrics

Timestamp metrics allow you to collect the times when certain task execution events (for example, task started and task completed) occur.

You can activate timestamp metrics collection after the task has been deployed. After the task is deployed, set the Analytics deployment field to Timestamp or All to enable timestamp metrics to be collected at run time.

Configuring Task Property Metrics

Property metrics allow you to collect task data that are bound to task properties. To configure the metrics collection for task properties, you must define (in the task) the subset of metrics to be captured, and then activate the collection of the metrics after deployment of the application.

To configure task property metrics

1. In the Object Explorer (in Siebel Tools), click the Task object type.
2. In the Tasks OBLE, select the task that you want to define metrics for.
3. In the Object Explorer, click the Task Metric object type.
4. Right-click the Task Metrics OBLE, and choose New Record from the pop-up menu.
5. For the Metric Name property, select the desired metric from the list of predefined metric names.
6. For the Property Name property, select a task property from the list of those that have been defined for the task.

   The run-time value of the metric is set to the run-time value of the task property referenced. You change the value only for customizable metrics, and not for system metrics, whose values are set by the task engine.
7. Make sure the Inactive property is set to FALSE.

   The Inactive column contains a checkbox, which is not checked by default. It provides a convenient means for task designers to disable a metric and reenable it later.
8. After the task is deployed, set the Analytics deployment field to Property or All to enable property metrics to be collected at run time.

Configuring Task Chapters

Organizing task steps into chapters is optional, but when you assign one step to a chapter, all the other steps in the same task must also be assigned to chapters. You can create any number of chapters, but for a given task, create only as many as you will use.

NOTE: Limit the use of chapters to only those chapters that you need. If you create chapters without associated steps, the Validator throws an error at validation time.

You can use colors to differentiate between chapters. Steps that share the same color belong to the same chapter grouping. Figure 14 shows the Multi Value Property Window, in which you create chapters. Each chapter record includes a Color property as part of its definition.

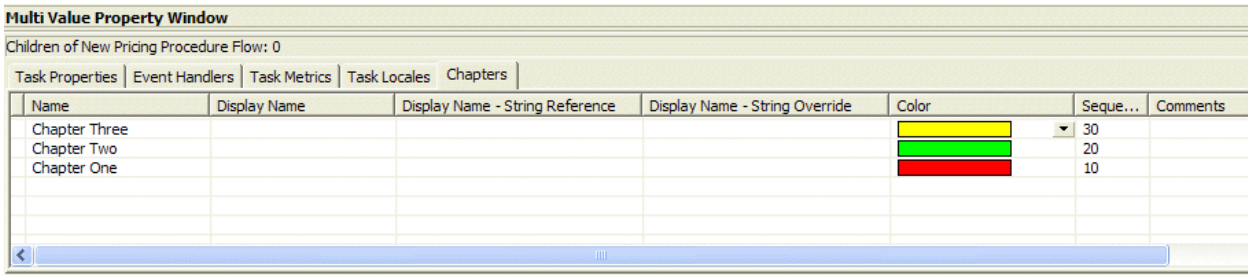


Figure 14. Chapters Tab in the Multi Value Property Window

When assigning multiple task steps to a single chapter, the task steps must be adjacent to one another in the sequence. For example, you cannot assign the fourth step to the same chapter as the first step (Chapter 1) if the two steps in between belong to a different chapter (Chapter 2). Step 4 must belong to either Chapter 2, or Chapter 3. Likewise, forward navigation that moves to a previous chapter is not possible. This is called a chapter cycle. Validation rules prevent the flow from moving this way.

However, you can design looping of task steps between chapters. For example, a step within Chapter 2 goes back to a step in Chapter 1 after a decision point.

NOTE: You can assign task steps to chapters only for main tasks, not for subtasks. If you assign a chapter to a subtask step, the steps in the subtask are grouped under the same chapter header of the subtask step while the subtask is being executed.

To define a chapter

1. Open the appropriate task in the Task Designer.
2. Right-click the Task Designer, and choose Show Chapters from the pop-up menu.

   If no chapters have been defined for the task, the colors in the task flow disappear, leaving all the task steps white.
3. In the Multi Value Property Window, click the Chapters tab.
4. Right-click, and choose New Record from the pop-up menu.
5. Specify the properties for the chapter:
   a. Enter a Name for the chapter.
   b. Select a Display Name - String Reference.
   c. Choose a color from the pop-up Color pick applet and click OK.

d Select a Sequence.

By default, the Chapters tab records are sorted on sequence number.

NOTE: When you create a new chapter, the Sequence attribute pre-defaults to the next available number in the series 10, 20, 30, and so on. This is so that when you need to add additional chapters to a task, there is room for the new chapters to fit in the sequence and you avoid having to renumber the existing chapters.

To assign a task step to a chapter

1 Make sure chapters have been defined for the task.

2 In the Task Designer, right-click a step, and choose Assign Chapter from the pop-up menu.

NOTE: To assign multiple steps to the chapter at one time, click and drag your mouse to form a box around the steps.

3 Select the appropriate chapter, and click OK.

The list is limited to the task chapters defined in the task.

4 Repeat until all the steps within the task have been assigned to chapters.

NOTE: Make sure each step has a chapter assigned. While using chapters is not required, when you have assigned one step within a task flow to a chapter, every step within the flow must have a chapter assigned to it.

To delete a chapter

1 In the Multi Value Property Window, click the Chapters tab.

2 Right-click the chapter you want to delete, and choose Delete Record from the pop-up menu.

The chapter assignment is removed from the applicable steps in the Chapters view. The steps remain, but now they are not assigned to any chapter.

Configuring Error Steps

Task UI supports exceptions in a manner similar to Workflow. A task can have, for example, a business service step that calls an external system synchronously. If an external system is down, an error can occur. Such errors are handled with an exception branch and an error step. The error step raises an error and returns control to the current view.

When a task flow arrives at an error step, the end user sees the error message on top of the current view, and internally the task is reset to the most recent view step before the error step was encountered.

To define an error step

1 Open the appropriate task in the Task Designer.

2 From the Task Palette, drag an error step and drop it on the desired location on the Task Designer.

3 In the Properties window, enter a Name for the step and a Description of its purpose.

NOTE: If the Properties window is not showing, right-click the step and choose View Properties from the pop-up menu.

4 For the Error Code property, select an error code.

NOTE: To define a custom error message, select an error code starting with WF_ERR_CUSTOM and provide input arguments as necessary. See "Defining Custom Error Messages Using Input Arguments" on page 83 for more information.

Defining Custom Error Messages Using Input Arguments

You can define a custom error message for an error step using a custom error code and input arguments. The input arguments for an error step are the substitution variables in the error message. Substitution variables are identified by a percent symbol (%).

To define an error step with a custom error message

1 Open the appropriate task in the Task Designer.

2 From the Task Palette, drag an error step and drop it on the desired location on the Task Designer.

3 In the Properties window, enter a Name for the step and a Description of its purpose.

NOTE: If the Properties window is not showing, right-click the step, and choose View Properties from the pop-up menu.

4 For the Error Code property, select an error code starting with WF_ERR_CUSTOM (for example, WF_ERR_CUSTOM_1).

The error message displayed will be a substitution variable (for example, %1).

5 Add an input argument to define the text of the custom error message:

a For the Input Argument property, enter the appropriate substitution variable (for example, %1).

b Select a Type for the input argument. This is the source of the value that is used for the error message text.

c Specify the remaining properties, based on the Type selection.

NOTE: See "Specifying Task Step Input Arguments" on page 75 for more information about defining input arguments.

d Step off the record to save the changes.

e Repeat as necessary for multiple input arguments.

Configuring Exception Branches as Error Handlers

An exception branch is a type of branch designed for handling system and user-defined errors. An example of a system generated error is a failure when sending an email notification. A user-defined error can be trying to submit an order that was incomplete. Exception branches are illustrated in the Task Palette as red connectors.

To define an exception branch

1. Open the appropriate task in the Task Designer.
2. From the Task Palette, drag an Error Exception connector and drop it on the desired location on the Task Designer, connecting it to an existing step icon. Make sure that the end of the connector is attached to the step.
3. In the Properties window, enter a Name for the connector.

   NOTE: If the Properties window is not showing, right-click the connector, and choose View Properties from the pop-up menu.
4. For the Type property, select Error Exception or User Defined Exception.

   NOTE: The task controller treats user-defined exceptions the same way as error exceptions.
5. Double-click the exception icon (in the Task Designer) to display the Compose Condition Criteria dialog box.
6. Define the conditions that apply to the exception.

Configuring Task Event Handlers

The configuration of event handlers for Task UI employs the use of two object types: Task Event and Task Event IO Argument. The process of defining task event handlers is described in the following topics:

- "Defining Task Event Handlers" on page 84
- "Defining Task Event IO Arguments" on page 85

Event Handlers cannot be associated with subtasks. Therefore, you cannot add Task Event objects to a Task object if the Is Subtask property of the Task object is set to TRUE.

For a description of the Task Event and Task Event IO Argument objects and their properties, see the *Siebel Object Types Reference*.

Defining Task Event Handlers

The Task Event object (a child of the Task object) defines event handlers for certain Task UI events (such as Cancel, Delete, Pause, and Resume). This is where you specify the actions to take when such events occur.

To define a task event handler

1. In the Object Explorer (in Siebel Tools), click the Task object type.
2. In the Tasks OBLE, select the task that you want to define an event handler for.
3. In the Object Explorer, click the Task Event object type.
4. Right-click the Task Events OBLE, and choose New Record from the pop-up menu.

5 Select a Name for the event handler.

The drop-down list allows you to select from the available types of events to handle (Delete, Pause, Pre-Cancel or Resume). This is the name for the event handler.

6 Specify the type of event handler using one of the following sets of properties:

- **Workflow Process**. To use a Workflow process to handle the event, select the name of the workflow process for the Workflow Process property.

  NOTE: If you select Workflow Process to handle the event, you can not define input and output arguments. However, you can run the Workflow process as a business service handler. To do this, select Workflow Process Manager for the Business Service Name property and RunProcess for the Business Service Method property.

- **Business Service**. To use a business service to handle the event, select the Business Service Name and Business Service Method to handle the event.

NOTE: If the Workflow Process property contains a value, the Business Service Name and Business Service Method properties are ignored. The specified workflow process is the event handler.

7 (Optional) You can now provide input and output arguments for the event handler (see "Defining Task Event IO Arguments" on page 85).

Defining Task Event IO Arguments

The Task Event IO Argument object (a child of the Task Event object) defines the input and output arguments to the event handlers.

NOTE: If you have entered a Workflow process name in the parent task event object, you can not define input and output arguments. However, you can run the Workflow process as a business service handler. For details, see the procedure for "Defining Task Event Handlers" on page 84.

When the Input/Output property is set to Input, an input argument name of the event handler, whichever type being configured in the parent Task Event object, should be entered into the Argument property. When the Input/Output flag is set to Output, an output argument name of the event handler should be entered into the Argument property.

To define an input argument for a business service method

1 In the Object Explorer (in Siebel Tools), click the Task object type.

2 In the Tasks OBLE, select the appropriate task.

3 In the Object Explorer, click the Task Event object type, and select the appropriate task event in the Task Events OBLE.

4 In the Object Explorer, click the Task Event IO Argument object type.

5 Right-click the Task Event IO Arguments OBLE, and choose New Record from the pop-up menu.

6 Select Input from the drop-down list for the Input/Output property.

NOTE: Leave the Name property blank for now, it is a system-defined property.

7 For the Argument property, select the name of an argument.

The picklist shows the input arguments available for the business service method specified in the parent Task object. After you select an Argument, the Name property is populated.

8 For the Type property, select a source type (Business Component, Expression, Literal, or Task Property). This is the source of the value for your input argument.

9 Specify the remaining properties, based on the Type selected in the previous step. Fields not applicable to the destination type are disabled:

- **Business Component.** When the source Type is set to Business Component, you can select a business component and one of its fields in the Business Component and Business Component Field properties.
- **Expression.** When the source Type is set to Expression, the Value property is used to specify an expression, which is evaluated at run time to determine the input argument value.
- **Literal.** When the source Type is set to Literal, the Value property of the input argument is used to specify a literal value for the input argument.
- **Task Property.** When the source Type is set to Task Property, you can select a task property in the Property Name property.

To define an output argument for a business service method

1 In the Object Explorer (in Siebel Tools), click the Task object type.

2 In the Tasks OBLE, select the appropriate task.

3 In the Object Explorer, click the Task Event object type, and select the appropriate task event in the Task Events OBLE.

4 In the Object Explorer, click the Task Event IO Argument object type.

5 Right-click the Task Event IO Arguments OBLE, and choose New Record from the pop-up menu.

6 Select Output from the drop-down list for the Input/Output property.

NOTE: Leave the Name property blank for now, it is a system-defined property.

7 For the Type property, select a destination type (Business Component, Expression, Literal, or Output Argument). This is the type of destination for the value of your output argument.

8 Specify the remaining properties, based on the Type selected in the previous step. Fields not applicable to the source type are disabled.

- **Business Component.** When the output argument Type is set to Business Component, you can select a business component and one of its fields in the Business Component and Business Component Field properties.
- **Expression.** When the output argument Type is set to Expression, the Value property is used to specify an expression, which is evaluated at run time to determine the output argument value.
- **Literal.** When the output argument Type is set to Literal, the Value property of the output argument is used to specify a literal value for the output argument.

- **Output Argument.** When the output argument Type is set to Output Argument, you can select a task property in the Argument property. The picklist shows the output arguments available for the business service method specified in the parent task object.

9 For the Property Name property, select the name of a task property.

This is the destination of your output argument. For example, if you want to populate the value of the output argument into the Object Id task property, then select Object Id.

Associating Tasks with a Business Object Instance

You can use Task UI to support scenarios where a task is started and paused by one user, and then subsequently resumed by another user. For example, a task can be initiated by one call center agent, and paused when the customer needs to end the call. Then a different call center agent resumes the task when the customer calls back. You can configure this functionality by associating the task with a particular instance of a business object.

Supporting this usage scenario requires specific configuration of the task flow as described in this topic. It might also require customizing the user interface, as explained in the topic "Configuring Ad-Hoc Views for Task Transfer" on page 105.

To associate a task instance with a business component instance, you must invoke the Associate method of the Task Administration business service from the task instance. The Associate method requires the following arguments:

- **ObjectId**. The Id of the business component record with which the task is to be associated.
- **ObjectType**. The name to be used for the association.

  This is subsequently used as part of the search specification. The name (ObjectType) is not required to match the business component name, but it can be helpful to keep them in sync.

The Associate method must be invoked as early in the task as possible. For tasks that create new records that must be associated, this can be performed anywhere between the view that first validates the new record and the first subsequent commit step. The associated record can not be seen outside of the task before the task's transaction is committed.

For tasks updating an existing record, association with that record should be done before the first view in the task. In this case, the ObjectId parameter of the Associate method typically takes the value of the Context BC Id task property.

The Task Administration business service also provides a Disassociate method, which removes an existing association between the current task and the given business component record. This option is useful for cases where the record to be associated with the task is dependent on the selected branch of the task. In this case, disassociation should be included in the compensating part of the task flow.

NOTE: A task instance can be associated with any number of records of any type. The Task UI framework does not enforce constraints on the number or type of associations. For example, a particular task instance could be associated with a single account and three opportunities. However, associations consume computing resources, so they should not be created without an identified business need.

The Task UI framework caches associations until the current task is paused, at which time they are committed to the Siebel database and visible to other users. This behavior is a performance optimization to support the case where a task is never paused, or an association with a particular object is first created and then deleted before the first pause. This optimization is important primarily for Task UI developers debugging their configuration. Users do not notice any difference, because only paused tasks can be transferred.

This mechanism of associating tasks instances with business component records has the following limitation: tasks created by long running processes that were never started can not be associated with business objects. Such use cases must use either manual or automated escalation mechanisms to avoid deadlocks.

Configuring the Inbox Context Field

When a task is paused, the task state and the history stack are serialized and stored in the database. An inbox item, which may be used to resume the task, is created in the inbox in the database. The Inbox Item list view displays the tasks that belong to a given user. By default, every inbox item shows the task name and the user who created the task instance.

In some cases, you might want to display certain instance-specific messages in the inbox item list. This instance-specific message can be information that helps the owner of the task instance to distinguish between task instances that have the same task name, or it may be instructions that are essential to the current task instance owner.

Defining an Instance Identifier

You can configure a task to populate instance-specific messages into the Inbox Context field in the Inbox Item list view. A predefined task property called Instance Identifier is provided for this purpose. The information you populate into the Instance Identifier task property is saved and displayed in the Inbox Context field when the task is paused or completed.

To configure the message to be displayed, add an output argument to the appropriate task step, specifying the Instance Identifier task property for the output argument's Property Name property.

NOTE: The instance identifier can be configured only for task steps that allow output arguments. For more information on how to add output arguments, see "Specifying Task Step Output Arguments" on page 77.

Initiating a Long-Running Workflow

As with Workflow, you can invoke a long-running workflow as part of a task.

For more information about long-running workflows, see the *Siebel Business Process Framework: Workflow Guide*.

Configuring UI Objects Used in Tasks

In configuring the user interface for a Siebel implementation of Task UI, your work employs a task view wizard and entails configuring transient business components, user properties, radio buttons, applet messages, as well as other items, such as titles and custom buttons.

The process of configuring the user interface covered in this topic involves the use of Web templates that are modified versions of the standard Siebel Web templates used for ad-hoc views. For details on Task UI-specific Web templates, see "About the Task View Templates" on page 92. Information on configuring the user interface for tasks is organized as follows:

Configuring Task Applets

The following topics outline the concepts and procedures for configuring task applets:

Creating Task Applets Using the Task Applet Wizard

This topic explains how to use the Task Applet wizard to create a task applet.

Before you create a task applet using the Task Applet wizard, you must have already created the task and transient business component needed for the applet.

To create a task applet using the wizard

1. Launch the New Task Applet wizard:
   a. In Siebel Tools, choose New Object from the File menu.
   b. In the New Object Wizards dialog box, click the Task tab.
   c. Select the Task Applet icon, and click OK.

   Alternatively, you can launch the wizard by right-clicking in the Tasks OBLE, and choosing New Object Wizards > Task Applet from the pop-up menu.
2. Specify the appropriate options on the General screen:

a Select a project for the task applet.

b Enter a name for the task applet.

The name is used to refer to the applet when laying out a view and cannot be the same as the name of an existing applet.

c Enter a display title for the task applet.

This is the user-visible label for the applet within the view and may be left blank if no title is desired.

d Select the task this applet will be associated with.

NOTE: If no task is selected for the applet, the applet can be reused for multiple tasks. If the applet is associated with a task, the applet can only be used for that task.

e Select the appropriate upgrade behavior (Admin, Non-Preservable, Preserve).

f Select the transient business component (TBC) for the applet.

3 Click Next to proceed to the Web Layout - Fields screen.

4 Select the fields from the TBC that you want to appear on the Web layout.

5 Click Next to proceed to the Finish screen.

This screen displays the parameters that will be used to create the new task applet. You can use the Back button to return to the previous screens and make changes if necessary.

6 Click Finish to generate the task applet.

About Configuring Applet Methods

By default, standard applets used in a task view have restrictions on what users can do with them. An individual applet behaves differently according to whether it is used within or outside of a task. These restrictions prevent users from making changes to records and altering the record context in ways that could cause issues with the underlying task-flow logic.

You can relax these restrictions by setting the EnableStandardMethods user property. The restrictions are applied in addition to other configured restrictions on the applet, such as No Insert, No Update, and so on. These restrictions are described for each applicable applet in Table 2.

Table 2. Operation Restrictions Applied on Applets

Operation	Form	List	Pick	MVG	Association
Query	No	No	Yes	Yes	Yes
Insert	No	No	Yes	No	Yes
Delete	No	No	N/A	Yes	Yes
Update	Yes	No	Yes	No	Yes
Next Record	No	Yes	Yes	Yes	Yes
Associate	N/A	Yes	N/A	Yes	N/A

Table 2. Operation Restrictions Applied on Applets

Operation	Form	List	Pick	MVG	Association
Merge	N/A	No	N/A	N/A	No
Drilldown	N/A	No	N/A	N/A	N/A

If the EnableStandardMethods user property is declared on an applet, end users can use the applet to perform the record manipulation operations within a task view that they can within an ad hoc view, such as performing queries, advancing the record pointer, editing multiple records at once and so on.

NOTE: MergeRecords operations are not available in a list applet on a task view even if the EnableStandard Methods user property is enabled.

Configuring Task Views

This topic is organized as follows:

About Task Playbar Applets

The Task Playbar applet is the container for task navigation buttons. The applet can appear at the top or bottom (or both) of the task view. The Playbar applet is required for each task view, and can be added in two different ways:

- Create a task view using the Task View Wizard. The Task Playbar applet is associated with the task view through a rule enforced by the Task View wizard.
- Create a task view using the Task View Editor. You must manually add the Task Playbar applet using this method.

The Task Playbar applet has the following characteristics:

- Uses two special applet Web templates:
  - Applet Task Playbar - Bottom
  - Applet Task Playbar - Top

  The applet Web templates are for bottom and top placement on the screen respectively.
- The applet is of a frame class, CSSSWEFrameTaskPlaybar, which allows it to handle the buttons and to enable or disable some of the buttons depending on the business needs.

For more information about the task playbar, see "Task Playbar" on page 43.

About the Task View Templates

When configuring task views for your implementation of Task UI, you work with Web templates that are standard Siebel Web templates modified with IsInTask clauses. These templates control the behavior that is specific to task views. Task UI Web templates do not display the applet title or applet menu, nor do they display the screen menu, thread applet, thread field, or thread title. Each task view has a title in its top-left corner, which indicates to the user that the view is not an ad-hoc view.

Templates for Task Views

The Task UI application does not require specialized task view templates. However, the following templates are typically used for task views:

- View Detail (Parent with Pointer)

  You can use this template for your base task views. This template allows you to customize your task, and place applets side-by-side for a more structured view for the business user.

- View 1 Over 2 Over 1

  You can use this template when using shuttle applets in base task views.

Templates for the Playbar Applet

If you are creating specialized playbar applets, you copy, and modify these templates:

- CCAppletPlaybarButtons.swt
- CCAppletPlaybarBottom.swt
- CCAppletPlaybarTop.swt

For more information on configuring playbars, see "About Task Playbar Applets" on page 91.

Style Sheet for Task UI

You use the `main_task.css` style sheet to create a Task UI-specific look and feel. When you modify the Task UI style sheet, the changes apply globally to all your tasks.

Creating Task Views Using the Task View Wizard

This topic explains how to use the Task View wizard to create a task view.

Before you create a task view using the Task View wizard, you must first create the task applets, applets, and transient business components needed for the view.

To create a Task view using the wizard

1. Launch the Task View wizard:
   a. In Siebel Tools, choose New Object from the File menu.
   b. In the New Objects Wizards dialog, click the Task tab.
   c. Select Task View, and click OK to launch the Task View wizard.

2 In the New View screen, enter the project name, the view name, the view title, the business object of the view, and upgrade behavior.

3 Click Next to proceed to the View Web Layout - Select Layout screen.

4 Select a Web template for your view, and click Next to proceed to the Web Layout - Applets screen.

   This screen displays the applets associated with the specified business object.

5 Select the applets you want to add to the task view.

   NOTE: The applets you select on this screen are ad-hoc applets. You can select task applets in a subsequent screen of the wizard.

6 Click Next to proceed to the Task View - Pick Task screen.

   This screen allows you to specify whether you need a task applet in this view. Task applets are applets bound to a transient business component which stores transient task data. If you want to display a transient business component in this view, and have already created a task applet for this transient business component, click Yes. Otherwise, click No.

7 Make your selection and click Next:

   - If you have chosen not to use a task applet, go to Step 9.
   - If you have chosen to include task applets, you must specify the tasks that the task applets were created for.

8 Select the task applets you want to add to the task view, and click Next to proceed to the Task View - Select Playbar Applet screen.

9 Select the playbar applet you want to use for displaying the playbar at the top and the bottom of your view. If you need only one, leave one of the fields empty.

   NOTE: It is recommended that you use at least one playbar in task views.

10 Click Next to proceed to the Finish screen.

   This screen displays the parameters that will be used to create the new view. You can use the Back button to return to the previous screens and make changes, if necessary.

11 Click Finish to create the view.

   The newly created view is opened in the Web Layout Editor.

Editing Task Views Using the Web Layout Editor

The Web Layout Editor allows you to edit the mapping between applets in the view and placeholders in the template.

To edit a Task view using the Web Layout Editor

1 Open the Task view in the Web Layout Editor.

   - Right-click a Task view in the Views OBLE (in Siebel Tools), and choose Edit Web Layout from the pop-up menu.

2. Right-click the view displayed in the Web Layout Editor, and choose Preview from the pop-up menu.

   The view is displayed in preview mode, as it appears at run time.

3. Right-click and choose Preview again to return to editing mode.
4. Modify the applet sequence as necessary:
   a. In the Object Explorer, click the View Web Template object type.
   b. Select the appropriate row in the Web Templates OBLE.
   c. In the Object Explorer, click the View Web Template Item object type.
   d. Specify the appropriate sequence for each item using the Item Identifier property.
5. Modify the Applet Mode as necessary:
   a. In the Object Explorer, click the View Web Template object type.
   b. Select the appropriate row in the Web Templates OBLE.
   c. In the Object Explorer, click the View Web Template Item object type.
   d. Specify the appropriate applet mode for each item using the Applet Mode property.

For more information on using the Web Layout Editor, see the topic on configuring screens and views in *Configuring Siebel Business Applications*.

You can also modify the Web template file using an external editor, which you can define using the Options dialog box. For more information on defining the editor and editing the Web template file, see *Using Siebel Tools*.

Example of Configuring a Shuttle Applet in Task UI

This topic provides an example of how to configure a shuttle applet using a multi-value group (MVG) applet and an association applet within a task view. To configure the shuttle applet, perform the following tasks:

- "Creating the Association Applet" on page 94
- "Creating the MVG Applet" on page 95
- "Creating the Task View" on page 97

For more information about MVG applets, association applets, and shuttle applets, see *Configuring Siebel Business Applications*.

Creating the Association Applet

In a shuttle applet, the association applet appears on the left side of the view and contains the list of *available* records.

This task is a step in the "Example of Configuring a Shuttle Applet in Task UI" on page 94.

To create the association applet

1. From the File menu (in Siebel Tools), choose New Object.
2. Click the Applets tab, select MVG Applet, and click OK.
3. On the General page of the MVG Applet wizard, specify the following values:
   - Project — XTRM_LAB
   - Applet name — CC - Create Contact Access List Assoc
   - Display title — All Employees
   - Business component — Position
   - Upgrade behavior — Admin
4. Click Next.
5. For the Edit List mode, select Base Shuttle Assoc Applet EditList, and click Next.
6. On the Web Layout - Fields page, select the following fields to add to your applet:
   - First Name
   - Last Name
7. Click Next, then click Next again.
8. Click Finish to generate the applet.
9. In the Applets OBLE, select the applet you created, and modify the following user properties:
   - Class — CSSSWEFrameShuttleBaseAssoc
   - Type — Association List
10. In the Object Explorer, click the Applet User Prop object type.
11. Add the following Applet User Properties:
    - Name — CanInvokeMethod: AddRecords
      Value — [Active]
    - Name — EnableStandardMethods
      Value — Y
    - Name — High Interactivity Enabled
      Value — Y
12. Open the applet in the Web Layout Editor.
13. Delete the Query Assistant minibutton from the applet.
14. Save your changes.

Creating the MVG Applet

In a shuttle applet, the MVG applet appears on the right side of the view and contains the list of *selected* records.

This task is a step in the "Example of Configuring a Shuttle Applet in Task UI" on page 94.

To create the MVG applet

1. From the File menu (in Siebel Tools), choose New Object.
2. Click the Applets tab, select MVG Applet, and click OK.
3. On the General page of the MVG Applet wizard, specify the following values:
   - Project — XTRM_LAB
   - Applet name — CC - Create Contact Access List MVG
   - Display title — Team Members
   - Business component — Position
   - Upgrade behavior — Admin
4. Click Next.
5. For the Edit List mode, select Base Shuttle Mvg Applet EditList, and click Next.
6. On the Web Layout - Fields page, select the following fields to add to your applet:
   - SSA Primary Field
   - First Name
   - Last Name
7. Click Next, then click Next again.
8. Click Finish to generate the applet.
9. In the Applets OBLE, select the applet you created, and modify the following user property:
   - Class — CSSSWEFrameShuttleBaseMvg
   - Associate Applet — CC - Create Contact Access List Assoc
10. In the Object Explorer, click the Applet User Prop object type.
11. Add the following Applet User Properties:
    - Name — CanInvokeMethod: AddRecords
      Value — [Active]
    - Name — CanInvokeMethod: DeleteAllRecords
      Value — [Active]
    - Name — CanInvokeMethod: DeleteRecords
      Value — [Active]
    - Name — EnableStandardMethods
      Value — Y
    - Name — High Interactivity Enabled
      Value — Y
12. Open the applet in the Web Layout Editor.
13. Delete the Query Assistant minibutton from the applet.
14. From the Controls window, add the following controls to the applet (as shown in Figure 15):

- Place the Add Record, Remove Record, and Remove All Records minibuttons at the far left.
- Place PositionOnRow to the left of SSA Primary Field.

Figure 15. Layout of Sample MVG Applet

15 Save your changes.

Creating the Task View

In this shuttle applet, the task view contains the MVG applet and the association applet.

This task is a step in the "Example of Configuring a Shuttle Applet in Task UI" on page 94.

To create the task view

1 From the File menu (in Siebel Tools), choose New Object.

2 Click the Task tab, select Task View, and click OK.

3 On the New View page of the Task View wizard, specify the following values:

- Project — XTRM_LAB
- View name — CC - Contact Team Task View
- View title — CC - Contact Team Task View
- Business component — Contact
- Upgrade behavior — Admin

4 Click Next.

5 On the View Web Layout - Select Template page, select View 1 Over 2 Over 1, and click Next.

6 On the Web Layout - Applets page, select the following applets:

- CC - Create Contact Access List Assoc
- CC - Create Contact Access List MVG

NOTE: If the MVG applet and Assoc applets you created do not appear in the Available Applets list, perform the following steps:

a Click Cancel to close the wizard.

b In the Tasks OBLE, change the Type for both of the applets you created to Standard.

c Restart this procedure to create the task view.

d After you finish creating the task view, change the Type of the association applet to Association List, and change the Type of the MVG applet to MVG.

7 Click Next.

8 On the Task View - Pick Task page, select No, and click Next.

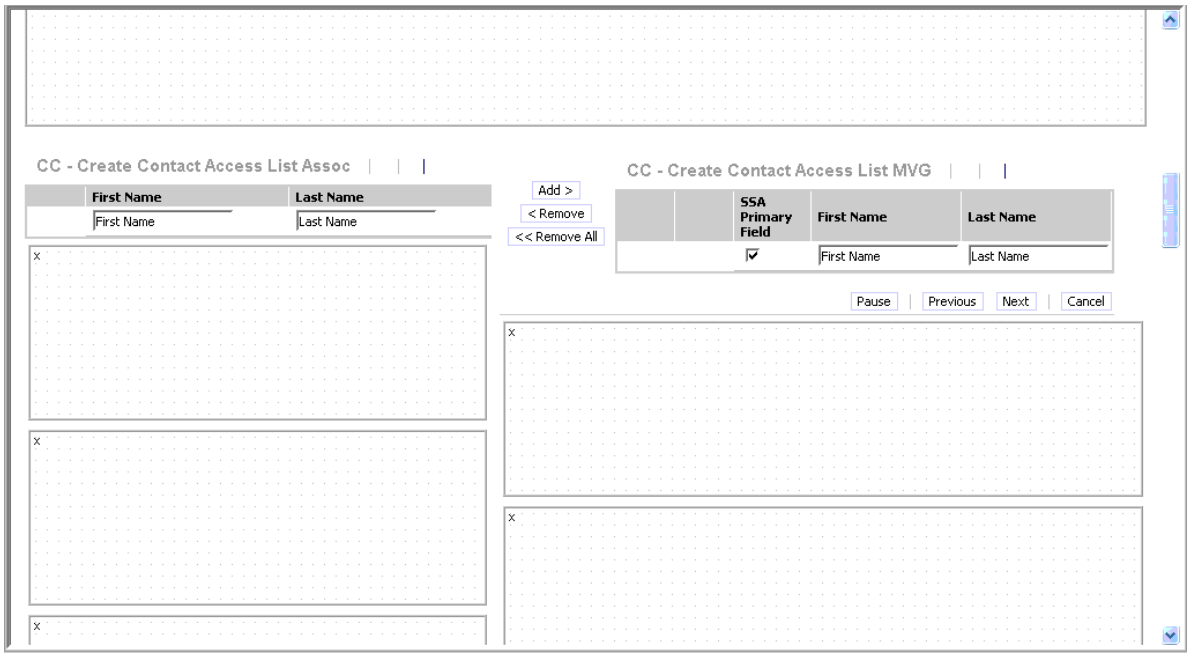
9 For the bottom playbar applet, select Task Playbar Applet - Bottom, and click Next.

10 Click Finish to generate the view.

11 In the Views OBLE, add the following View User Props:

- Name — ShuttleViewMvgAppletName
  Value — CC - Create Contact Access List MVG
- Name — ShuttleViewMvgField
  Value — Sales Rep
- Name — ShuttleViewParentBuscomp
  Value — Contact

12 Open the view in the Web Layout Editor, and modify the layout as necessary to look similar to the following:



13 Save your changes.

Configuring View Task Groups

The View Task Group object specifies the groups to display in the Task Pane when the current view is shown. A task group that is configured in the view task group of a view is a candidate to render on the task pane (in the specified view) if the Application property of the view task group is either null or matches the application currently running.

To configure a view task group

1. In the Object Explorer (in Siebel Tools), click the View object type.
2. In the Views OBLE, select the view to which you want to add a view task group.
3. In the Object Explorer, click the View Task Group object type.

   NOTE: If the View Task Group object is not visible in the Object Explorer, you can make it visible using the View > Options menu item. See *Using Siebel Tools* for more information.
4. For the selected view, add a new record to its View Task Group.
5. Set the appropriate application name in the Application property if you want that task group to show only for a certain application, or leave that field blank so the task group is shown in every application.

   The Sequence property can be used to arrange task group in ascending order.

Configuring Radio Button Groups

There are two different cases in which you might configure a radio button group:

- To create a new set of radio buttons based on a new LOV
- To create a new set of radio buttons based on an existing LOV

In the first case, the New Pick List wizard allows you to create the LOV picklist all at once. In the second case, because the LOV has been previously created, it can be reused.

NOTE: For a radio button group to be rendered in the client, the LOV on which the radio button group is based must exist in the run-time environment. When you deploy the application for testing or production, make sure you propagate the LOV to the target environment. For more information on deploying LOVs, see *Configuring Siebel Business Applications*.

To configure a radio button group

1. In Siebel Tools, create a field on the business component for the LOV radio button group.
2. In the Object Explorer, click Pick List.
3. Right-click the Picklists OBLE, and choose New Pick List Wizard from the pop-up menu.
4. Follow the prompts in the New Pick List wizard to establish the appropriate associations, create the Pick List, and select or create the LOV.
5. Verify that the Business Component property of the newly created picklist is set to PickList Generic.
6. Associate the business component field created in Step 1 with the picklist:
   a. In the Object Explorer, click Business Component.
   b. In the Business Components OBLE, select the appropriate business component.
   c. In the Object Explorer, click Field.

d In the Fields OBLE, select the appropriate field.

e For the Picklist property, select the newly created picklist.

7 In the Object Explorer, click Applet.

8 In the Applets OBLE, select the applet to which you want add the radio button group.

9 In the Object Explorer, click Control (under Applet).

10 Right-click the Controls OBLE, and choose New Record from the pop-up menu:

a Enter a Name for the control.

b For the Field property, select the field you created for the radio button group.

c Set the HTMLType property to RadioButton.

d Specify other properties for the control as appropriate.

Configuring the Default Value for Radio Button Groups

Setting a default value for a radio button group allows you to specify the likely user choice for most use cases, saving effort for the user.

NOTE: Setting the default value for a radio button group is optional. If you do set a default value, make sure to choose the default value carefully. The default value chosen should be the most likely user choice in most cases.

If you do not explicitly configure one value as the default, then the first value listed (alphabetically) in the LOV is the default value.

To configure the default value for a radio button group

1 In the Object Explorer (in Siebel Tools), click Business Component.

2 In the Business Components OBLE, select the business component with which the applet is associated.

3 In the Object Explorer, click Field (under Business Component).

4 In the Fields OBLE, select the field defined for the radio button group.

5 For the Predefault Value property, specify the default for the radio button group.

NOTE: Only one value of a radio button group can be the default.

Configuring Applet Messages

An applet message allows you to combine static text and dynamic data into a message to be displayed to the user. Applet messages can be used in task applets (based on a TBC), or in standard applets (based on standard BCs). The applet message configuration process is the same in both cases.

In Siebel Tools, the applet message is configured using the Applet Message and Applet Message Variable object types. You can place applet messages anywhere within an applet. Before you can position an applet message within an applet, you must create the definition of the applet message.

For more information about the Applet Message and other associated object types, see the *Siebel Object Types Reference*.

To create an applet message definition

1. In Siebel Tools, create a symbolic string reference containing the text of the applet message. Use %1, %2, and so on as placeholders for the dynamic data to be defined in the applet message variables.

   NOTE: To avoid run-time errors, make sure you use a strictly ascending sequence of numbers.

2. In the Object Explorer, click the Applet object type.
3. In the OBLE, select the Applet you want to add the applet message to.
4. In the Objects Explorer, click the Applet Message object type.

   NOTE: If the Applet Message object or its child objects are not visible in the Object Explorer, you can make them visible using the View > Options menu item.

5. Add an Applet Message object to the applet.

   NOTE: For details about adding objects, see *Using Siebel Tools*.

   a. In the Text Message property, select the symbolic string reference you created in Step 1.

   b. Fill in the other properties as appropriate.

6. In the Object Explorer, click the Applet Message Variable object type.
7. In the Applet Message Variables OBLE, create one record for each value that needs to be substituted in the applet message:

   a. For the Value property, enter the appropriate substitution number (for example, 1 for %1 substitution).

   b. For the Field property, choose the appropriate business component field whose value should be displayed for the placeholder in the message.

   c. Fill in the other properties as appropriate.

   For example, the following properties cause the substitution of the string %1 in the applet message with the value of the field named Opportunity Name:

   ```
   Value = 1

   Field = Opportunity Name
   ```

To configure the layout of an applet message

You can use the Applet OBLE to refine the layout of the task applet in which you want to place an applet message.

1. In the Object Explorer (in Siebel Tools), click the Applet object type.

2 In the Applets OBLE, select the Applet you want to add the applet message to.

3 In the Object Explorer, click the Controls object type, and create a new control for the newly created applet message:

   a Specify the name of the applet message as the Field property.

      The Field Type property is set to Message.

4 From the Dynamic Controls Palette, drag and drop the Static Dialog text control onto the applet grid layout in the position you want the applet message to appear.

   The applet message control is rendered on the grid.

5 (Optional) To specify the control width (for text wrapping), change the area size for this control.

   The associated text within the control wraps (in the edit and preview modes). At run time, the applet message includes the dynamic text where the placeholders sit, and the text wraps.

6 In the Properties window, set HTML Display Mode = FormatData.

   This allows you to put line returns and spacing within the string of text you are about to enter.

7 (Optional) If you have not entered text already for the message (when creating the referenced applet message), you can enter it now:

   a Double-click on the control.

   b In the text box within the layout editor, enter the message text.

   NOTE: Because applet messages use symbolic strings, they must always be written as stand-alone sentences of less than 250 characters in length.

Configuring the Ad-Hoc UI for Task Instantiation

This topic explains how to configure an ad-hoc (traditional) Siebel user interface to run tasks.

Configuring Task Groups

The following procedures explain how to configure a task group.

To add a task group

1 In the Object Explorer (in Siebel Tools), click the Task Group object type.

2 Right-click the Task Groups OBLE, and choose New Record from the pop-up menu.

3 Enter a Name for the task group.

4 For the Project property, select the project to which you want to add task group.

5 If this task group is to contain context-sensitive tasks, check the Require Context BC checkbox.

6 Now you can add task group items, as described in the next procedure.

To add task group items to a task group

1. In the Object Explorer (in Siebel Tools), click the Task Group object type.
2. In the Task Groups OBLE, and select the task group to which you want to add a task group item.
3. Right-click the Task Group Items OBLE, and choose New Record from the pop-up menu.
4. For the Type property, select Task.
5. For the Action Invoked property, select the task you want to invoke.
6. If you checked the Require Context BC checkbox when defining this task group, select a Context Business Component.

   The Context Business Component property of the task group item is used to associate a business component with a task. A task can require that the context business component be the business component of one of the applets in the current view in order to execute that task.
7. (Optional) You can specify a Sequence.

   The Sequence allows you to specify the order in which to arrange the tasks in the group.

Determining Visibility of a Task Group

The top-level Task Group object references tasks that can be launched when the business component is attached to an active and visible applet. In addition, the task must be defined as a task in the running Siebel application (for example, CallCenter). The display names for the tasks are sourced from the associated task objects.

Information in the Action pane is rendered based on the properties of the Task Step and Task Group objects.

NOTE: To make a task available for every view, add the task to a task group associated with the Task Pane View.

The tasks in the Context pane are rendered based on the following:

- **Association between view and task group.** If the task or a group of tasks belongs to a task group associated with the relevant view, it is rendered.
- **Application attribute.** If the Application attribute does not filter out the task, the task is rendered.
- **Require Context BC attribute.** This attribute indicates that the task group is to contain context-sensitive tasks. When a task group to which a task belongs is a candidate to render, the context business component can constrain the display of a task as follows:
  - If the task does not require any context business component, it is displayed.
  - If the task requires a context business component, it is shown only on the view in which the context business component is the business component one of the applets.
- **Responsibilities.** The end user must have the appropriate responsibility assigned in order to view the relevant task.

A task group, if used across multiple views, will have the same display name. This is the *group display name*.

NOTE: While a group display name is the same for all tasks in a task group, an individual task will have the same display name regardless of the task group with which it is associated.

Configuring Buttons for Task Instantiation

The following procedure explains how to configure buttons on applets that users can click to initiate a task (for example, a button labeled Update Contact Info in your ad-hoc My Contacts list applet). For more information about configuring buttons, see *Configuring Siebel Business Applications*.

To configure a button for task instantiation

1. In Siebel Tools, add a control to the applet.
2. For the HTML Type property, select MiniButton.
3. Specify a Display Name so it can be displayed.
4. For the Method Invoked property, enter LaunchTask.
5. For the Task Name property, enter the name of the task to be invoked.

Configuring Menus for Task Instantiation

The following procedure explains how to configure menus on applets to initiate tasks. For more information about configuring menus, see *Configuring Siebel Business Applications*.

To configure a menu for task instantiation

1. In Siebel Tools, create a new command:
   a. Provide a Name for the command that reflects the action it performs.
   b. For the Business Service property, select Task UI Service (SWE).
   c. For the Target property, select Server.
   d. For the Method property, enter LaunchTask.
   e. For the Method Argument property, enter the name of the task.

   NOTE: You must create a new command for each task, because the task to launch is specified in the properties of the command.
2. Add a menu item to the applet's menu that invokes the created command for the task.

Configuring iHelp Links for Task Instantiation

There are two types of links that you can use in the iHelp pane: an iHelp item (traditionally called iHelp task), and a Task UI task. To display a Task UI task link in the iHelp pane, the task must be associated with an iHelp item on iHelp pane. For information about configuring iHelp items, see the *Siebel Applications Administration Guide*.

To add a link for instantiating a task through iHelp

1. Configure an iHelp item in the Administration - iHelp screen using the procedures described in the *Siebel Applications Administration Guide*.
2. In Administration-iHelp all iHelp items, select an iHelp item to associate with the Task UI task.
3. Click Revise, and then click the More Info tab.
4. Select the appropriate task in the Related Task field.
5. Click Activate to activate the iHelp item.

   NOTE: If the Activate button is unavailable, check the Active flag in the Responsibility tab for the appropriate role to enable the Activate button.
6. Click the iHelp icon to see the link to the task displayed under its associated iHelp item.

Configuring Ad-Hoc Views for Task Transfer

This topic explains how to extend the user interface configuration for business objects that are not preconfigured to be multicall enabled. This configuration allows you to add a task view to a screen that is not preconfigured with one.

The process of configuring an ad-hoc view for task transfer consists of the follow procedures:

- “Create a New Link for Task Transfer” on page 105
- “Modify the Business Object for Task Transfer” on page 106
- “Create a New Ad-Hoc View for Task Transfer” on page 106
- “Modify the Screen for Task Transfer” on page 107

After you have configured the ad-hoc view for task transfer, you must compile the configuration changes to an SRF and deploy your tasks. You must also make sure that the newly created view can be accessed by the appropriate responsibilities.

Create a New Link for Task Transfer

This procedure explains how to create a new link to allow for task transfer. It is part of the process for “Configuring Ad-Hoc Views for Task Transfer” on page 105.

To create a new link to allow for task transfer

1. In the Object Explorer (in Siebel Tools), click the Link object type.
2. Right-click the Links OBLE, and choose New Record from the pop-up menu.
3. For the Parent Business Component property, select the business component for which you want to enable task transfer.
4. For the Child Business Component property, select UInbox Item Context.
5. For the Source Field property, select Id.

6 For the Destination Field property, select Item Context Id.

7 Set the No Update, No Delete, No Insert properties to TRUE.

8 For the Search Specification property, enter the following expression:

```
(LookupName('WF_INST_STAT_CD', [Task Status]) = 'PAUSED' OR
LookupName('WF_INST_STAT_CD', [Task Status]) = 'TRANSFERRED') AND [Item Context
Object Name] = 'object_name'
```

Substitute *object_name* with the name you used to associate your tasks with (as explained in topic "Associating Tasks with a Business Object Instance" on page 87). This name must match the string you used in the task definition when associating the task with the business object. However, it does not necessarily need to match the business object's name.

9 After you have defined the link, right-click the new link, and choose Validate from the pop-up menu to make sure there are no configuration errors.

Modify the Business Object for Task Transfer

This procedure explains how to modify the business object to allow for task transfer. It is part of the process for "Configuring Ad-Hoc Views for Task Transfer" on page 105.

To modify the business object to allow for task transfer

1 In the Object Explorer (in Siebel Tools), click the Business Object object type.

2 In the Business Objects OBLE, select the business object for which you want to enable task transfer.

3 In the Object Explorer, click the Business Object Component object type.

4 Right-click the Business Object Components OBLE, and choose New Record from the pop-up menu.

5 For the Bus Comp property, select UInbox Item Context.

6 For the Link property, select the newly created link.

7 After you have modified the business object, right-click the business object, and choose Validate from the pop-up menu to make sure there are no configuration errors.

Create a New Ad-Hoc View for Task Transfer

This procedure explains how to create a new ad-hoc view to allow for task transfer. It is part of the process for "Configuring Ad-Hoc Views for Task Transfer" on page 105.

To define an ad-hoc view to allow for task transfer

1 Launch the New View wizard:

a In Siebel Tools, choose New Object from the File menu.

b In the New Objects Wizards dialog, select View (on the General tab), and click OK to launch the New View wizard.

2 Follow the prompts to create the new view, using the following properties:

- **Business Object.** Indicates the business object modified for task transfer
- **View Name.** A descriptive name for the view (for example, *ObjectName* - Paused Tasks)
- **Upgrade Behavior.** Preserve (to preserve this modification during subsequent upgrades)
- **Web Template.** An appropriate master-detail template (for example, View Basic)
- **Master Applet.** An appropriate master applet based on the specified master business component.
- **Child Applet.** Task Item Context List Applet

3 After you have defined the view, edit the Web Layout as necessary.

4 Right-click the new view, and choose Validate from the pop-up menu to make sure there are no configuration errors.

Modify the Screen for Task Transfer

This procedure explains how to modify the screen to allow for task transfer. It is part of the process for "Configuring Ad-Hoc Views for Task Transfer" on page 105.

To modify the screen to allow for task transfer

1 In the Object Explorer (in Siebel Tools), click the Screen object type.

2 In the Screens OBLE, select the screen to which you want to add the task transfer view.

3 In the Object Explorer, click the Screen View object type.

4 Right-click the Screen Views OBLE, and choose New Record from the pop-up menu.

5 For the View property, select the view you created for task transfer.

6 Specify a Sequence to appropriately position this view on the site map.

7 For the Type property, select Detail View.

8 Select an appropriate value for the Parent Category property.

9 Specify an appropriate value for the Viewbar Text and Menu Text properties. For example, you can use the symbolic string SBL_TASKS-1004224752-2S0 for Tasks.

10 Set the Display In Page and Display In Site Map properties to TRUE.

11 For the Upgrade Behavior property, select Preserve to preserve this modification during subsequent upgrades.

12 After you have modified the screen, right-click the screen, and choose Validate from the pop-up menu to make sure there are no configuration errors.

Configuring Transient Business Components

This topic explains how to use the New Transient BusComp wizard to create a transient business component (TBC) and how to configure it after it is created.

Creating New Transient Business Components

A *transient business component* is a business component that can be used to house data to control task flow or logic, but without storing the data in the database. The data is cleared when a task is completed. TBCs are created in the same way as any other business components in the repository, except that they are always based on the S_TU_LOG table, and they are of BC type Transient.

To create a new Transient Business Component

1. In Siebel Tools, choose File > New Object.
2. In the New Objects Wizards dialog, select the Task tab.
3. Select the Transient BusComp icon, and click OK.
4. Use the wizard's dialog box to enter information, such as the new TBC's name, and the project to which the TBC is to belong.

   The wizard automatically assigns the TBC to the CSSBCTaskTransient class and the S_TU_LOG table, with a BC type of Transient.

NOTE: Although the recommended way to create a TBC is to use the wizard, if for some reason you must, you can create a TBC manually, without using the wizard. In Siebel Tools, use the OBLE as you normally would to create a regular business component. Column and Join properties are automatically calculated and read-only. Make sure to set the Class, Table, and Type fields as the wizard does, as described in Step 4. For more details, see the procedure that follows.

Creating New Transient Business Component Fields

You can create new TBC fields. You cannot create multi-value fields. Calculated fields are allowed, and no column is assigned to the field.

To create a new Transient BC field

1. Select the Transient Business Component record.
2. Create a new Field for the TBC, and enter values for the following:
   - Name
   - Type

- Text Length

Based on the Type and Text Length properties, the Column and Join properties are auto-populated (and are read-only).

NOTE: Type is a required attribute, while Text Length is not. As a best practice, provide a value for text length so that the Column property can be accurately determined. The fields of type DTYPE_TEXT with unspecified text length imply a length of 255 characters, so failure to specify the text length in cases where the real length is shorter might cause unnecessary joins to extension tables, or even failure to match all the fields to the corresponding columns.

Writing Scripts to Instantiate Tasks

You can use browser and server scripts to instantiate tasks. Your script calls the Task UI Service business service and invokes the method LaunchTaskFromScript to start a task, passing in the task name. The script must be compiled into proxy JavaScript objects or into the SRF.

Browser scripts. The InvokeMethods for scripts are first handled on the client-side objects, and then passed to their server-side equivalents. There is no CanInvokeMethod check.

Server scripts. If on the server-side, JavaScript files containing the scripted methods are compiled into the SRF, and then called from the OM at the appropriate pre- or postevent, there is no CanInvokeMethod check.

At run time, the Task UI Service checks the task name and checks that the user has the license and responsibility to run the task.

NOTE: Its important to understand that the ability to call tasks from scripts requires UI context. Do not invoke tasks from scripts that do not have UI context (for example, workflow scripts) or from events where the UI has not finished processing (for example, Applet_Load).

Sample Client-Side Script

In this example, the browser script looks for the task called *Create a Contact* and launches it.

```
function Applet_PreInvokeMethod (name, inputPropSet)
{

   try
   {

      if (name == "Test")
      {

         var inputPropSet;
         var outputPropSet;
         var taskUIsvc;

         inputPropSet = theApplication().NewPropertySet();

         outputPropSet = theApplication().NewPropertySet();

         taskUIsvc = theApplication().GetService("Task UI Service (SWE)");
```

```
            inputPropSet.SetProperty("TaskName","Create a Contact");

            taskUIsvc.InvokeMethod("LaunchTaskFromScript",inputPropSet,outputPropSet);

            return ("CancelOperation");

        }

    }

    catch(e)
    {

        theApplication().alert("Error" + e.toString());

    }

    finally
    {
    }

    return ("ContinueOperation");

}
```

Sample Server-Side Script

This example shows the server script approach to launch a task called *Create a Contact*.

```
function WebApplet_PreInvokeMethod (MethodName)
{

    try
    {

        if (MethodName == "Test")
        {

            var inputPropSet;
            var outputPropSet;
            var taskUIsvc;

            inputPropSet = TheApplication().NewPropertySet();

            outputPropSet = TheApplication().NewPropertySet();

            taskUIsvc = TheApplication().GetService("Task UI Service (SWE)");

            inputPropSet.SetProperty("TaskName","Create a Contact");

            taskUIsvc.InvokeMethod("LaunchTaskFromScript",inputPropSet,outputPropSet);

            return (CancelOperation);

        }

    }
```

```
    catch(e)
    {

        TheApplication().RaiseErrorText("Error" + e.toString());

    }

    finally
    {
    }

    return (ContinueOperation);

}
```

6 Deploying Tasks

This chapter discusses the procedures involved in deploying applications that employ Task UI. It includes the following topics:

After you have configured a task in Siebel Tools, you deploy the task by publishing and then activating it in the run-time client application.

About Task Deployment

A deployment scheme is used to move task objects from the repository to the run-time environment. After you have designed a task in Siebel Tools, you deploy the task by publishing it using Siebel Tools, and then activating it in the run-time client application. The processes for task design and task deployment occur separately. This topic contains further information about deploying tasks in the following topics:

For more detail on deploying tasks, see "Deploying Tasks from Siebel Tools" on page 115. For more information about deploying tasks for mobile clients, see "Setting Up Tasks for Running on Mobile Clients" on page 122.

About Deploying Actions Using Siebel Tools

You can use the buttons on the WF/Task Editor toolbar (shown in Figure 16) to perform deployment actions in Siebel Tools.

Figure 16. WF/Task Editor Toolbar

The buttons on the WF/Task Editor toolbar are (from left to right in Figure 16):

- **Publish/Activate.** Publishes and then activates the task.

  Use this button to activate a task only for debugging on a thick client. Because the activation is performed directly in the Siebel database from a Siebel Tools session, an application client that is currently running cannot see the activated task in its cache until you restart the client.

  To use the Publish/Activate button, you must set the VerCheckTime configuration parameter to -1 in the [Workflow] section of the application's .CFG file.

  ```
  ...
  [Workflow]
  VerCheckTime = -1
  ...
  ```

  NOTE: Do not use the Publish/Activate button when you are running or testing your task flows in a server environment. Instead, use the Publish button to publish the task flows, then go to the application client, and activate the task flows using the Activate button in the task deployment view.

- **Publish.** Publishes the task to the run-time environment.
- **Revise.** Creates a new version of the task.
- **Expire.** Sets the status of the task to expired.

For more information about the actions performed by these buttons, see "Deploying Tasks from Siebel Tools" on page 115.

To show the WF/Task Editor toolbar

- From the View menu (in Siebel Tools), choose Toolbars > WF/Task Editor Toolbar to display the WF/Task Editor toolbar.

Deploying Tasks Using Application Deployment Manager

You can use the Siebel Application Deployment Manager (ADM) to deploy the task. ADM is a tool that streamlines the process of deploying enterprise customization data (such as views, responsibilities, assignment rules, tasks, and Workflow processes and policies). It also allows you to migrate this data from one Siebel application environment to another.

A task deployment package in ADM includes the repository file (SRF), tasks, subtasks, and their run-time settings (such as activation and expiration times). For information on how to deploy and migrate tasks using ADM, see the *Siebel Application Deployment Manager Guide*.

Migrating Tasks Between Environments

After you have configured a task in your development environment, you need to move it to a testing or production environment. As with the standard Siebel applications, you can use one of the following three ways to accomplish this migration:

- **Archive Files.** Using the features of Siebel Tools, you can export objects from the repository to an archive file (SIF), and then import objects from the archive file back into the repository using Siebel Tools. Use archive files when you want to back up sets of objects or migrate these objects. You can include individual objects (such as business components, applets, views, and tasks) or entire projects in archive files. For more information on how to export and import archive files, see *Using Siebel Tools*.
- **REPIMEXP.** You can use the Repository Import and Export (REPIMEXP) utility for bulk migration of repository objects, including task definitions. REPIMEXP is a command-line utility found in the BIN directory of your Siebel installation. Type `repimexp /help` in a command prompt to view your usage options.

  NOTE: If you use REPIMEXP you cannot pick and choose which tasks to migrate. To select a single task or only certain tasks for migration, use the ADM migration option.
- **ADM.** The Siebel Application Deployment Manager (ADM) is a tool that streamlines the process of deploying and migrating enterprise customization data, including tasks, from one Siebel application environment to another. This process includes migrating from a development environment to a testing environment, and from a testing (or development) environment to a production environment. For information on how to migrate tasks using ADM, see the *Siebel Application Deployment Manager Guide*.

NOTE: Before migrating data, make sure that all data required for the task is also present in the target environment. For example, if your task requires custom entries in the list of values (LOV) table, make sure that these are present and active.

Deploying Tasks from Siebel Tools

Deploying a task requires two steps: publishing and activating. The task definitions (stored as repository objects) must be published to the run-time environment, and then they must be activated in the run-time environment. The process for deploying a task involves the following procedures:

1. Publish the task from Siebel Tools.

   This makes the task visible in the run-time client application. For more information, see "Publishing Tasks" on page 116.

2. Activate the task from the run-time client.

   This makes the task available to users. For more information, see "Activating Tasks" on page 116.

Then, after you publish and activate the task, you can assign responsibilities for running the task and the access rights they have on the inbox. For more information, see "Controlling Task Access Privileges" on page 119.

Publishing Tasks

Because the task definitions are stored in the repository, you must compile the repository before publishing when you have added repository objects (such as business components, business services, and views) to the task.

NOTE: When you publish tasks that contain subtasks, make sure to publish the subtasks prior to publishing the main task so that the subtasks are available to the main task in the run-time environment.

When you publish a task, the task definition is read from the repository and written to the run-time environment (with its deployment parameters) in XML format. The task's deployment parameters (Replication, Activation Date, Expiration Date) are used when activating the task.

To publish a task from Siebel Tools

1. In the Object Explorer (in Siebel Tools), click the Task object type.
2. In the Tasks OBLE, select the task you want to publish.
3. On the WF/Task Editor toolbar, click the Publish button.

   NOTE: If the WF/Task Editor toolbar is not currently displayed, choose Toolbars > WF/Task Editor Toolbar from the View menu.

   When you click Publish, you are prompted to either validate before you publish or to go ahead.
4. Click Yes to go ahead and publish the task.

   When the status changes from In Progress to Completed, the task becomes visible in run-time client application.

After the task has been published, it must be activated before it can be used in the run-time environment.

NOTE: If you use the Publish/Activate button (rather than the Publish button) to publish the task, there is no need to activate the task separately in the run-time application.

Activating Tasks

After the task has been published, it must be activated before it can be used in the run-time environment. You specify the deployment status of the published task definitions using the Repository Task Deployment view in the run-time client. This view consists of the following two applets:

- **Published Tasks List Applet**. Displays the completed tasks from the repository tables.

  NOTE: When a task is revised and published, the repository version of this task is incremented. When you click on the task name link, the applet displays the child items (published task definitions) of the task.
- **Task Deployment Repository List Applet**. Displays the tasks that have been deployed.

In the run-time environment, the deployment status of a deployed task can be Active, Outdated, or Inactive:

- A task that is active is the basis for new tasks that are created. New instances of the task are created using this definition. At any given time, there can only be one active version of a task with a particular name.
- A task that is outdated allows tasks that are already actively in progress or in the inbox to continue using the prior definition, but new task instances are not based on this definition. When a task definition is outdated, no new instance can be created based on that definition, but existing tasks continue using that version.
- If a task is inactive, then no new or paused instances can continue using this definition.

To activate a task in the run-time environment

1. Launch the run-time client, and log in.
2. From the application-level menu in the run-time client, choose Navigate > Site Map > Administration - Business Process > Task Deployment.

   This displays the Task Deployment view where you can activate the task.
3. Query for the task you just published.
4. Select the task, and then click Activate.

   The syntax is checked for validity, and the deployment status of the task changes to Active.

   NOTE: When you activate a task, the deployment status of the previous active version (if any) changes to Outdated.
5. Specify the deployment parameters for the task in the Task Deployment Repository List Applet:
   a. Set the activation date in the Activation Date/Time field.
   b. Set the expiration date in the Expiration Date/Time field.
   c. Set the replication to None, unless you are deploying the Task to mobile clients.

      NOTE: If you are deploying the Task to mobile clients, see "Setting Up Tasks for Running on Mobile Clients" on page 122.

After you have activated the task, you must now assign the responsibilities for running the task and the access rights they have on the inbox. For more information, see "Controlling Task Access Privileges" on page 119.

To deactivate a task in the run-time environment

1. Launch the run-time client, and log in.
2. From the application-level menu in the run-time client, choose Navigate > Site Map > Administration - Business Process > Task Deployment.

   This displays the Task Deployment view,
3. Query for the active or outdated task definition you want to deactivate (in the lower applet for Active Tasks).

4 Select the task, and then click Deactivate Task.

The deployment status of this task changes to Inactive.

To delete a task in the run-time environment

1 Launch the run-time client, and log in.

2 From the application-level menu in the run-time client, choose Navigate > Site Map > Administration - Business Process > Task Deployment.

This displays the Task Deployment view,

3 Query for the active, inactive, or outdated task definition you want to delete (in the lower applet for Active Tasks).

4 Click to select the task, and then click Delete Task.

CAUTION: Deleting a deployed definition impacts all running instances of that definition and might corrupt the data. It is recommended that you do not delete a deployed task definition unless the task definition has been deactivated.

7 Administering Task UI

This chapter discusses the activities an administrator can perform to manage tasks, control access and transfer of tasks, and allow for the running of tasks on mobile clients. It includes the following topics:

Controlling Task Access Privileges

The Registered Task Administration view allows the application administrator to control user ability to execute, transfer, and delete tasks. For a task to be successfully executed by a user, the user must be added to a responsibility that has the privilege to execute the task.

NOTE: The task execution privilege subsumes all view access privileges while the task is being executed. A user that has the task execution privilege also has all the access privileges associated with the views within the task.

You administer task access control in a way that is similar to how view access control is administered. The following procedure describes the administrative steps to add task access privileges to a responsibility.

To administer a task and configure its access privileges

1. Make sure that the responsibilities for the task are already defined.

   NOTE: See the *Siebel Security Guide* for information on how to define responsibilities for Siebel applications.

2. Register the task. Before you can add task access privileges to a responsibility, you must first register the task so that it can be administered in the Task Administration view:

   a. Navigate to Site Map > Administration - Application > Tasks.

      The Registered Tasks applet shows all registered tasks.

   b. Create a new record in the Registered Task applet.

   c. Select the task you want to register from the picklist in the Task Name field.

      NOTE: The picklist displays only deployed (published and activated) tasks.

3. Associate the task with responsibilities:

a Create a new record in the Responsibilities list applet.

b Query the Siebel Administration responsibility, and set Allow Transfer and Allow Delete based on your requirements. Both of these are set to TRUE by default.

4 License tasks and task views.

Tasks must be licensed in order to be invoked. Task views also must be licensed, and they must belong to screens under the current application for the task views to visible.

5 Add the user to the responsibilities, if needed:

a Navigate to a different Task Administration View: Site Map > Administration - Application > Responsibilities > Tasks.

b In the Responsibility list applet, query for the responsibilities that you have associated the task with.

In the lower half of the view, there is a Task List applet and a Users list applet.

c For the selected responsibility, check if the task and the users are already added to the Task List applet and the User List applet. If not, click the New button to add them.

Controlling Task Transfer Privileges

By default, tasks are transferable. However, you can alter this behavior using the Responsibilities task list applet. If the ability to reassign is not available to the end user on the current task item (in the Inbox Item List view), the owner field is read-only and the Transfer button is disabled.

To set up a task to allow transfer

1 Navigate to Site Map > Administration - Application > Tasks, and select the task that needs to be transferred.

2 Make sure that the Allow Transfer option is checked. If necessary, check the box.

3 Step off the record to save your changes.

Transferring Tasks in the Universal Inbox

A task that has been created or paused can be transferred in from one inbox to another. A task can be transferred by the owner using the Universal Inbox.

When a user clicks the Transfer button in the Inbox Item List, a pop-up applet is launched to display a read-write version of the Owner field and a text area control for entering a transfer reason. The end user can pick a different owner and add comments explaining why the task is being transferred. When the user clicks OK, the change is saved and the pop-up applet closes. The task then appears in the Inbox of the new owner.

NOTE: The list of users to which a task can be transferred is constrained to those that have responsibilities for the task.

You can specify whether the reason for the transfer must be provided by the user by setting a flag on the intersection between a task and a responsibility.

To transfer a task

1. Navigate to Site Map > Inbox > Inbox Items, and select the task you want to transfer.
2. Verify that the Transfer button is enabled.

   If the Transfer button is disabled, the task is not transferable. For information about how to make a task transferable, see “Controlling Task Transfer Privileges” on page 120.
3. Click Transfer.

   The Transfer To dialog box is displayed.
4. Select the owner to transfer the Task to.
5. Enter a reason for transfer in the Comments text field.
6. Click OK.

   After the Transfer is completed, the Completed flag is checked in the Inbox List applet.
7. Refresh the screen.

   The item is removed from My Inbox Items and added to My Completed Items. The task then appears in the Inbox of the new owner.

Deleting Tasks

A task can be deleted from the Inbox. For the task to be deleted, certain options must be specified on the task. If these options are not set, the task cannot be deleted.

To set up a task to allow deletion

1. Navigate to Site Map > Administration - Application > Tasks, and select the task you want to delete.
2. Make sure that the Allow Delete option is checked. If necessary, check the box.
3. Step off the record to save your changes.

To delete a task

1. Navigate to Site Map > Inbox > Inbox Items, and select the task in the Inbox that you want to delete.
2. Click on the Detail tab, and click Delete.

   The task item is removed from the Inbox.

Setting Up Tasks for Running on Mobile Clients

This topic describes how tasks can be set up to run on mobile clients, which can be running against regional databases or local databases.

Setting Inbox Visibility for Mobile Clients

The Inbox screen is visible in mobile clients out of the box. All Task-type inbox items created on the server are replicated to mobile clients, and vice versa. This behavior makes sure that end users can always see a complete list of the task inbox items that they need to act upon, regardless of their work location.

Replicating Tasks to Mobile Clients

Repository definitions of tasks are not replicated to mobile clients. As a result, mobile users cannot view and activate task definitions from mobile clients.

The deployment records of tasks must be replicated to mobile clients before the tasks can be run on mobile clients.

To enable the replication of task deployment records from the server

1. Navigate to Site Map > Administration - Workflow Process > Task Deployment > Published Task.
2. In the Active Tasks applet, query for the task deployment record you want to replicate (for example, by searching on the task name).
3. Change the Replication field of the task deployment record to Regional for mobile clients connecting only to regional databases, or to All for mobile clients connecting to *either* regional *or* local databases.

Task run-time instances—the underlying object of Task-type inbox items—are not replicated between the server and mobile clients. As a result, Task-type inbox items created at a node (such as the server) can be run only from the same node (that is, the server). An error is returned if an attempt is made to run Task-type inbox items from a wrong node.

For the same reason, task transfers—that is, the reassignment of Task-type inbox items—are allowed only between users in the same node. In other words, Task-type inbox items created at the server/regional node can be transferred to other users in the same server/regional node. Task-type inbox items created on mobile clients connecting to local databases cannot be transferred.

Setting the Synchronization Interval

For information on how to set the synchronization interval for tasks replicated to mobile clients, see the topic on managing synchronization frequency in *Siebel Remote and Replication Manager Administration Guide*.

8 Debugging Tasks

This chapter discusses the Task Debugger, as well as ways of troubleshooting your implementation of Task UI. It includes the following topics:

Validating Task Configuration

Validation enforces the semantic consistency of a task that cannot be easily enforced by structural constraints. For example, using validation, you can make sure that an error process does not contain an error process. When you validate a task, you are given warnings about errors the task may contain. You can then correct the errors before publishing the task.

The Validate tool can detect the following errors:

- Connectors not attached correctly. Make sure the task diagram's branches are connected correctly.
- Outgoing branches not specified for decision points. Make sure to specify outgoing branches for each decision point in the task.
- Business services and business service methods not specified for business service steps. Make sure that each business service step in the task is not missing a business service or a business service method.
- Business components missing from Siebel Operation steps. Make sure to specify the business component upon which each Siebel Operation step acts.
- Subtask Name not specified for subtask steps. Make sure that each subtask step specifies the appropriate subtask name that the task flow calls.

To validate a task configuration

1. In the Tasks OBLE (in Siebel Tools), right-click the task you want to validate and choose Validate from the pop-up menu.

   The Validate dialog box appears.

2 Click Start.

Starting validation... appears in the bottom left corner of the Validate dialog box.

If the validation is successful, there are no errors to report; and *Total tests failed: 0* appears in the bottom left corner of the Validate dialog box.

Enabling Debug Mode

To run the task debugger from the application client, you must first edit your configuration file to enable the Debug Mode menu item. Debug Mode is a restricted menu item.

The EnableRestrictedMenu parameter (in the [InfraUIFramework] section of the configuration file) controls the display of the Debug Mode menu item.

If this parameter is set to TRUE, then users of that application can see the Debug Mode item on the Tools application-level menu. If this parameter is set to FALSE, the Debug Mode menu item is not visible to users.

To enable the Debug Mode menu item

1 Open the configuration file (.CFG) for the application in a text editor.

2 In the [InfraUIFramework] section of the configuration file, set the EnableRestrictedMenu parameter to TRUE.

For example:

```
[InfraUIFramework]

EnableRestrictedMenu = TRUE
```

3 Save your changes, and close the file.

NOTE: If the EnableRestrictedMenu parameter is set to anything other than TRUE or FALSE, or if it is not defined under [InfraUIFramework], the Debug Mode menu item is not visible to a user of the application unless the user has access to the Admin - Restricted Menu Items view.

Using Task Debugger

You use the Task Debugger to test a task's performance before deploying it. You work with the Task Debugger behaving as the end user behaves. To debug tasks, you use the Debug Mode in the Siebel client, reading watch windows while moving through the task flow.

You can turn the Debug Mode on or off. You can do this before, during, or after a task.

To debug a task

1 Verify that the Debug Mode menu item is enabled in your configuration file (see "Enabling Debug Mode" on page 126).

2 Launch the Siebel application client.

3 In the Siebel client, from the application-level menu, choose Tools > Debug Mode.

4 Click the Open Task Pane icon (in the same row of icons as the Site Map icon) to open the Task Pane.

5 From the Task pane, drill down on the task to be debugged.

The task is launched, and the first step in the task (the Start step) is executed.

The Task Properties watch window appears for the selected task, showing the properties for the last-executed step, which in this case is the Start step. Under the name of the task being debugged, there is a line titled Last Action, which indicates the last navigation action (for example, Next).

6 Click the Continue button in the watch window.

The next step in the task is executed. If the next step is a task view step, a task view is displayed.

7 Click one of the playbar buttons in the task view to continue task execution and debugging.

8 Repeat Step 6 and Step 7 to proceed through the debugging of the task.

Debugging terminates when the End step is reached.

9 (Optional) To terminate a debugging session during task execution, you can click Stop in the watch window, or click Cancel in a task view.

Using Server-Side Logging

Event logging is a convenient, nonintrusive way for developers to trace the components that need further examination. Similar to Workflow, Task UI provides for tracing through event logging.

Setting Tracing with Event Log Level

The log event reports different operations during execution of a task and prints certain data structure for analysis. Table 3 shows the Task UI-related event types in the Event Configuration View, along with the events being traced and the information symbols used in the log files for identifying the events. In this table, the Log Level column indicates the minimum level at which tracing is enabled.

NOTE: Task UI shares some event symbols with the Siebel Workflow engine.

CAUTION: Setting trace levels above the default parameters impacts performance. Trace levels should be reset to default parameters after troubleshooting has been completed.

Table 3. Task UI Logging Events

Event Type	Log Level	Event Traced	Event Symbol	Information Symbol and Description
Workflow Definition Loading	3	Steps and branch names loaded.	DfnLoad	Step — task definition steps and branch names.
	5	Chapters loaded.	DfnLoad	Chapter — task definition chapter names.

Table 3. Task UI Logging Events

Event Type	Log Level	Event Traced	Event Symbol	Information Symbol and Description
Task Execution	3	Information about various user operations performed on the task.	TskExec	Oper — information about various user operations during lifetime of the task (such as creating, deleting, pausing, restoring, instantiating, and cancelling a task).
	5	Further internal details for these operations.	TskExec	TskState — state transition details during task lifetime. TskInbox — more internal details on the task interaction with Universal Inbox.
Task Navigation	3	All navigation operations on the task.	TskNav	Oper — information on how a task navigates from one view to another view.
	4	All changes to navigation stacks.	TskNav	NavPath — changes to backward and forward navigation stacks are logged. When a new frame is inserted or deleted from backward or forward stacks, it is logged.
	5	Contents of navigation stacks.	TskNav	DumpStack — dumps all frames in the forward and backward navigation stacks after restoring/resuming a task.
Workflow Process Execution	4	Instance creation.	PrcExec	Create — information on new task instance creation.

Table 3. Task UI Logging Events

Event Type	Log Level	Event Traced	Event Symbol	Information Symbol and Description
Workflow Step Execution	4	Task step executed and input or output argument of the task step.	StpExec	Cond — information on branch condition evaluation. Create — information on step instance creation. End — information on step instance completion. Task — information on business service invocation. TaskArg — arguments to a business service.
Task Presentation Information	4	Presentation view information presented to an external component (such as SWE) from the task controller.	TskPresInfo	ViewInfo — entire task view info structure, including view name, playbar information, current task pane information, and all records pending validation.

Increasing the Trace Levels for Workflow Management Server Components

You can troubleshoot issues in your application using increased trace levels on server components.

To increase tracing levels

1. Launch the client application.
2. From the application-level menu, choose Navigate > Site Map > Administration - Server Configuration > Servers > Components > Events.
3. In the Components applet, choose the component for which you want to generate tracing (Call Center Object Manager, Custom Application Object Manager).
4. Click the Events tab to view the configurable event types for the selected component.

   The log level is set to a default value of 1.
5. Change the log level value to 3, 4, or 5. Use Table 3 to make your choice.

   For additional troubleshooting of the Object Manager Shadow Store, you can increase the tracing level for the following events:

   - Task UI Object Manager Log
   - Task UI Conflict Log

- Task UI Union Sql Log

NOTE: More tracing information is generated as the Component Event Configuration Log Level value increases. For more information on the different Log Level values available for Component Event Configuration, see the *Siebel System Monitoring and Diagnostics Guide*.

Enabling Client-Side Logging

You can set the log level for Siebel thick clients by setting the environment variable SIEBEL_LOG_EVENTS. For example, to set the logging level for TskNav and TskExec to 3 and the level for StpExec and TskPresInfo to 4, execute the following command (in a command prompt):

```
set SIEBEL_LOG_EVENTS = TskNav =3, TskExec=3, StpExec=4, TskPresInfo=4
```

Example of a log file:

The following is an example of a log file with TskExec, TskNav, and TskPresInfo log events enabled.

```
TskExec     TskState   3  0000000243fc1350:02006-02-22 23:49:49Task state transition changes : Action
invoked: 'Navigate', Current State: 'Navigate', Next State: 'Navigate'.

TskNav      Oper       3  0000000243fc1350:02006-02-22 23:49:49Task engine requested to navigate to next
view.

TskNav      Oper       3  0000000243fc1350:02006-02-22 23:49:49Task engine requested to navigate to next
step: 'Task View 1'.

TskNav      PathChange 3  0000000243fc1350:02006-02-22 23:49:49Pushing frame to stack :
'1*05*Start0*1*00*'.

TskNav      PathChange 3  0000000243fc1350:02006-02-22 23:49:49Pushing frame to stack : '1*011*Task View
03011*8#Bookmark4#8#viewName21#EnvironmentDetailView6#bAdmin1#06#viewId0#14#bIgnoreContext1#03#18#frameBo
okmarkArray18#frameBookmarkArray1#4#size1#33#1#21#210#6#rowIds0#16#extraInformation0#14#ToggleSequence2#-
111#columnNames0#11#InQueryMode1#013#SavedShowMode4#Edit9#frameName17#EnvironmentDetail16#RuntimeClassNam
e19#CSSSWEFrameBookmark8#ShowMode4#Edit8#UniqueID1#30#1#11#110#6#rowIds0#16#extraInformation0#14#ToggleSe
quence2#-111#columnNames0#11#InQueryMode1#013#SavedShowMode4#Base9#frameName28#Task Playbar Applet -

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (0) : TYPE =
WfTaskViewInfo

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (0) : ViewName
= RequiredFieldView

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (0) : child
count = 0TskPresInfoViewInfo0000000243fc1350:02006-02-22 23:49:49Task presentation view info (0) : child
count = 0

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : TYPE =
WfTaskPlaybarInfo

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : Submit =
HIDDEN

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : Next =
ENABLED

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : Prev =
ENABLED

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : Pause =
ENABLED

TskPresInfo ViewInfo   3  0000000243fc1350:02006-02-22 23:49:49Task presentation view info (1) : Cancel =
ENABLED
```

Disabling Task Transactions

There are two ways to disable the task transaction mechanism:

- **Disabling task transactions at the business component level.** For a business component, you can set the Immediate Commit in Task user property to turn a task transaction on or off. If its value is set to TRUE, then the task transaction is off, and updates on this business component within any task are immediately committed to the base tables.
- **Disabling task transactions at the task level.** You can set the Transactional task property to turn a task transaction on or off for a task definition. If Transactional is set to FALSE, then the task transaction is off, and updates on business components within the task are immediately committed to the base tables.

  CAUTION: Disabling task transactions can result in unpredictable behaviors and should be thoroughly tested before use in production.

NOTE: For a business component to be part of the task transaction, it must be enabled at both the business component and task levels.

Troubleshooting Common Problems

This topic discusses the following issues:

Task Does Not Show in the Task Pane

If your task fails to appear in the Task Pane, make sure that:

- All of the objects associated with your task (task applets, transient BCs, and so on) are compiled into the SRF.
- The task is deployed and activated.
- The task is associated with a task group.
- The parent task group is associated with a view.
- The task is added to the Administration - Application > Tasks view.
- The task is accessible to your responsibility (Administration - Application > Tasks).

Record Context Is Lost

While navigating between views, if your record context is lost even though there is no Query operation, verify the following:

- The applet search specifications between the two views match.
- The view step search specifications match.

If the first view has an applet or a view search spec, and the second one does not, and if *Retain Applet Search Spec*/*Retain Step Search Spec* is set to FALSE, there will be a forced re-execution; this causes loss of record context, because the second view is supposed to show the business component data without the constraints of the first view's search spec.

Consider your business logic. You may want to set the attributes *Retain Applet Search Spec* or *Retain Step Search Spec* to TRUE.

It is recommended that you set search spec constraints on the business component or on the view step, rather than on the applet. This way, the scope of the search spec is more clearly defined. Business component search specs apply throughout the task, and view step search specs apply for just the step (that is, until the next UI step, depending on the value of *Retain Step Search Spec*).

If both steps contain an identical applet, have identical step search specs, and *Retain Applet Search Spec*/*Retain Step Search Spec* is TRUE), and you are seeing re-execution, there may be specialized code that is triggering the execution. To debug, use logging with the ObjMgrSqlLog event set to 5 to inspect search specs for each SQL statement. Review your log for the CSSSqlObj::InvalidateSQLCursor call. Check to see if there are any errors triggering the execution of this call.

Task Build View Errors

In Task UI mode, a build view error may occur during navigation in the following two scenarios:

- The previous view is not destroyed yet so there is still an active view. In this case, a modal pop-up message is shown to user.

  You can dismiss the message, correct the problem, and try to navigate or cancel the task.
- The previous view has been destroyed and, therefore, no active view is available. In this case, an HTML page indicating the error is displayed to user.

Other Errors

The following errors can occur during the development of applications that use Task UI.

I created a TBC with a text field. But during execution the field is not displayed properly. Change the property of the field to Text. Default is RadioButton.

A TBC I created is not found in the business components list in the steps of the task flow. Add the TBC in the corresponding business object (Business Objects OBLE > Business Object Component).

A Task applet is not displayed while I am creating a view. Select the appropriate task flow, which is linked with the applet.

Task step with forward button type of Submit. Select a view step and change the type to Submit.

Task flow not listed in Application – Administration > Tasks. Publish and activate the task flow.

Missing TBC during task flow execution. Add the TBC in the corresponding business object (Business Objects OBLE > Business Object Component).

User-defined views or applets not found.

Verify that the correct views or applets are registered in Application – Administration > Views.

Verify that the views and applets have been compiled into the SRF.

Make sure the views or applets are added to the user with appropriate responsibility.

Task group name is empty in the task pane. Create a display name and update the display name in the task group.

Task group not found in the task pane. Make sure the task group is added to the ad-hoc UI.

Task flow not displayed in the task group. Add the task flow to the corresponding task group and compile.

Task group or task flow name not found in the ad-hoc UI. Compile the ad-hoc UI, which contains the above objects.

9 Best Practices for Working with Tasks

This chapter provides best practices for selected topics that it is recommended you consider when implementing Task UI. It includes the following topics:

Best Practices for User Experience of Tasks

This topic includes the following:

Best Practices for Designing a Task's Flow

When designing a task-based UI, envision the presentation of information in a series of linked pages, or steps. Although in many cases, a task step can be contained in a single page, this need not always be the case. If the step is complex, break it into two pages. The overriding principle of Task UI design is simplicity, both in content and the presentation of content.

Keep the following factors in mind while designing your task flows:

- **Task frequency.** How often will the user be performing the task?
- **Task complexity.** Make sure each task is self contained, a single point-to-point flow, not a collection of tasks contained in one task. Be mindful during the design phase of the number or steps or decisions, and if necessary break one task into two.
- **User experience.** Be mindful of not only the end user's experience, but of user's job role itself, and consider questions such as job turnover, and computer aptitude.
- **Task size.** Can the task be broken down into a number of logical chunks? The user should be taken through one task flow, easy to follow and review. If the task gets too long, you may lose the user.
- **Enforce forward navigation allowing task review and editing.** Design your tasks in such a way that your end user is guided forward through a task in a manner that allows editing and review of completed portions. Navigation that requires end-user use of the Previous button is acceptable, but the end user should not have to repeatedly click Previous back through a long list of views to correct data entered earlier.

- **Organize flows in logical sections.**
- **Organize flows around logical task transaction commit points.**
- **Use the Chapters feature.** Chapters represent a descriptive unit of the work covered by the task. The goal of chapters is to allow you to more easily guide a user through a process and supplement user inputs with additional procedural automations.

  Chapters also provide an outline of the task to the end user. This helps the end user understand what will be done in the task and how much is left to do. The individual steps within a chapter may vary depending on choices made by the user. However, the outline provided by chapters does not change. For more information on chapters, "Task Chapter" on page 40.
- **Allow users to enter record data across multiple views.**
- **Allow users to enter multiple records.** In general:
  - **Use of form views.** Looping over a form view is preferred if there is a lot of data being entered for each record.
  - **Use of list views.** Using a list view is preferred if the number of fields needed for each record is small and fits well on the screen without horizontal scrolling of the list view.
- **Allow user specified queries.** Putting query and results in separate views vs. using the same view for both query and result display.
- **Use the same Task View in multiple view steps within a task.**
- **Choose a step display style for the Current Task pane.** The choice is between displaying every step to the end user, or displaying a subset of steps that provides an outline of the task.
- **Avoid hyperlinks within tasks.** Remain within the task flow. Avoid scripts or run-time events that would break out of the task flow, putting the task on Pause.
- **Include a Review page.** A review page (or summary view) allows the user to review and change any data before it is committed. A review page can be at the end of a task, or in the midst of one, if the task flow it complex.

Best Practices for Designing a Task View

When designing a new task view, present only data specific to the particular task, maintaining simplicity of instruction as well as UI presentation. The Current Task pane provides feedback to the user about their progress within a task. Keep this feedback succinct and make sure it clearly relates to successful performance of the task at hand. When designing a page, start with its type (Overview, Work, Review, Summary), then with its title, and then proceed to fulfill the task in accordance with these two principles.

The following are common issues and concerns when approaching Task UI.

- Avoid the following common ad-hoc UI design styles:

- Drilldowns.

  For example, consider the Contact screen in the ad-hoc UI. The view displays a list of contacts. One item in each contact is the Account the contact is associated with. The Account Name is what is displayed; clicking on Account Name moves the end user to the Account screen displaying the selected account. This is drilldown functionality.

  Drilldowns are not permitted in Task UI. This is because using the drilldown requires pausing the task so that the ad-hoc UI view targeted by the drilldown can be reached.

- Buttons on applets.
- List applets that require horizontal scroll bars.
- Too many applets in a view. Keep the design of each view as simple as possible.

- Choosing between Form and List applets.

  - **List applets.** Should be used for list and summary views.
  - **Form applets.** Should be used when the user needs to manipulate the displayed record.

  Form applets are generally preferred in Task UI because they focus the end user's attention on a single record and the work that needs to be done with it.

  List views are often used in the ad-hoc UI whenever a new record needs to be created. In the ad-hoc UI, the list view is displayed, and the end user presses the New button to create a new record. This is unnecessary in Task UI, at least when only a single record is being created. Instead, the new record can be created using a Siebel Operation step. Then use a View step to display a view having a form applet. The form applet shows the new record, which has empty fields for the end user to fill in.

  In your implementation of Task UI, list views are appropriate whenever lists of items need to be shown or created. However, you will likely find that you do not use list views as often in Task UI as you do in the ad-hoc UI.

- Using applet messages.

  When you are combining dynamic data, such as business components (including transient business components) with static text.

- Using radio buttons and picklists for user decisions.

  Use radio buttons and picklists to constrain user responses and maintain a tight task flow.

- Overriding default method disabling.

  In most cases, default methods should be disabled. The common exceptions are: New buttons on list applets for nonloop operation, as well as Query buttons which constrain data.

- Using buttons to invoke methods and services.

  Buttons and scripts should not navigate the user to a different view. If this is done, the task pauses. Services that call server components should not assume they are working on uncommitted task data.

- Modifying the look and feel using templates.

  Usually, you do not need to modify templates. Use the IsInTask method, target Task UI Service (SWE), to determine whether a section of template should be executed in task mode only.

- Avoiding Applet search specs.

  Review the applet search specs and move to Task View Step Search Specs if possible. This way, you can avoid confusion when reusing applets across task views

Structuring Pages

Consider the following points when designing pages within a task.

- **Create goal-oriented task titles.** Your task titles should emphasize the goal of the task to be accomplished. For example: Create a New Company.
- **Create concise page titles.** Each page title should be an explicit statement of its purpose.
- **Create useful page explanations.** Briefly elaborate on the purpose of the page.

Creating Page Content

A page should not include content that is not suited for the purpose of the page. Page content types fall into the following four general categories:

- **Overview page.** A page at the start of the task that is used to provide an overview of the task's goal, which can help the user determine whether to proceed with the task or not. For example, an overview page of a driver license renewal task may indicate that completing that task requires the user's social security number and driver license number.
- **Step page.** Should contain only data that is relevant or meaningful to the current task step.
- **Review page.** Allows the end user to review any data before it is committed. This is different from a summary page, which is post-commit. The review page allows the user to review, edit, and navigate back in the task flow to make adjustments. Review pages can appear anywhere in a task where an end user is required to commit data.
- **Summary page.** Provides a log and confirmation of that which the end user has accomplished. This page should not contain any editable data fields, as it is a confirmation page of information already committed to the database.

Reusing Views and Applets

It is recommended that you create new applets and views to enforce the simpler layout requirements of Task UI. Use the CSSSWEFrame template for new applets where possible, to avoid performance implications from any specialized code that may exist.

Working with Buttons and Menus

Consider the following points when implementing buttons and menus:

- Buttons and menus should not require a context business component or record.

- Make sure that tasks are deployed before adding buttons or menus to invoke them. Only activated tasks should be invoked from buttons or menu items. Otherwise users receive run-time errors.

Using Script to Invoke a Task

Invoking tasks from scripts has limited support. In general, there must be an active UI context before invoking a task. Note the following restrictions:

- You cannot invoke tasks from the middle of events (for example, WriteRecord).
- Tasks started from script are launched immediately and the end user is immediately shown the first view. The restriction is that tasks cannot be created from script and then be immediately paused to the Inbox. The Workflow Task step does this, but it cannot be done from script.
- Tasks that require a context record: scripts should only call tasks that do not require, or optionally require, a context.

Using Default Focus

When laying out any page, make sure the most likely first action is suggested by smart placement of the cursor, which is to say, the default focus. For example, on an overview page, the most logical next action is to press the Next button, so place the default focus on the Next button. On a content page, it is most likely the first editable field, so configure the cursor to appear there first. All of your pages should have default focus so the user can use the keyboard to navigate.

Common Types of Views Used in Tasks

Task views fall into the common type categories that follow:

- **User decision or user selection.** The end user chooses from a set of options that determine how the task flow proceeds.
- **Single-object data entry.** A form view with built-in new record.
- **Multiple-object data entry.** A list view with New button.
- **Task review.** This view displays data entered so far in the task, with options for modification.
- **Summary view.** This view is generally used at the conclusion of the task.

Design Patterns for Task User Experience

User Experience Design Patterns consist of the following:

- “Selector” on page 140
- “Hierarchical Selector” on page 140
- “Split View” on page 140

Selector

End users need to choose between several available options.

Hierarchical Selector

End users need to make several mutually dependent decisions. Avoid cascading a series of selector views with only relevant options.

Split View

Because business components with large numbers of fields are common to Oracle’s Siebel Business Applications, there are often dependencies between fields which require they be entered in a specific order. Views showing all business component fields at once are complex and difficult to use. Task UI must guide the user through the complexity in Siebel business components and the data model.

Split data entry for a single record in multiple views, typically contains a form applet based on a business component whose fields are being split. Navigating backward and forward again presents different views of the same record, with field values that have already been set being preserved.

NOTE: It is best to keep dependent fields in the same view, thus avoiding a possible deadlock, where changing a field shown in one view changes a field shown (for example, through a pick-map or the On Field Update user property) in another view and makes it irreversibly invalid.

Example

In a task reviewing account data, relevant sub-set of Account BC's 213 single-value and 88 multi-value fields can be split in several views: first view shows identification data (Name, Location, DUNS, CSN etc.), second shows general classification info (Account Type, Industry, Territory, etc.) and third shows the communication info (primary address, phone #, fax # and URL).

Optional View

When a task needs to present some data for review, whose existence is optional, add a Siebel Query Option step before the optional view to check for the existence of data to be presented. Assign the query step's NumAffRows output argument to a task property of type Number. Follow the query step with a decision step with two outgoing branches: one of type Condition, and the other of type Default. In the conditional branch, test that this task property's value equals literal value of 0. Use that condition branch to by-pass the optional view, because it will be executed if no rows were found in the query step.

Example

Field Activity may have instructions associated with it, but if there are no instructions to show, the view that presents them can be skipped.

Mixed View

When a Siebel applet is bound to a single business component, but the business component boundaries are not always intuitive or obvious to the end user, your task-based UI needs to present a logical set of data in a view. You need to combine data from multiple business components, including a transient business component in a single task view step. Because applets can be bound together in Task UI, the end user does not need to worry about artificial boundaries between business components, except that validation is performed when input focus changes from one applet to another, which should be taken into account when laying out the view.

Example

A view presents a list of items in an order and asks whether the user wants to add a new item. The list of order items is based on the Order Item business component, and is displayed by a single applet, while the second applet contains the question, displayed as an applet message, and the answer to the question, stored as a field in the TBC and presented as a radio button.

Mixed Applet

When data from several business components needs to be presented in a single applet, you need to create a transient business component that holds data for the Mixed Applet. Then you move the data to the appropriate nontransient business components as soon as possible. Ideally, you perform this immediately following the view containing the Mixed Applet.

Example

CME Order task needs to ask for account name and customer first and last name in a single applet. Later, this data is used to query existing account and contact, and if none is found, create new account and/or contact record.

Business Component Method Invocation

The ad-hoc UI largely depends on the end user clicking a button or choosing a menu item to invoke a particular piece of business logic. This business logic typically is encapsulated in a business component method call, and not readily available as a Business Service method. The problem with this kind of user experience is that the end users need to know the following:

- That they are required to click a button
- Which behavior is caused by pressing the button
- The prescribed sequence in which buttons are pressed

Task UI allows you to avoid this complexity by:

- Presenting the end user with unambiguous choices
- Explaining which behavior different choices can cause
- Providing backward-navigation choices in case the end user makes a mistake in selection
- Enforcing a prescribed sequence when applying business logic

With Task UI, you can provide the end user with a selection of clear choices, one at a time. The end user makes a selection and presses the Next button on the task playbar applet, after which the task controller invokes appropriate business logic in the correct order as defined by the task flow. In case of an exception, an error dialog pops up; otherwise taskflow execution moves to the next view.

If one does not already exist, define an adapter Business Service based on the CSSBCAdapterSvc class, for which the value of the Business Component user property specifies the business component whose method(s) it is adapting. Define a method with the same name as the business component method you need to invoke from the task.

Now you can invoke this method as a Business Service step in your task flow following the view step in which clicking the Next button needs to trigger invocation of the business logic in the business component method.

NOTE: If you enable backward navigation to the selector view, and it presents an option that logically negates the business logic invoked behind the scenes, you might need to compensate for it if that option is selected.

Example

A field activity task presents the end user an option to automatically generate an invoice, which in the ad-hoc UI is done by clicking on the AutoInvoice button in the Invoice applet. In the field activity task, this means calling an adapter business service method following the selector view. Presenting the AutoInvoice button in the Invoice applet is no longer needed.

About Queries That Users Can Refine

In general, it is recommend that queries be pre-configured into a Task's definition, typically as a search specification in Task View steps. However, sometimes it is just not possible to know at design time the exact search specification that user will need at run-time, which may result in a large number of records that satisfy the pre-defined search specification. Thus, you need to allow users to refine queries by using query-by-example.

In this design pattern, task view step specifies a relatively broad search specification. Task Controller sets this search specification as an anonymous search specification on the given business component, substituting any task properties referenced using '&' prefix with their respective literal values. Task view contains a 'Refine Query' button, which executes 'Refine Query' method on the BusComp, which keeps the original search spec, but lets the user change the search criteria. After entering refined search specification, user clicks on 'Execute Query' button in task view (that is only shown in query mode), which re-executes the query using the newly refined search specification. Note that if the fields that the original query was based on are not shown in the task view, the user can not remove the original query, which is often required.

Example

The Create Order task (in Communications) contains a task view step that allows the user to select a number of products. Initially, the products are searched by the customer's zip code, but that query may return hundreds of applicable products. Thus, the user has the option of refining the query by name (for example, [Name] LIKE '*300 minutes*'). Since zip code is not shown, user can only narrow down the original search spec, but can not widen it.

Customer Dashboard Data Push

A task that is capturing customer information needs to push that data into customer dashboard pane.

Review

A task may take long time to complete, and business requirements ask for data updated by the task to be visible by other users as quickly as possible. At a point in task where data entered so far should become visible to other users of Siebel database, add a view step with (Forward Button = 'Submit) that reviews the data entered in the task so far, followed by a commit step.

Example

In a Field Activity task, whenever the Status field is updated (for example, to In Process), it must be committed to the base table, so that other users can see it and act accordingly, regardless of whether the current task has finished or not. So every Siebel Operation that updates the activity's status is followed by a review view step with a Submit button, followed by commit step. This behavior makes sure that end user understands that changes are being committed, and gives her a chance to step back and choose a different status before committing it for other users to see, if new status has been changed by error.

Best Practices for Configuring Tasks

This topic includes the following topics:

Best Practices for Configuring Multi-Lingual Tasks

Design UI tasks that are correctly internationalized and that can be localized to run in multiple languages (using language-independent values and formats in Decision steps). Pay close attention to the following:

- All Display Names use Symbolic Strings (no overrides)
- Use LIC instead of hard-coded LDV
- Task Property values
- Business component and TBC Field values
- Siebel Operation steps
- Conditions
- LOVs
- Multilingual LOVs (MLOVs)
- Marked for translation

Best Practices for Invoking Business Services

Business services allow you to execute predefined or custom actions in a task flow. Some examples of predefined business services include:

- **Notifications.** Notifications can be sent to employees or contacts using the Outbound Communication Server business service.
- **Assignment.** Assignment Manager can assign an object in a task by calling the Synchronous Assignment Manager Request business service.
- **Server tasks.** You can run a server component task using the Asynchronous or Synchronous Server Requests business service.

  NOTE: Any code invoked (either synchronously or asynchronously) will not be able to see any data not yet committed from temporary storage to base tables. You can avoid this problem by adding a commit step before invoking the server component task.

You can use Siebel VB or Siebel eScript to define your own custom business services that you can invoke from a task. You can define business services by navigating to Site Map > Administration - Business Service, or by selecting the business service object in Oracle's Siebel Tools. The methods and arguments you define in your business service appear in the picklists in the Arguments list applets for the business service.

The main parts of creating Business Service steps for a task are:

- Defining a business service step.
- Defining input arguments for business service steps.
- Defining output arguments for business service steps.

CAUTION: Business services invoked from tasks cannot include browser scripts; they only work with server scripts. A business service with browser scripts will fail if it is executed from a task on the Siebel Server.

Index

No index is available for this guide.