Siebel

Order Management Infrastructure Guide

June 2024

Siebel
Order Management Infrastructure Guide

June 2024

F87429-03

Copyright © 1994, 2024, Oracle and/or its affiliates.

Author: Sukanya Mishra

Contents

Get Help i

1 What's New in This Release 1

What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 24.6 Update 1
What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 20.1 Update 1
What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 19.1 Update 1

2 Service-Oriented Architecture 3

Service-Oriented Architecture 3
About Business Services 3
About Service-Oriented Architecture 3
How Siebel C/OM Is Built on a Service-Oriented Architecture 5
How Siebel C/OM Can Be Integrated with Other SOA Applications 5
Web Services for Customer Order Management 8

3 Signals 17

Signals 17
About the Signals Mechanism 17
Creating Signal Actions 19
Invoking Signals from Controls and Custom Script 24
Using Recursion with Signals 25
Migrating Signals Between Environments 25

4 Variable Maps 27

Variable Maps 27
About Variable Maps 27
Components of Variable Maps 29
Supported Source Types for Variables 31
About Using Variable Maps 35
Variable Map Methods of the Context Service Business Service 41

5 PSP Engine 43
PSP Engine 43
About the Product Selection and Pricing Engine 43
Components of the PSP Engine 46
PSP Driver Workflow 51
Conditions and Actions for PSP Procedures 53
About Temporary Variables 58
Row Set Transformation Toolkit Methods 58
Configuring PSP Procedures 74
Creating a Custom PSP Application 79
Calling a PSP Procedure from an External Application 79
About Logging of PSP 80
About Troubleshooting of PSP 80
About Tuning Performance of PSP 83

6 PSP Waterfall 93
PSP Waterfall 93
About Waterfalls 93
About Configuring Waterfall Output 95

7 Unified Messaging 101
Unified Messaging 101
About Unified Messaging 101
Components of Unified Messaging 102
Unified Messaging Service Business Service Methods 107
Creating Message Types 108
Configuring the Display of Messages 110
Implementing Multilingual Substituted Text 111
Implementing a Custom Message-Generation Engine 112
About Working with Message Responses 112
About Suppressing Duplicate Messages 113
Suppressing Duplicate Messages 114
Migrating Message Types Between Environments 115
Tuning Performance of Unified Messaging 115
Using Unified Messaging with the PSP Engine 115

8 Data Validation Manager 117
Data Validation Manager 117
About Data Validation Manager 117
Roadmap for Implementing Data Validation Processing 118
Process of Administering Data Validation Rules 118
Process of Invoking the Data Validation Manager Business Service 127

9 Approvals Manager 133
Approvals Manager 133
About Approval Processing 133
ISS Approval Business Service Methods 134
Defining Approval Items and Approval Stages 135
About Invoking the Approvals Manager Business Service from a Workflow 136
Approving or Declining Approval Stages (End User) 139

10 Asset-Based Ordering Methods Reference 141
Asset-Based Ordering Methods Reference 141
Product Manipulation Toolkit Business Service Methods 141
Order Entry Toolkit Business Service Methods 197
Account Administration Toolkit Business Service Methods 205
Complex Product AutoMatch Business Service Method 208

11 Projected Asset Cache 213
Projected Asset Cache 213
About Projected Asset Cache 213
Projected Asset Cache Business Service Methods 214
Using the VORD Projected Asset Cache Business Service 218

12 Compound Product Validation 221
Compound Product Validation 221
About Compound Product Validation Engine Business Service 221
Compound Product Validation Engine Business Service Methods 222

13 Copy Service 227
Copy Service 227

About Copy Service 227
Copy Service Methods 228

14 Data Transfer Utilities Business Service 239
Data Transfer Utilities Business Service 239
About Data Transfer Utilities 239
About Data Maps 246
Example of Defining Data Maps to Use with the DTU 251
Examples of Invoking the DTU 253
Data Transfer Utilities Methods 256

15 Other Component Business Services for C/OM 261
Other Component Business Services for C/OM 261
InMemory Upgrade Data Service 262
SIS OM PMT Service 264
Context Service Business Service 265
ISS ATP Service 266
ISS Credit Card Transaction Service 267
ISS Credit Check Service 268
ISS Disable Service 269
ISS Package Product Service 270
ISS Payment Profile Service 271
ISS Promotion Agreement Manager 271
ISS Promotion CP Admin Service 276
ISS Promotion Edit UI Service 277
ISS Promotion Management Service 278
ISS Revenue Synchronization Service 285
ISS Sequence Service 285
ISS Service Product Service 285
ISS Shipping Calculation Service 286
ISS Shipping Cost Service 286
ISS Smart Part Number Generation Service 286
ISS Spread Discount Service 287
ISS Tax Calculation Service 287
ISS Template Service 287

Get Help

Preface

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

Using Oracle Applications

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

Documentation Accessibility

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

Contacting Oracle

Access to Oracle Support

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

Comments and Suggestions

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

1 What's New in This Release

What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 24.6 Update

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

Topic	Description
InMemory Upgrade Data Service	New topic. InMemory Upgrade Data Service stores document in cache and operates on stored document through provided methods.
SIS OM PMT Service	New topic. SIS OM PMT Service provides many of the functions to support out-of-the-box order management workflows, and as part of these workflows various OOTB Profile Attributes are set.
ISS Promotion Management Service	Modified topic. *CollectAssetList Method* is added. This method collects assets selected by the user and assets not selected but covered by selected promotions.

What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 20.1 Update

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

What's New in Siebel Order Management Infrastructure Guide, Siebel CRM 19.1 Update

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

2 Service-Oriented Architecture

Service-Oriented Architecture

This chapter discusses Oracle's Siebel order management which is based on a service-oriented architecture (SOA), and the services that form the foundation of Customer Order Management (C/OM) functions. This chapter includes the following topics:

- *About Business Services*
- *About Service-Oriented Architecture*
- *How Siebel C/OM Is Built on a Service-Oriented Architecture*
- *How Siebel C/OM Can Be Integrated with Other SOA Applications*
- *Web Services for Customer Order Management*

About Business Services

A business service defines reusable business logic that can be executed within the Object Manager. Business services are the building blocks of all C/OM functions.

Generally, a business service:

- Can be a built-in service that is defined in Web Tools or a run-time service that is defined in the Siebel client application by administrators
- Can be based on the CSSService Class (standard business service) or on specialized classes (specialized business service)

  Note: Specialized business services are used only by internal Siebel Engineering personnel. Do not use specialized business services unless their behavior is specifically documented.

- Can be configured by properties or scripts (written in Siebel VB or Siebel eScript)
- Can be used for generic code libraries that are called from other scripts
- Can be referred to by commands associated with a menu item or toolbar button

About Service-Oriented Architecture

Service-oriented architecture (SOA) is the environment that supports the building of applications using service technology. Siebel order management is a composite application built following the discipline of SOA.

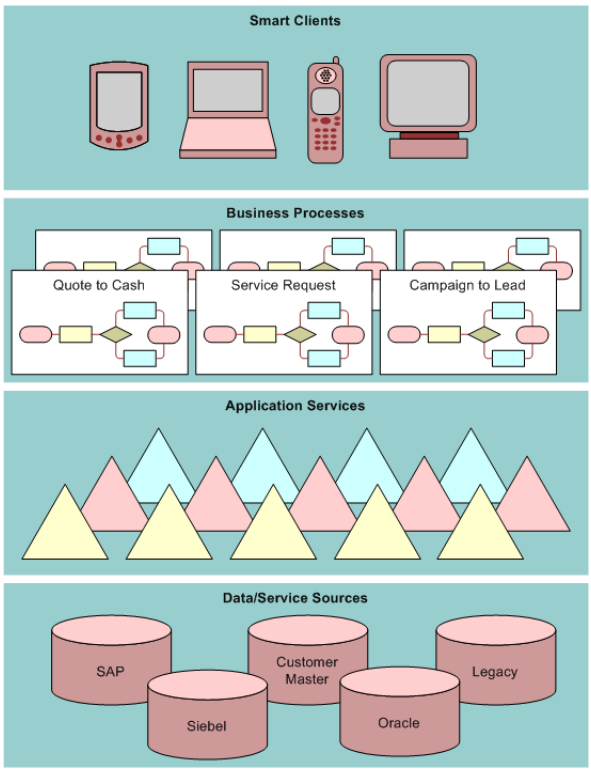
SOA allows for sharing of business logic across multiple access channels, using data and application features wherever they reside. An SOA application must include the following:

- **Smart clients.** A set of clients—connected or mobile, and with multiple form factors—provides for multichannel, role-based access to the application. The clients are "smart" in the sense that appropriate

application code is transparently loaded into the client, allowing high interactivity with no administration overhead; a smart client offers the advantage of both browser technology and client or server technology, without the drawbacks of either. Smart clients support role-based user interfaces.

- **Business processes.** SOA supports process-enabled applications. Each process is declaratively defined as an orchestration of services. The location of services is transparent to the applications, and the processes may cross applications. Various sections of a process may be implemented in different applications, each executed under the control of its own process controller, whether BPEL-compatible or custom.
- **Application services.** All application functions are modeled using service technology. All services—whether data services, business services, or integration services—follow the service paradigm. Data services use the methods associated with data. Business services may drive role-based user interfaces or they may implement automated steps. Integration services (or integration applications) map services consumed to services offered between applications, so that all services appear to be local to each application, smoothing out the differences in object structure and service interface semantics.
- **Data sources and service sources.** At the logic level, all applications are peers as providers and consumers of services and data.

The following figure illustrates a service-oriented architecture.



SOA allows for abstraction of the application interface from the application's implementation. Because of this abstraction and standardization, generalized (coarse-grained) services can be used for a wide range of needs. Using generalized services means that there is reduced demand for new services, and services can be reused in unforeseen contexts. At the same time, services that are fine-grained can be used for the composition of new services.

How Siebel C/OM Is Built on a Service-Oriented Architecture

Siebel order management is a composite application following the principles of SOA as follows:

- **Services are autonomous, and they act independently of one another.** C/OM business functions are based on independent services involving pieces of code and data. Each service is a unique piece of code that stands alone, independent of other services. Services share standards, schema, and contract—but because services are autonomous, each one has its own implementation, deployment, and operational environment. For this reason, a service can be rewritten or replaced with no impact on partner services.
- **Messaging carries information between services.** Services interact through messaging. The only way into and out of a service is through messages. A message's schema describes the format and content of the message. A message's contract describes the message sequences allowed in and out of the service. The schema definition and the contract definition give a service its black box nature. A partner service of any given service is aware only of the sequencing of messages flowing back and forth, not of the service's inner workings.
- **Boundaries are explicit.** Explicit boundaries mean that there is no ambiguity regarding the location of each part of the code; it is clear whether the code resides inside or outside of the service. The same principle applies to data. It is known whether a database table resides inside or outside the service.
- **Service location and compatibility are describable and discoverable.** Policies exist as formal criteria for getting a service to do its work and for specifying service location. The criteria are located in a document that outlines the service's rules for use and its location.

In this release, C/OM business processes are implemented as workflows that invoke a series of internal services. The SOA also allows C/OM applications to incorporate external services into any business process.

Because Siebel order management is built upon SOA principles, C/OM business functions are encapsulated in well-defined services. Data is passed to and from services as hierarchical documents.

The C/OM Signals mechanism provides the service invocation framework. The C/OM Variable Maps mechanism defines, constructs, and persists the data passed to and from the services.

The service-oriented architecture of Siebel order management also means that C/OM business processes and functions can be exposed (as stateless services), so that they can be called by external applications. The service definition and run-time is supported by the Siebel ASI framework.

How Siebel C/OM Can Be Integrated with Other SOA Applications

Web Services is the most common enabler of SOA. Siebel Business Applications support both inbound and outbound Web Services. The Siebel application can:

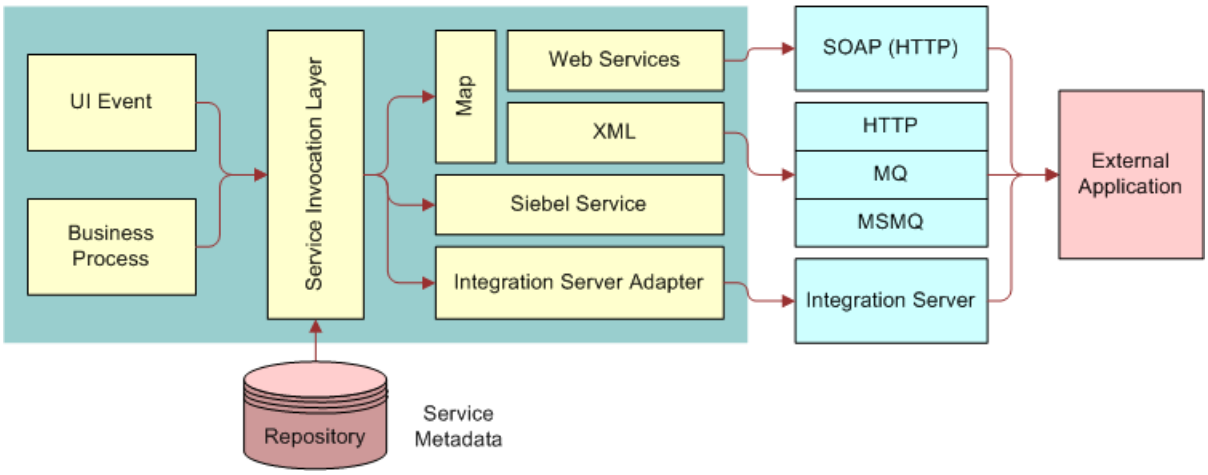
- Generate and read WSDL
- Process and transform XML
- Receive and process Web Service requests over HTTP

- Invoke an external Web Service from any Siebel event, script, or Workflow

Outbound Integration of C/OM Services

You can call an external service from C/OM. Predefined integration interfaces can be implemented or hosted by an external application. Service can be provided by an external application, an integration server, a Siebel business service, or a Siebel business process (workflow).

The following figure illustrates services integration for outbound integration.



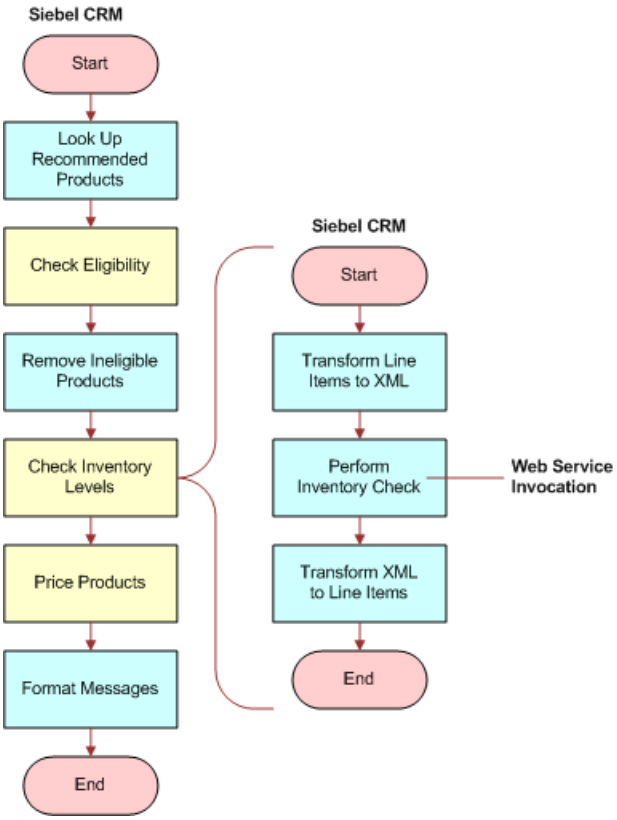
As shown in this image:

- UI Event and Business Process connect to the Service Invocation Layer.

  The Repository (Service Metadata) also connects to the Service Invocation Layer.
- The Service Invocation Layer connects to Map (Web Services and XML), Siebel Service, and the Integration Server Adapter.
- Connection to the External Application is possible via:
  - SOAP (HTTP) using Web Services.
  - HTTP/MQ/MSMQ using XML.
  - Integration Server using Integration Server Adapter.

Calling an External Service from C/OM

The workflow process shown in the following figure provides an example of calling an external service from Siebel order management. The figure shows a workflow process that includes a subprocess called Check Inventory Levels. The subprocess includes a step called Perform Inventory Check, which involves a Web service invocation.



As shown in this image:

- The steps in the main workflow are as follows: Start, Look Up Recommended Products, Check Eligibility, Remove Ineligible Products, Check Inventory Levels, Price Products, Format Messages, End.
- The Check Inventory Levels step in the main workflow connects to the subprocess workflow.
- The steps in the subprocess workflow are as follows: Start, Transform Line Items to XML, Perform Inventory Check (Web Service Invocation), Transform XML to Line Items, End

Web Service Performance

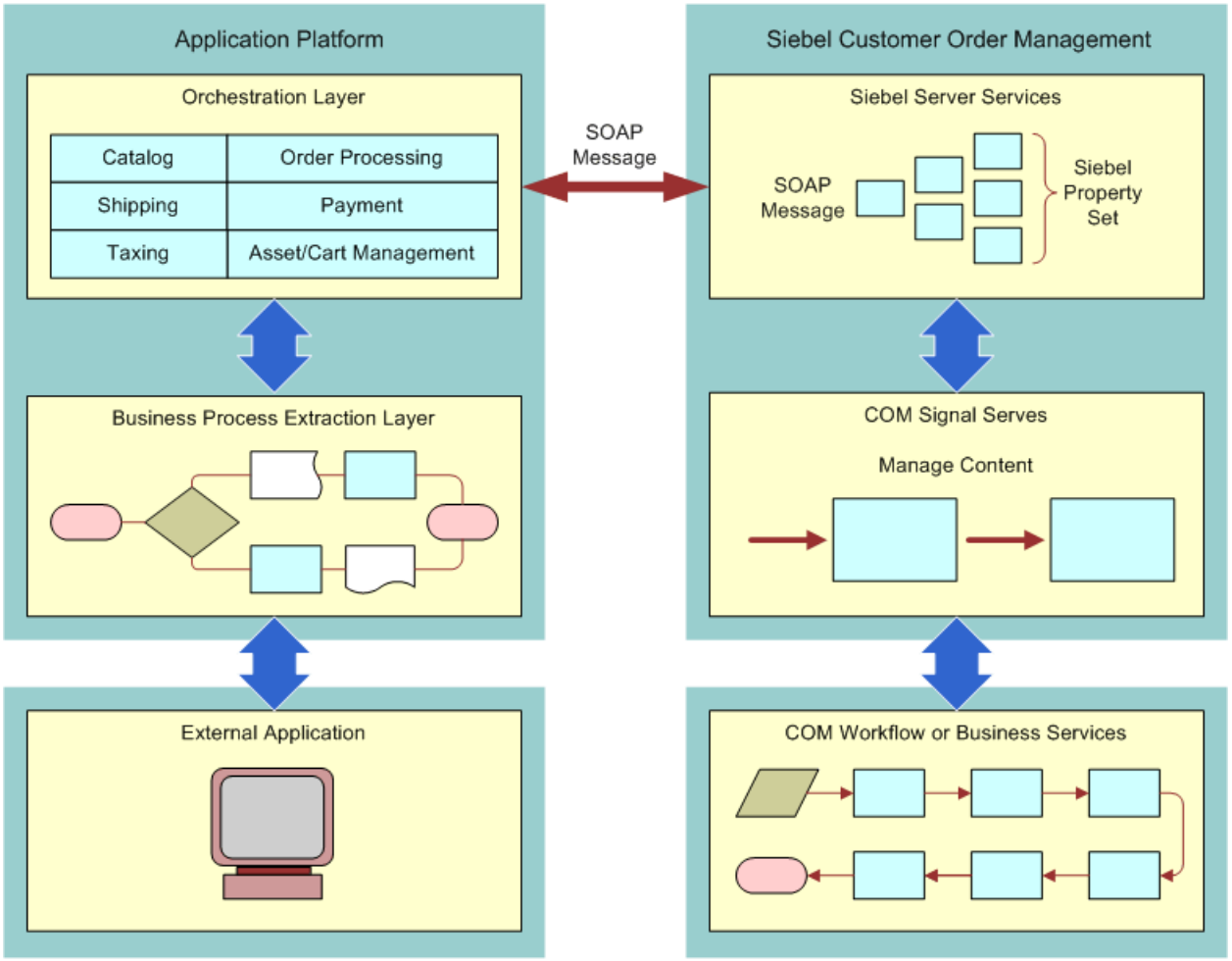
C/OM services such as Pricer or Eligibility are designed to work on batches of data to improve end-user response times. Any external service called by Pricer or Eligibility must support a batched interface that processes an entire set of data (such as all line items in an order) in a single invocation. Thus the overhead associated with Web Service invocation and with context establishment within the Web Service is only incurred once instead of, potentially, hundreds of times.

How Siebel C/OM Can Be Used with SOA

Siebel customer order management can be used as a service by any SOA application though the process flow illustrated in the following figure. In this flow:

- The external application UI first identifies (through a business process extraction layer) the right set of Web services that it needs to call to support the business process event.

- The external application layer then:
  - Identifies the right sequence of Web service invocations.
  - Prepares the input to these Web services, and generates the SOAP message appropriately.
- The Siebel Business Application server Web service listener will receive the soap message, and if needed, facilitates session management and converts the SOAP message to a native property set.
- The signal service invokes a COM signal, or calls the native service and invokes the COM order management workflow or business service to complete the task.
- Once the task is completed in the Siebel Business Application, the Siebel application returns a SOAP message back to the calling application which in turn extracts the output and updates the UI.



Web Services for Customer Order Management

The Web services used for customer order management are listed in the following table.

- *Web Services in Version 8.1* describes the new, modified, and consolidated web services to support release 8.1 of the software.

- The following table lists the workflows that you must activate in order to use the Web services for customer order management.

For more information about these Web services and for information about enabling Web services, see *Siebel CRM Web Services Reference* .

Namespace	Name
http://siebel.com/OrderManagement/ABO	ABOWebService
http://siebel.com/OrderManagement/Asset	AssetWebService
http://siebel.com/OrderManagement/Quote/PSP	CalculatePriceWS
http://siebel.com/OrderManagement/External/PSP	CalculatePriceWS
http://siebel.com/OrderManagement/Catalog	CatalogWebService
http://siebel.com/OrderManagement/Contact	ContactWebService
http://www.siebel.com/OrderManagement/ContextService	ContextServiceWrapperService
http://siebel.com/OrderManagement/Quote/PSP	EligibilityCompatibility
http://siebel.com/OrderManagement/Order	OrderWebService
http://siebel.com/OrderManagement/Configurator	ProductConfigurator
http://siebel.com/OrderManagement/Quote/PSP	ProductRecommendation
http://siebel.com/OrderManagement/Promotion	PromotionWebService
http://siebel.com/OrderManagement/Quote	QuoteAddItemsWS
http://siebel.com/OrderManagement/Quote	QuoteWebService

Namespace	Name

Web Services in Version 8.1

This topic describes the new, modified, and consolidated web services to support release 8.1 of the software:

- The first table in this topic lists the Web services that are new for customer order management.
- The second table in this topic lists the new self service Web services that were added for customer order management.
- The third table in this topic lists the new Communications, Media, and Utilities (CMU) Web services for customer order management.
- The fourth table in this topic lists the Web services that were modified for customer order management.
- The fifth table in this topic lists the Web services that have been consolidated for customer order management.

New Web Services

The following table lists the Web services that are new for customer order management.

Type	Web Service Name	Description
Catalog	Get Categories	Retrieves a list of all available product categories in a single Web service interaction.
	Publish Catalogs	Retrieves all catalog objects (categories, products, attributes, attribute domains) for a given catalog including private catalog objects based on current catalog access control, and eligibility enforcement options.
Shopping Cart	Price Lists	Gets all active price lists for a given context.
	Order Detail	New UI data service (UDS) based on Quote and Order Web services that activate and return only the information (fields) that you requested.
	Quote Detail	
	Quoting	
Promotion	Get Promotion	Gets commitments for a given promotion asset.
	Modify Promotion or Asset	Upgrades or migrates a promotion instance to another promotion. Supports promotion upgrade or downgrade process from an external application.
	Modify Promotion or Asset	Modifies a promotion or asset item from an external application through the modify asset Web service.

New Self Service Web Services

The following table lists the new self service Web services that were added for customer order management.

Self Service Web Service	Description
SelfServicePostLogin	Loads the logged in users responsibilities, contact details and primary account. If the primary account is null, loads the root account of the logged in user's contact.
SelfServiceRegistration	Performs the following actions for user registration: • Creates a user • Creates a contact • Creates an account, if required • Assigns responsibilities • Triggers approval process, if applicable This Web service also performs the reset password and update password transactions.
SelfServiceContact	Allows you to insert, delete, update, and query a contact, contact details, and the accounts associated with a contact.
SelfServiceAccount	Allows you to insert, delete, update, and query accounts, account addresses, and contacts associated with accounts.
SelfServiceAccountRootPath	Queries the account hierarchy details of the requested account.
SelfServiceAllAccountsList	Queries accounts and account details.
SelfServiceUser	Retrieves user details and user responsibilities.
SelfServiceWebSite	Allows you to insert, update, delete, and query the Self Service site and its details.
OrderDetailWebService	Queries Order header, Order Line, Order Payments, Order Approval, and Shipment details.
SelfServiceSmtpEmail	Send an email using the Outbound Communications Manager business service.
SelfServiceTemplateRule	Queries the Self Service Site template rules.
SelfServiceTimeZone	Queries the time zone and time zone translation details.
SelfServicePaymentHistory	Gets the payment history for a contact or account.
SelfServiceAccountBillingProfile	Performs Insert, Delete, Update, and Query on the accounts billing profile.

Self Service Web Service	Description
SelfServiceAddress	Gets the Address details for a particular account or contact.
SelfServiceResponsibility	Queries responsibilities.

New Communications, Media, and Utilities Web Services

The following table lists the new Communications, Media, and Utilities (CMU) Web services for customer order management.

CMU Web Service	Description
CatalogWebService	Equivalent to the Publish Catalogs Web service.
OrderDetailWebService	Equivalent to the Order Detail Web service.
SelfServiceAccount	Equivalent to the SelfServiceAccount Web service.
SelfServiceAllAccountsList	Equivalent to the SelfServiceAllAccountsList Web service.

Modified Web Services

The following table lists the Web services that were modified for customer order management.

Web Service	Description
PromotionWebService	The following methods have been added: • Promotion Commitments • Upgrade Promotion
QuoteWebService	Allows you to insert, update, query, and delete the quote header, quote item, and quote payments.
ProductConfigurator	To support release 8.1 of the software, updates the Web service: • To meet all new requirements • To fix any issues found

Consolidated Web Services

The following table lists the Web services that have been consolidated for customer order management.

Web Service	Description
QuoteCheckOutWebService	This is the Submit Order Web service that invokes the QuoteCheckOut workflow. The workflow consolidates the submit order functionality, and the credit card validation process.
QuotingWebService	This Web service is used to save a quote, and it invokes the Web Channel Quoting Workflow. This workflow consolidates the following processes: • Run the eligibility and compatibility workflow • Re-price the quote • Calculate delta for ABO quote • Perform promotion instance check • Calculating shipping charge • Calculate Tax
SelfServiceRegistration	This is the self service registration process, and it invokes the SelfServiceRegistration workflow. It consolidates the following processes: • Create a user • Create a contact • Create an account, if required • Assign responsibilities to the user • Trigger approval process, if applicable
SelfServicePostLogin	SelfServicePostLogin is invoked after a user logs into the self service application, and it invokes the SelfServicePostLogin workflow. It consolidates the following processes: • Load contact details and responsibilities • Load primary account id and root account details

Workflows to Activate for Customer Order Management

In addition, to use the Web services for customer order management, you must activate the following workflows:

- CalculatePriceExternal
- Check Eligibility & Compatibility - Default
- Compatibility Multiple Popup Workflow
- Configurator Eligibility Compatibility Workflow
- Configurator External Validate Workflow
- Configurator Load
- Configurator PAC Query
- Configurator Product Info Lookup
- Configurator Save
- Contact - New Order

- Contact - New Quote
- ContextServiceWrapperService-OrderHeader-Verify
- ContextServiceWrapperService-OrderItem-Verify
- ContextServiceWrapperService-QuoteHeader-Verify
- ContextServiceWrapperService-QuoteItem-Verify
- Get Config Item Price
- Get Product List Price
- Goto_Order
- Goto_Quote
- ISS Approval (Agreement)
- ISS Approval (Order)
- ISS Approval (Quote)
- ISS Post Approval Workflow (Agreement)
- ISS Post Approval Workflow (Order)
- ISS Post Approval Workflow (Quote)
- ISS Promotion Agreement Covered Assets Sub Process
- ISS Promotion Agreement Management Sub Process
- ISS Promotion Commitment Compliance Check SubProcess
- ISS Promotion Create Agreement Details
- ISS Promotion Disconnect Integration SubProcess
- ISS Promotion Disconnect Process
- ISS Promotion Disconnect Process - for Verify
- ISS Promotion Recommendation SubProcess
- ISS Promotion Upgrade Process
- ISS Promotion Verify SubProcess
- ISS Validation (Agreement)
- ISS Validation (Order)
- ISS Validation (Quote)
- PSP Driver Workflow Process
- PSP Dynamic Matrix - Refresh Matrix Cache
- PSP Refresh Cache On Cache Key - Price List
- PSP Waterfall Driver Workflow Process
- PSP Waterfall Synch Test Workflow
- PSP Waterfall Synch to DB Workflow
- Pricing Procedure - Bundle Discount Unit Test
- Pricing Procedure - Calculate Net Price

- Pricing Procedure - Default
- Pricing Procedure - Keep Discount Flag
- Pricing Procedure - Service
- Pricing Procedure - Volume Discount
- Product Compatibility - Default
- Product Eligibility & Compatibility - Default
- Product Recommendation Delete Msgs
- Product Recommendation Driver Workflow
- Product Recommendation Get Recommended Products
- SIS OM Active Order Sub-Process
- SIS OM Active Order Sub-Process - Contact
- SIS OM Active Quote Sub-process - Contact
- SIS OM Apply Completed Service Order Line Item to Service Profile
- SIS OM Auto Select Order Billing and Service Accounts
- SIS OM Go to Products & Services Sub-Process
- SIS OM Go to Quote Detail View Sub-Process
- SIS OM Modify Products & Services Process
- SIS OM Modify Products & Services Process - Contact
- SIS OM Modify Products & Services Process - Quote & Order
- SIS OM New Products & Services Process
- SIS OM New Products & Services Process - Contact
- SIS OM New Products & Services Process - VORD
- SIS OM Profile Process
- SIS OM Profile Process - Order
- SIS OM Quote To Order Workflow PMT Version
- SIS OM Submit Order Process
- SIS OM Suspend, Resume Asset Sub-Process
- SIS OM Suspend, Resume Asset Sub-Process - Contact
- SIS OM Suspend, Resume Products & Services Process
- SIS OM Suspend, Resume Products & Services Process - Contact
- SIS OM Suspend, Resume Products & Services Process - Quote & Order
- SIS OM Ungroup Order
- SIS OM Ungroup Quote

For more information about activating workflows, see these Web services and for information about enabling Web services, see *Siebel Business Process Framework: Workflow Guide* .

3 Signals

Signals

In earlier releases, Siebel order management functions and function calls were handled locally by the code in the calling object or business component. In this release, every interaction between C/OM components occurs through an API invocation that you can configure or redirect. These API invocations are called signals. A signal is a request to perform a business function.

Each API invocation supports a configurable set of input arguments. The API is handled by a series of business service methods and workflows that you define using the Administration - Order Management screen, then the Signals view.

This chapter includes the following topics:

- *About the Signals Mechanism*
- *Creating Signal Actions*
- *Invoking Signals from Controls and Custom Script*
- *Using Recursion with Signals*
- *Migrating Signals Between Environments*

About the Signals Mechanism

Within Siebel order management, you use the Signals mechanism to invoke configurable business logic. *Signals* are used to hold together all the C/OM services, and they are used to call external services. You configure signals in the run-time client.

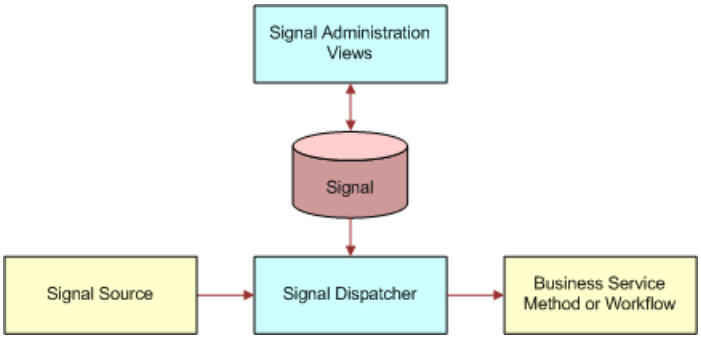
Nearly all buttons and engines within Siebel order management invoke business logic by raising a signal. The business logic invoked comes in the form of one or more business service methods or workflows.

Signals are versioned objects. The numbering assigned to a signal version means that a new version of a signal can be developed while you are using the current version in the same production environment.

A signal action property specifies the business service or workflow to invoke when the signal is raised. Signal action parameters are the input arguments to signal action workflows or business service methods.

Components of the Signals Mechanism

The following figure shows how a signal created using the Administration - Order Management screen, then the Signals view is routed through a signal dispatcher to invoke a business service method or workflow.



Signals Administration

Administer signals using the Administration - Order Management screen, then the Signals view. Use the Signals view for creating new signals, working with existing signals, and releasing signals into production. In the Versions list applet, click the Work Space hyperlink to access the Actions and Properties tabs, and the Parameters list applet, for the selected signal.

Signal Dispatcher

Signals are dispatched by the Context Service business service. The Context Service business service reads and caches signal definitions, and then when a business component raises the signal, Context Service executes the appropriate business service or workflow.

For more information about the Context Service business service, see *Variable Maps* and *Context Service Business Service*.

Signal Sources

All applications supported by the Siebel order management infrastructure can be sources of signals. A signal can be invoked by any C/OM business component. Signals are not supported on non-C/OM business components.

Signals are invoked through the standard InvokeMethod call. If the method is not handled by logic or script on the source business component, then the signal dispatcher (Context Service business service) is invoked to handle the signal.

Signal Actions

A signal action can be either a business service method or a workflow. By setting the Service Type field using the Administration - Order Management screen, Signals, and then the Actions view, you define whether the action is a business service method or a workflow.

Creating Signal Actions

You create and modify signal actions in the Signals view. You can set the sequence for a group of actions, set input arguments for actions, and set filter values that determine when an action executes. Use the following procedure to create and modify signal actions.

The Signals view lists and describes all the standard signals that are available in your Siebel Business Application. The following table lists some of the signals that are relevant to order management and pricing administration.

Signal Name	Description
ApproveItem	Invoked by clicking Generate Approvals in quote, order, and agreement views. Reprices and then generates approval requests for the current line item.
CalculatePriceAndCheckEligibility	Invoked by Add Item and Verify. Performs Pricing and Eligibility on selected rows.
CalculatePrice	Invoked by clicking Reprice or selecting a product for a line item. Establishes context, calls the pricing procedure, and then synchronizes any updates back to the database.
CalculatePriceAll	Invoked by clicking Reprice All. Establishes context, calls the pricing procedure, and then synchronizes any updates back to the database.
CalculatePriceExternal	This signal is called when a user calls the method CalculatePriceExternal for the Pricing Manager business service; it is used to calculate the price of a product externally.
CalculatePrice_eSales	Invoked by clicking Reprice in Siebel eSales. Establishes context, calls the pricing procedure, and then synchronizes any updates back to the database.
CalculatePriceAll_eSales	Invoked by clicking Reprice All in Siebel eSales. Establishes context, calls the pricing procedure, and then synchronizes any updates back to the database.
CalculatePrice_Configurator	Invoked by clicking Reprice in Siebel Configurator. Establishes context, calls the pricing procedure, and then synchronizes any updates back to the CP instance.
CheckEligibility	Invoked by the Verify Eligibility Status menu item on the line items applet. Executes the eligibility procedure for the current line item.
CheckEligibilityAll	Not invoked out-of-the-box. Executes the eligibility procedure for all line items.
CopyOrder	Invoked when copying Order records.
CopyQuote	Invoked when copying Quote records.
Get Config Item Price	Called when the user calls the GetConfigItemPrice method of the Pricing Manager business service; it is used to get the price of a configurable item externally.

Signal Name	Description
Get Product List Price	Called when the user calls the GetProductListPrice method of the Pricing Manager business service; it is used to get the product list price of the product, as well as other price list field values (for example, MSRP Price and Cost).
GetUserProdPrice	Calculates the list price and net price for a list of products using the login user profile as context.
MergeIntoOnePackage	Invoked by the Package menu item in quote, order, and agreement views. Calls the ISS Package Product Service to combine the selected line items into a package; it then reprices the package.
OrderTemplate	Invoked by clicking Add Items in the Catalog or Favorites list applet. Calls the ISS Template Service to copy line items from favorites in to the shopping cart.
OrderTemplateCopy	Invoked by the ISS Template Service to copy template line items to the current order.
OrderTemplateSelectItems	Invoked by the ISS Template Service to copy template line items from the available product applet to the current order.
Product Recommendation Signal	Invoked by a product selection or by a current line item change. Calls the Product Recommendation Driver Workflow to generate and display product recommendation messages.
QuotesAndOrdersValidate	Invoked by clicking the Verify button on a Header or Line Item. On Header: runs Pricing and Eligibility, verifies promotions, validates data rules, and validates CP. On Line Item: checks eligibility and validates CP.
QuoteTemplateCopy	Invoked by the ISS Template Service to copy template line items to the current quote.
QuoteTemplateSelectItems	Invoked by the ISS Template Service to copy template line items from the available product applet to the current quote.
SetFieldValue	Invoked whenever a field value changes in the quote, order or agreement header, or line items. Triggers various processing depending on the field changed.
SpreadDiscount	Invoked by clicking Spread in the spread discount pop-up applet. Calls the Spread Discount Driver Workflow Process to query the selected line items and spread the specified discount.
SpreadDiscount - All	Invoked by clicking Spread in the spread discount pop-up applet when Type is set to All. Calls the Spread Discount Driver Workflow Process to query all the line items and spread the specified discount.
VerifyItem	Invoked by clicking the Verify button or menu item in quotes, orders or agreements. Calls workflows to reprice and check eligibility. Then calls the FINS Validator business service to execute validation rules.

To create and modify signal actions

1. Navigate to the Administration - Order Management screen, then the Signals view.
2. In the Signals list applet, select an existing signal to modify, or create a new signal record.

3. Lock the signal by checking the Locked Flag box.

   This locks the object for your user ID.
4. If you are creating a new signal, give it a name and description, and then save the record.
5. In the Versions list applet, click the Work Space link to drill down to the Actions list applet.
6. In the Actions list applet, create a new action, or select an existing signal action.
7. Complete the fields.

   a. Set the Sequence field value to reflect the sequence number of a particular action relative to other actions for the signal.
   b. Set the Service Type field to specify whether the action is a business service method or a workflow:

      - If you specified that the signal action is a workflow, enter the workflow's name in the Service Name field and enter "RunProcess" in the Service Method field.
      - If you specified that the signal action is a business service method, enter the business service name in the Service Name field and enter the method name in the Service Method field.
   c. (Optional) Set filter fields for the action as described in the table that follows.

      Filter fields limit the execution of a signal action. The action occurs only if all filter field values specified match the current situation.

Filter Field	Allowed Values	Description
Application Name	Any "Application" repository object name	Used to define industry-specific or channel-specific logic.
Mode	Any string from the ISS_MODE LOV	Must match the value of the Mode user property on the business component (such as Quote, Order, Asset or Agreement).
Instance Type	Any string from the ISS_INSTANCE_ TYPE LOV	Must match the integration component name from the integration object specified in the business component "Instance Uniform Name EAI Object:[Current Business Object]" user property, for example, "Header", "Line Item", or "Payments".
Fields	A semi-colon-separated list of field names (example: "Account; Product; Net Price")	The business component fields for which the action is executed. The action occurs if the active field in the calling business component appears in the list.
Condition	A Siebel logical expression that returns TRUE or FALSE.	If the condition is not empty, the action is only invoked when the condition returns TRUE.

8. (Optional) In the Parameters list applet (at the end), enter input arguments for the action.

   As an example, parameters for the CalculatePrice signal are listed in the following table:

Input Argument	Example Value
CPScope	Whole
RowScope	Selected
SubPSPWFName	Pricing Procedure - Default
Variable Map - Context	Default Pricing Variable Map - Context
Variable Map - Row Set	Default Pricing Variable Map - Row Set

9. (Optional) You can specify a CanInvoke check by completing the fields in the Properties list applet. See *Modifying Signal Properties for Signal Actions*.
10. Navigate back to the Signals list applet.
11. Click the Release New Version button to release the signal version.
12. If you are creating a new signal action, create a button, script or workflow to invoke the signal. See *Invoking Signals from Controls and Custom Script*.
13. Test the signal.

    You test the signal by triggering the appropriate event.

    > **Note:** After releasing a new version, you must start a new user session (by logging out and logging in again) to test the latest version.

14. Using Application Deployment Manager (ADM), promote the updated signal definition to the production environment. For information about using ADM, see *Siebel Application Deployment Manager Guide* .

Modifying Signal Properties for Signal Actions

Signal properties are similar to user properties on repository objects. Signal properties are name-value pairs used to configure processing. In this release, the only supported use of signal properties is to provide a CanInvoke check.

> **Note:** For some signals (for example, the QuoteAndOrderValidate signal), the CanInvoke property for the signal can cause related buttons to be disabled. Removing the property, however, results in enabling the related button. `Name: CanInvoke:Order Value: [Status] = LookupValue('ORDER_STATUS', 'Open')`

To modify signal properties for a Can Invoke check

- (Optional) You can specify a CanInvoke check by completing the fields in the Properties list applet as follows:
  - **Name.** CanInvoke:[Mode]

- **Value.** A Boolean expression using one of the following:
  - Business component fields. The expression can be comprised of real business fields or pseudo business fields supported by Context Service. Allowed fields include the following:

Field	Comment
[$IsNewRecordPending]	None
[$HasActiveRow]	None
[$IsInQueryMode]	None
[$CanUpdate]	Returns 'Y' or 'N'
[$GetType]	Returns instance type such as 'Line Item', 'Header', 'XA'

  - Profile attributes. As an example, the following table shows a properties setting for the SetFieldValue signal:

Property	Example Attributes
CanInvoke:Any	`GetProfileAttr('Block Variable Map Operations')='N' OR GetProfileAttr('Block Variable Map Operations') IS NULL`

Example of Signal Properties Settings for a CanInvoke Check

An example of the fields set for a CanInvoke check is listed in the following table.

Name	Value
CanInvoke:Any	[Account Id] is not null
CanInvoke:Quote	[Status] = LookupValue('ORDER_STATUS', 'Open')

Note: If CanInvoke logic exists for a specific Mode, it overwrites the CanInvoke logic defined for the mode Any. In the example for Signal Properties Settings, using Quote mode, the CanInvoke logic used will be `[Status] = LookupValue('ORDER_STATUS', 'Open')` instead of [Account Id] is not null.

Invoking Signals from Controls and Custom Script

Siebel order management business components route unrecognized InvokeMethod calls to the Context Service business service's RaiseSignal method. All business components of class CSSBCOrderMgmtBase, CSSBCPecBase, and their subclasses, support this routing.

Note: CSSBCPecBase only supports standard signals. It does not support custom signals.

You can invoke signals from controls, such as buttons. You can also invoke signals from a script.

Invoking Signals from a Button

Use the following procedure to invoke signals from a button.

To invoke a signal from a button

1. In Web Tools, create or open a workspace and then navigate to Object Explorer.

   To use the workspace dashboard, see *Using Siebel Tools* .
2. Click Applet and then locate the applet you must modify.
3. Expand the applet list in Object Explorer and click Control.
4. Set the MethodInvoked property of the control to the signal name.
5. Save your changes using the gear icon and submit the workspace for delivery.

Invoking Signals from a Script

Use the following procedure to invoke signals from a script.

To invoke a signal from a script

1. Access the Siebel Script Editor in Siebel Tools by selecting the affected object in the Object Explorer.
2. Right-click the object, and choose Edit Scripts.
3. Modify the script to execute the InvokeMethod method on the appropriate C/OM business component, passing the signal name as the MethodName input argument.

   For example:

   ```
   pQuoteBC.InvokeMethod("Calculate Tax");
   ```

Using Recursion with Signals

Recursion of signals is supported, but you cannot use recursive variable map APIs such as GetRowSet and SyncRowSet in recursed signals. If your recursive signal calls recursive variable map APIs, you will receive an error message. When this happens, you must revisit the definition of the signal and make modifications to make sure these variable map APIs are not involved (for example, you might remove GetRowSet and SyncRowSet, or instead add a CanInvoke method to skip the signal).

Recursive variable map APIs are not supported because these APIs read data from, or write data to, the database. This kind of recursive read and write is not safe.

For details on how to use signal properties and profile attributes when making sure your recursion works properly with signals, see *Modifying Signal Properties for Signal Actions*.

Migrating Signals Between Environments

Signals can be moved between environments, such as from the development environment to the test environment, by using the Application Deployment Manager (ADM). For information about using ADM, see *Siebel Application Deployment Manager Guide* .

You can also export a specific version of a signal using the Export Version applet menu in the Signal Version list applet. To import a signal, navigate to the Administration - Products screen, then the Joint Workspace view. This is a joint workspace for all types of versioned objects (signals, variable maps, products, product attributes, and so on). For more information about import and export, see *Siebel Product Administration Guide* .

4 Variable Maps

Variable Maps

This chapter explains how variable maps are used by PSP procedures to handle transactional data. It includes the following topics:

- *About Variable Maps*
- *Components of Variable Maps*
- *Supported Source Types for Variables*
- *About Using Variable Maps*
- *Variable Map Methods of the Context Service Business Service*

About Variable Maps

Siebel order management applications that use the PSP engine—such as for pricing, eligibility, and product recommendation—require a consistent way of loading, querying, and synchronizing transactional data. For example, the Quote Item and Order Entry - Line Items business components represent fundamentally the same concept, but can use different field names to represent the same value. Variable maps meet this requirement.

Variable maps are also used to extend the capabilities of customizable product linked items. A linked item can now refer to a value in a Context property set constructed by the variable maps mechanism. For more information about linked items, see *Siebel Product Administration Guide* .

Variable maps provide a mechanism for mapping transactional data to a common namespace regardless of the data source. PSP procedures rely on variable maps to map the name of a variable used by a PSP procedure to a field in a Siebel business component or to an attribute used in attribute pricing.

The variable map mechanism employs the Context Service business service, which provides a set of APIs for constructing a property set from the current ordering context and synchronizing changes to that property set back to the source. You can configure the set of data queried and written by a particular transaction.

> **Note:** The variable map APIs work only during an event triggered on a business component derived from CSSBCOrderMgmtBase, CSSBCPecBase, and their subclasses.

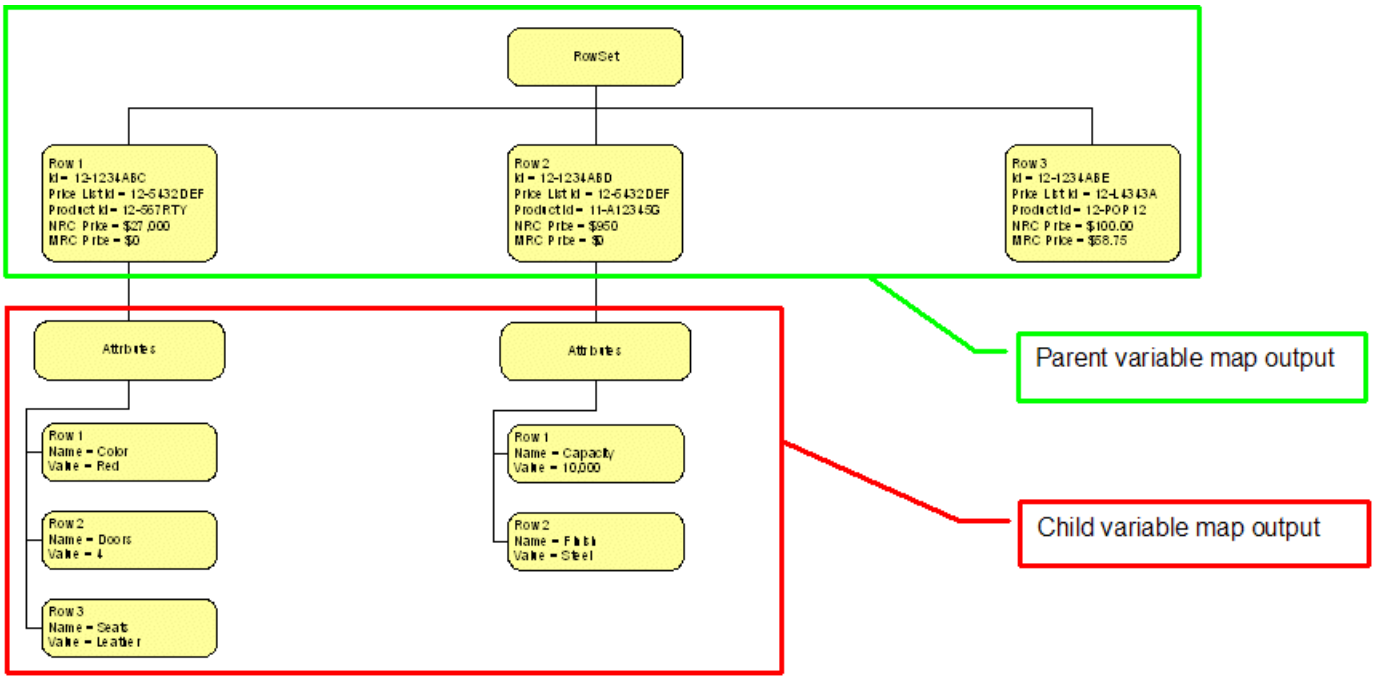
You define the particular variable map used by a PSP procedure in the Signals Administration views (navigate to the Administration - Order Management screen, then the Signals view).

Concepts of Variable Maps

A variable is a name-value pair in a property set. A variable map is a definition of how to construct a property set in a given situation and of which changes to save.

Each variable has one or more variable sources that define how to retrieve the variable value in a given mode (such as Quote, Order, or Any). The source type of a variable source can be a business object query, the active business

component instance, a business service, a profile attribute, a system preference, or a server parameter. A child variable map is another variable map that is executed for each row retrieved by the current variable map and attached as a child property set. A Business Service source can also construct a child property set for each row. The following figure shows example child variable map output (Row1 Attributes - RowA, RowB, RowC; Row2 Attributes - Row1, Row2), in relation to the parent variable map output (RowSet - Row1, Row2, Row3).



Variable Map Types

There are three types of variable maps:

- **Context.** Loads a single row containing shared header-level details (such as Channel, Account Type, User Role).
- **Row Set.** Loads any iteration of rows (such as order line items or shipments for an order).
- **XA.** Loads product attributes for a line item.

Working with Variable Maps

You create variable maps using the Administration - Order Management screen, then the Variable Maps views. Here you define the variables that are queried and written in various situations.

To access the Variable Maps views, navigate to the Administration - Order Management screen, then the Variable Maps view, and in the Versions list applet, click the Work Space hyperlink. Variable Maps views include the following:

- **Variable Maps view.** Create new variable maps, update existing variable maps, release a variable map into production.
- **Child Variable Maps view.** Define child variable maps to be executed and attached to the parent. These are returned as a child property set of each row in the parent property set.

- **Modes view.** Define modes for variable maps. Set the mode according to the Mode user property on the business component invoking the signal that causes the variable map to be executed.

To configure mappings from business component field names to the variable map namespace, use the Integration Object object type in Web Tools. For more information, see *About Using ISS Integration Objects with the Variable Map Mechanism*.

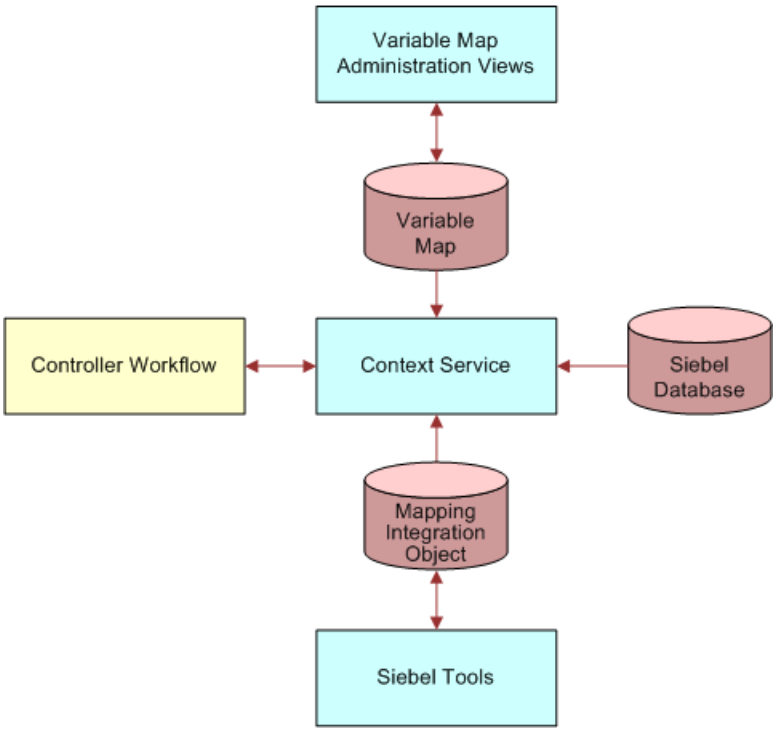
Components of Variable Maps

See the following figure for a graphical representation of the interaction of variable map components. The way the variable map mechanism works follows the following process:

1. An administrator defines a variable map using the Variable Maps views. This definition is stored in the Siebel database.
2. A user of a Siebel order management application makes a request (for example, by clicking the Reprice button). The request triggers a signal, which in turn launches a controller workflow.
3. The controller workflow invokes the Context Service business service's GetRowSetData method, passing the variable map name for the line item row set, as well as the required CPScope (such as the entire customizable product [CP]) and the required RowScope (such as the currently selected rows).
4. The Context Service business service retrieves the variable map definition (either from the database or from the cache).
5. The Context Service business service issues the required queries and business service calls to construct a property set.

   For source type Instance, the path specified for the variable is translated into a query against the active business component using one of the ISS mapping integration objects. For more information, see *About Using ISS Integration Objects with the Variable Map Mechanism*.
6. The Context Service business service returns the resultant row set property set to the controller workflow.
7. The controller workflow invokes a PSP procedure to update the row set (for example, to attach prices).
8. The controller workflow invokes the Context Service business service's SyncRowSetData method passing the variable map name for the line item row set and the updated row set property set.
9. The Context Service business service writes any updated field values back to the Siebel database.

The following figure shows how, in the process described in these steps, the various components of the variable maps mechanism interact.



About Using ISS Integration Objects with the Variable Map Mechanism

A schema of a particular entity, an integration object is metadata; it is a generalized representation or model of a particular set of data. A Siebel integration object is an object stored in the Siebel repository that represents a Siebel business object. An ISS integration object is a special type of Siebel integration object used exclusively within Siebel Order Management.

Generally, a Siebel integration object is used by Siebel EAI to transfer data between Siebel objects and external objects. ISS integration objects, on the other hand, are used to create mappings between business components and their Uniform Name mappings. In this way, while ISS integration uses the Siebel EAI structure used by a Siebel integration object—that is, a structure designed for data transfer—its main purpose is different from that of Siebel integration objects.

Using ISS integration objects, you can apply the same uniform names, such as “Line Item”, to different business components (such as Quote Item, Order Item, and so on), because there is no difference between these business components as far as the PSP engine is concerned.

For general information about integration objects and how to build them, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

For information about configuring ISS integration objects, see *Configuring ISS Integration Objects*.

Configuring ISS Integration Objects

Integration object configuration is used to create uniform field names for different physical business components (such as Quote Item, Order Item, and so on). When defining variable maps then, you can specify one source path for all modes.

> **Note:** To avoid errors when trying to retrieve data from business components, it is recommended that you always define the component field names in integration objects, and use the uniform name in the variable source path for necessary modes only.

At run time, the user property of the business component that raised the signal determines which integration object to use. For each business component, there are various user properties (such as the property called Instance Uniform Name EAI Object:Catalog).

To configure ISS integration objects

1. For the business component that raises the signal, set the value of the user property Instance Uniform Name EAI Object:[BusObj] to the integration object name, for example:

   ```
   Instance Uniform Name EAI Object:Catalog
   ```

   In this example, Catalog refers to the business object that the business component is in at run time.
2. From the integration object definition, use the value in Integration Object Component Name as the Instance Type.

   For example, use Header for the Quote business component in the Catalog business object.

   > **Note:** For each business object, each business component within it must have a unique Instance type—for example, no two business components within the Quote business object can be referred to as Header.

Supported Source Types for Variables

A variable can be derived from a number of sources. The same variable can be derived in different ways, depending on the Mode user property of the business component invoking the signal that causes the variable map to be executed.

For example, the Quote Item business component has Mode set to Quote. Using Mode set to Quote, you retrieve the Product ID from the [Product ID] field in the Quote Item business component for the current quote. Using a different setting, Mode set to Configuration, you retrieve the [Product ID] from the [Product ID] value in the Line Items property set of the product instance currently being configured.

The default setting is `Mode = Any`.

> **Note:** Variable maps work only when invoked by a signal.

You define the driving integration object component for each business component's mode by navigating to the Administration - Order Management screen, Variable Maps, then the Modes view. The driving integration object component for a mode is the component over which the variable map iterates to generate an output property set.

You set the mode using the user property called *Mode*. Existing modes are: Asset, Quote, Order, Payment, Product, Agreement, Configuration.

Note: You can configure your own modes. See *Creating Variable Maps*.

A variable can be derived from the sources listed in the following table.

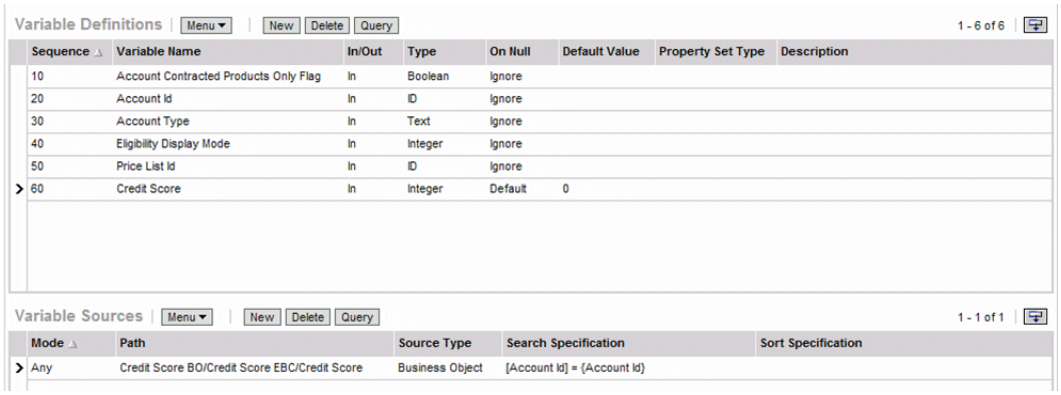
Source Type	Path	Example
Business Object	`[BOName]/[BCName]/[FieldName]`	`Account/Account/Region`
Business Service	`[BusSvcName]/[BusSvcMethod]`	`ABC Assets BS/Get External Assets`
Instance	$Current/[IntegrationObjectComponentType]/[IntegrationObjectComponentField]	`$Current/Line Items/Quantity`
Profile Attribute	`ProfileAttributeName`	`AnonymousUserZipCode`
System	`[LoginName], or [LogInId], or [LogInPassword], or Today`	`Today`
Server Parameter	`[ServerParameter]`	`PARAM_PSP_ELIGIBILITY_DISPL_MODE`

Business Object

The Business Object source type, shown in the Variable Sources list applet in the following figure, is used to query business components that are not in the current context.

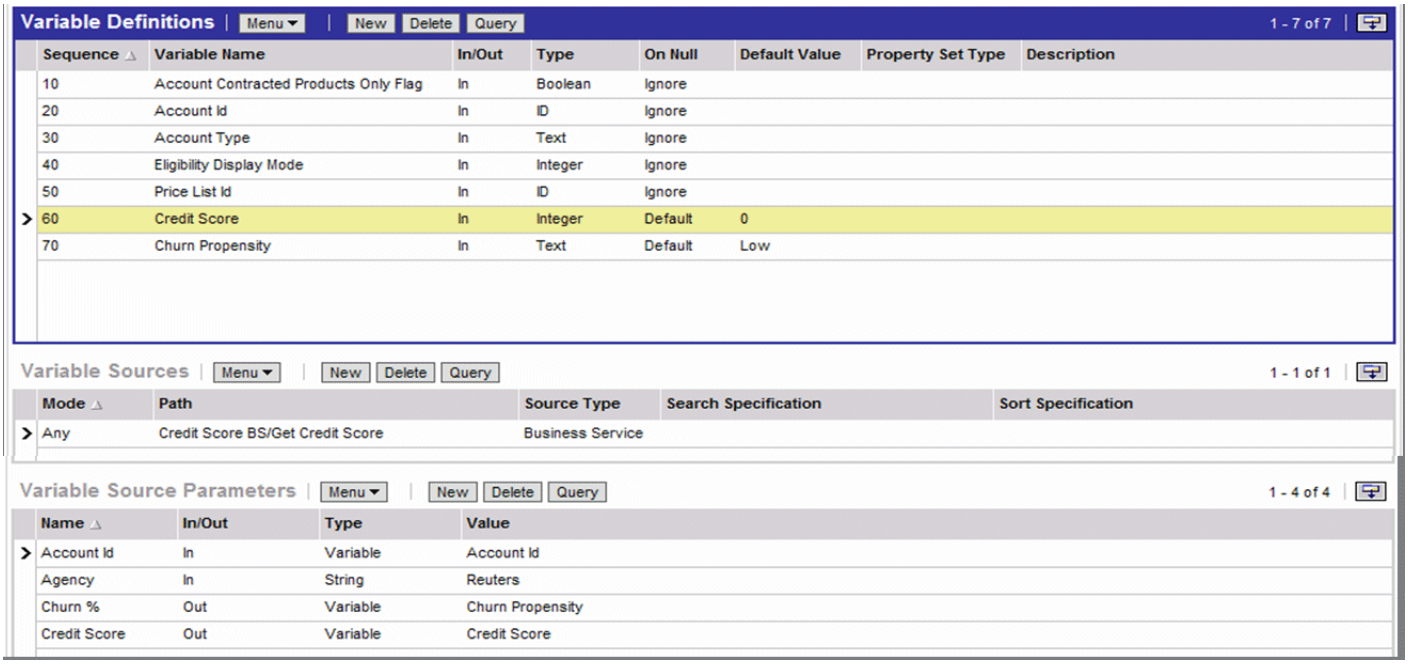
Note: You can also query for joined business components (such as Account and Address).

For more information, see *Querying with the Business Object Source Type*.



Business Service

Used with business services (such as Projected Asset Cache), the Business Service source type allows you to invoke a business service to populate one or more variables, and to populate a child property set of the current row. The following figure shows the Business Service source type.



Using the Business Service source type, you can populate multiple variables or child variable maps in a single method invocation. This is possible if the variables are invoking the same business service and method with the same inputs. This consolidated call to the business service can happen regardless of the number of variables (that is, Property Set, another one, or a mix of two) needing to be populated.

> **Note:** Make sure the values for the Sequence field contain appropriate numbers. In the example shown in the figure, Account Id is an input for the business service. This input uses the value of variable Account Id, therefore the Sequence value for Account Id must be smaller than the Sequence value for Credit Score.

For details on how to populate multiple variables or child variable maps in a consolidated call to the business service, see *Using the Business Service Source Type to Populate Variables*.

Instance

The Instance source type can be used to refer to both of the following:

- The current UI context when viewing quotes and orders
- The current customizable product instance being configured

Used for Siebel order management business components (such as Order Line Item), the Instance source type, shown in the Business Component User Properties list applet in the following figure, allows you to query active UI business components to retrieve variable values. The business component queried can be a regular, external, or virtual business component.

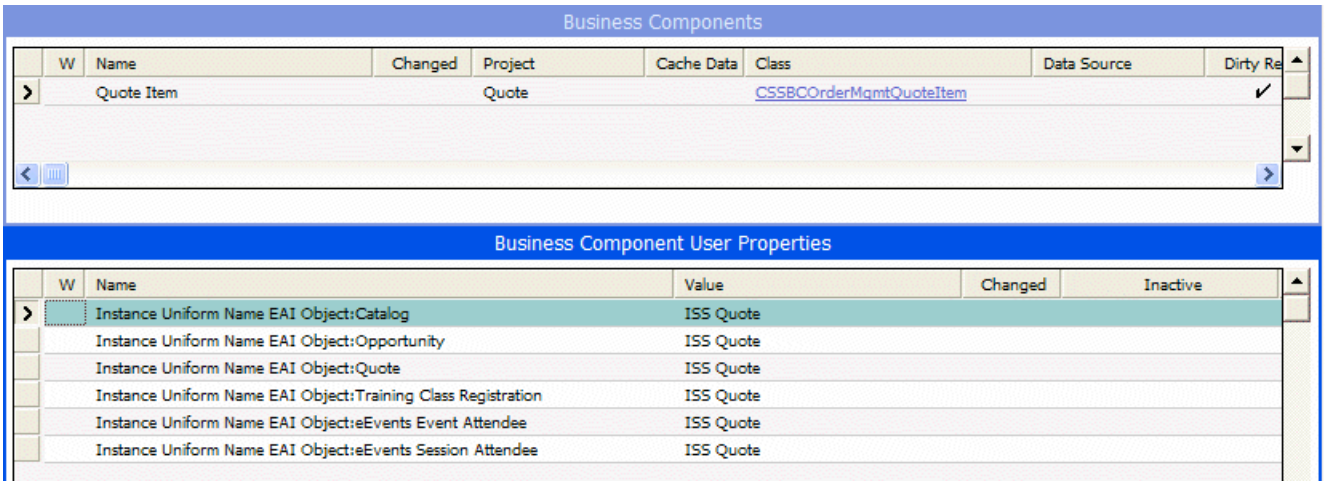
Used for customizable product instance property sets, the Instance source type retrieves data from the business component initiating the signal that causes the GetRowSetData method to be called. Data can also be retrieved from any parent or child business component.

Mapping Integration Objects

For business components, the Instance source type uses mapping of integration objects to resolve the different business component naming and field naming between Quotes, Orders, Assets, and Agreements. You map integration objects using Web Tools.

For a customizable product with Instance source type, the instances are loaded by Configurator services and the structure of these instances is hierarchical with three types: Header, Line Item, and XA. No other types are supported for a customizable product. The namespace mapping is a simple match between the type specified in the variable source path and the customizable product Instance type.

For more information about using the Instance source type, see *Using the Instance Source Type for the Customizable Product Instance Property Sets*.



The following topics include further information about creating and using variable maps:

- *About Variable Maps*
- *Components of Variable Maps*

- *Supported Source Types for Variables*
- *About Using Variable Maps*
- *Variable Map Methods of the Context Service Business Service*

Note: For variable map information that is specific to pricing, see *Siebel Pricing Administration Guide* .

About Using Variable Maps

The following topics provide information about how variable maps are used and defined:

- *Querying with the Business Object Source Type*
- *Using the Business Service Source Type to Populate Variables*
- *Using the Instance Source Type for the Customizable Product Instance Property Sets*
- *Creating Variable Maps*
- *Defining the Variable Map Used by a PSP Procedure*
- *Migrating Variable Maps Between Environments*

Querying with the Business Object Source Type

You can query a business component for values by using the Business Object source type.

To query a business component to retrieve variable values

- In the Variable Sources list applet, set the following fields:

Field	Value
Source Type	“Business Object”
Path	[Business Object]/[Business Component]/[Field Name]
Search Specification	[Business Component Search Spec]
Sort Specification (Optional)	[Business Component Sort Spec]

The Search Specification can include any previously evaluated variable value in {}. Use the Sequence column to provide a correct evaluation sequence.

Using the Business Service Source Type to Populate Variables

The Business Service source type allows you to invoke a business service to populate:

- One or more variables as described in *Invoking a Business Service to Populate Variables*.
- A child property set of the current row as described in *Invoking a Business Service to Populate a Child Property Set*.

Invoking a Business Service to Populate Variables

Use the following procedure to invoke a business service to populate variables.

To invoke a business service to populate variables

1. In the Variable Sources list applet, set the following fields:

Field	Value
Source Type	"Business Service"
Path	[BusSvcName]/[BusSvcMethod]

2. In the Variable Source Parameters list applet:
   a. Add variable source parameters for each input argument. The variable source parameters can be a literal string or another variable value.
   b. Add variable source parameters for each output argument, and specify which variable to populate.

Invoking a Business Service to Populate a Child Property Set

Use the following procedure to invoke a business service to populate a child property set of the current row.

To invoke a business service to populate a child property set of the current row

1. In the Variable Definitions list applet, set Type (the variable type) to `Property Set`.
2. In the Variable Sources list applet, set the following fields:

Field	Value
Source Type	"Business Service"
Path	[BusSvcName]/[BusSvcMethod]

3. In the Variable Source Parameters list applet:
    a. Add variable source parameters for each input argument. The variable source parameters can be a literal string or another variable value.
    b. Add variable source parameters for each output argument and specify which variable to populate.

Using a Single Invocation to Populate Multiple Variables or Child Variable Maps

Use the following procedure to populate multiple variables or child variable maps using a single invocation.

To populate multiple variables or child variable maps in a single method invocation

- Specify the Source and all the In or Out parameters under a single variable.

  It is recommended that other variables have only a definition, no source. This reduces the burden on the Context Service because a separate call to the business service will be issued if there is a second variable having the same business service source.

  > **Note:** A best practice is to compare the source and source parameters from different variables to determine whether to consolidate. If the values of all these variables can be obtained through a single call to the business service, combine them and only specify the business service or method as a source under one variable. Otherwise, the same call will be issued multiple times, giving the same result each time.

Using the Instance Source Type for the Customizable Product Instance Property Sets

If you are working with customizable product instance property sets, use the Instance source type. The Instance source type retrieves data from the business component initiating the signal that causes the GetRowSetData method to be called. Data can also be retrieved from any parent or child business component.

To define the data element to be retrieved

- In the Variable Sources list applet, set the Path to:

  ```
  $Current/[IntegrationObjectComponentType]/[IntegrationObjectComponentField]
  ```

  Examples:

  ```
  $Current/Header/Price List Id
  $Current/Line Item/Quantity
  ```

  Uniform component and field names are defined by integration objects: ISS Quote, ISS Order, ISS Agreement, and ISS Asset.

  Each of these integration objects defines the specific business components and fields that provide data for a generic value such as Line Item or Quantity. For example, ISS Quote integration component Line Item maps to

the Quote Item business component, whereas ISS Order has the Line Item integration component mapped to Order Entry – Line Items.

> **Note:** To expose a custom business component field in a variable map, you must first add it to the corresponding ISS [XXX] integration object.

The ISS integration objects associated with a particular business component are defined by user properties on the business component, for example: Instance Uniform Name EAI Object: [Business Object].

> **Note:** Make sure that you create a user property for every business object in which the business component can be exposed.

Creating Variable Maps

You create and modify variable maps in the Variable Maps views of the run-time client. Use the following procedure to implement a new variable map.

To implement a new variable map

1. Navigate to the Administration - Order Management screen, then the Variable Maps view.
2. In the Variable Maps list applet, create a new record.
3. Give the variable map a name.
4. Lock the variable map by checking the Locked Flag field.

   This locks the object for your user ID.
5. In the Versions list applet, click the Work Space hyperlink to drill down on the variable map version.
6. Click the Modes tab to access the Modes view.
7. In the Modes list applet, define the variable map modes.
8. Click the Details tab and create variable definitions and variable maps.

   a. In the Variable Definitions list applet, you can define the list of variables in the variable map. These names are independent of the source.

      The In or Out field defines whether the variable map can update the variable.

   b. In the Variable Sources list applet, you can define the source of the variable for each mode.
9. Define child variable maps, as necessary.
10. Navigate back to the Variable Maps list applet and click the Release New Version button to release the variable map version.
11. Test the variable map in the run-time client by executing a reprice or another PSP procedure.
12. Using Application Deployment Manager (ADM), promote the updated signal definition to the production environment.

    For information about ADM, see *Siebel Application Deployment Manager Guide* .

Updating an Existing Variable Map

Use the following procedure to update an existing variable map.

To update an existing variable map

1. Navigate to the Administration - Order Management screen, then the Variable Maps view.
2. Select the variable map and lock it by checking the Locked Flag field.

   This locks the object for your user ID.
3. In the Versions list applet, click the Work Space hyperlink to drill down on the variable map version.
4. Modify the variables as necessary.
5. Navigate back to the Variable Maps list applet and click the Release New Version button to release the variable map version.
6. Test the variable map in the run-time client by executing a reprice or another PSP procedure.
7. Using Application Deployment Manager, promote the updated signal definition to the production environment.

   For information about ADM, see *Siebel Application Deployment Manager Guide* .

Configuring a Custom Mode User Property for a Business Component

Use the following procedure to configure a customer Mode user property for the driving business component.

To configure a custom Mode user property for the driving business component

1. On the business component that raises the signal, set the Mode user property.
2. In the Variable Maps, then Modes view, declare the new mode for one or more variable maps.
3. Define variable sources for the mode.

Behavior of the On Null Property When Defining Variables in a Variable Map

When defining variables in a variable map, note that the *On Null* property behaves as follows:

- If a path is specified for a variable, then that variable appears in the Row Set with whatever value is retrieved. In this instance, the On Null property is an empty string.
- If no path is specified for a variable and the On Null property is set to Ignore, then the variable is not included in the Row Set.
- If no path is specified for a variable, the On Null Property is set to Default, and a default value is specified, then the variable is included in the Row Set with the default value.

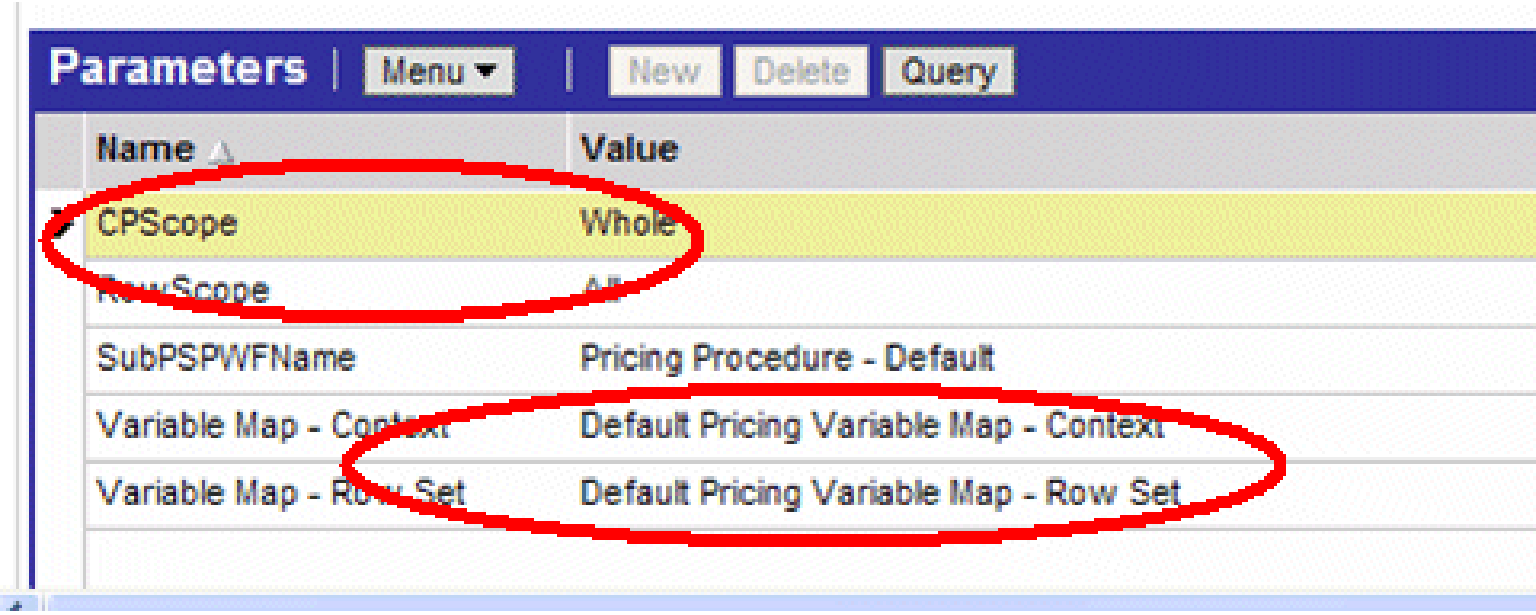
Defining the Variable Map Used by a PSP Procedure

Variable maps are used by the context service to create the property sets that are used by PSP Procedures. You specify the variable maps used by a PSP procedure in the Parameters list applet of the Administration - Order Management screen, then the Signals view.

> **Note:** Certain methods of the Context Service business service include variable map arguments for these definitions. See *Variable Map Methods of the Context Service Business Service*.

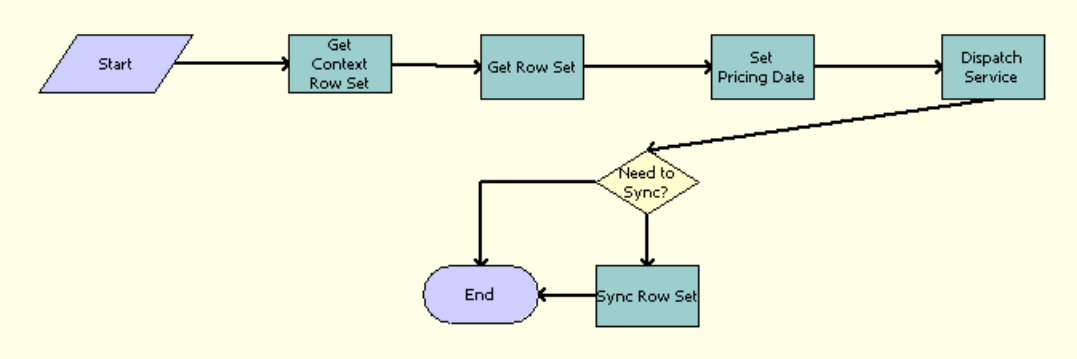
The PSP procedure is independent of the calling context. Most order management signals invoke the PSP Driver Workflow Process (shown in the second figure in this topic), which is a generic controller workflow. The controller workflow uses variable maps to construct inputs for the PSP procedure. Those variable maps are defined in the signal that invokes the controller workflow.

Signal parameters define the scope of line items retrieved using the variable map, and they define the variable maps used to retrieve Context and Row Set data. The following figure shows example signal parameters.



Example of Variable Map Methods in Use

The following figure shows the PSP Driver Workflow Process, which is a generic example of a controller workflow. PSP Driver Workflow Process uses variable maps to retrieve data that is then synchronized back to the database.



As shown in the previous figure, the PSP Driver Workflow Process steps perform as follows:

- **Get Context Row Set.** Calls the Context Service business service method GetRowSetData to retrieve header-level information using the Context variable map (Variable Map - Context).
- **Get Row Set.** Calls the Context Service business service method GetRowSetData to retrieve row-level information using the Row Set variable map (Variable Map - Row Set) and to scope input arguments.
- **Dispatch Service.** Calls the sub workflow process defined in the calling signal and passes in the input argument.

- **Set Pricing Date.** Sets pricing date based on whether it is for scenario testing:
  - Set system time stamp if it is not for scenario testing:

    ```
    {Context.Effective Pricing Date} = TimeStamp()
    ```

  - Set Active JWS Test DT if it is for scenario testing:

    ```
    {Context.Effective Pricing Date} = GetProfileAttr("Active JWS Test DT")
    ```

- **Sync Row Set.** Calls the Context Service business service method SyncRowSetData to write any updates back to the database (for example, updates to prices).

To define the variable map used by a PSP procedure

1. Navigate to the Administration - Order Management screen, then the Signals view.
2. In the Signals list applet, select the signal that will invoke the PSP procedure.
3. In the Versions list applet, click the Work Space hyperlink to access the Actions list.
4. In the Actions list applet, enter the name of the controller workflow in the Service Name field.

   Note: Most PSP signals invoke the generic PSP Driver Workflow Process.

5. In the Parameters list applet, (scroll down, if necessary), enter parameters for the signal, as follows:
   - Using Scope arguments and values, define the scope of line items retrieved by the variable map.
   - Using Variable Map parameters, define the variable maps used to retrieve Context and Row Set data.

Migrating Variable Maps Between Environments

You can move variable maps between environments (such as from development to test) by using the Application Deployment Manager (ADM). For information about using ADM, see *Siebel Application Deployment Manager Guide* .

You can also export a specific version of a variable map using the Export Version applet menu in the Variable Map Version list applet. To import a variable map, navigate to the Administration - Products screen, then the Joint Workspace view. This is a joint workspace for all types of versioned objects (signals, variable maps, products, product attributes, and so on). For more information about import or export, see *Siebel Product Administration Guide* .

When exported, a variable map and its child variable maps are exported together into an XML file. When imported through the joint workspace, both parent and child variable maps will be imported and listed in the Joint Workspace view.

Note: You must go to each of these variable maps separately to release them.

Variable Map Methods of the Context Service Business Service

The Context Service business service provides the APIs shown in the following table for variable maps.

Method	Arguments	Description
GetRowSetData	[in] CPCollapseAll: String	By default, if the CPScope requires expansion of the customizable product to read all products, after GetRowSetData, this customizable product is expanded on the UI unless this flag is set to be true, in which case the customizable product is collapsed.
	[in] CPScope: String	Component, Master, Whole, Component, and Subcomponents. This argument defines which parts of the current customizable product are queried.
	[out] PropSet: Hierarchy	A row set property set containing the query results.
	[in] RowScope: String	Current, Selected, or All. Defines which rows the Context Service service will read data from.
	[in] VariableMap: String	The variable map defining objects to query.
SyncRowSetData	[in] RowSet: Hierarchy	The updated row set property set.
	[in] VariableMap: String	The variable map defining objects to update.

5 PSP Engine

PSP Engine

This chapter describes the Product Selection and Pricing (PSP) engine and explains the how-to aspects of working with PSP. It includes the following topics:

- *About the Product Selection and Pricing Engine*
- *Components of the PSP Engine*
- *PSP Driver Workflow*
- *Conditions and Actions for PSP Procedures*
- *About Temporary Variables*
- *Row Set Transformation Toolkit Methods*
- *Configuring PSP Procedures*
- *Creating a Custom PSP Application*
- *Calling a PSP Procedure from an External Application*
- *About Logging of PSP*
- *About Troubleshooting of PSP*
- *About Tuning Performance of PSP*

About the Product Selection and Pricing Engine

The Product Selection and Pricing (PSP) engine is a generalized procedural logic engine for transforming an input row set into an output row set. PSP is an extension of Siebel Workflow. A PSP procedure is a workflow process that includes a Business Service step employing methods of the Row Set Transformation Toolkit business service.

A PSP procedure transforms a set of input rows into a set of output rows by executing matrix lookups, conditional logic, and external function calls. In this release, the functions of pricing, eligibility (product, attribute, and promotion), and product recommendation use the PSP engine.

For example, Siebel Pricer uses PSP procedures to apply all of the different types of discounts that are available with a particular product. Because these discounts are based on a PSP procedure rather than on C++ code, you can:

- Change the order in which discounts are applied.
- Customize the calculations used by discounts.
- Extend the preconfigured pricing PSP procedure to calculate additional costs or prices and margin.

Advantages of PSP Usage

The following are some of the benefits of using PSP procedures as a basis for Siebel order management tasks:

- Highly configurable procedural logic eliminates or reduces the need for custom script.

- PSP methods and infrastructure are optimized for performance (for example, with set-based processing, caching, and SQL query consolidation).
- The PSP framework can be extended for use with external services and functions.
- Integrators can learn this one framework and use it for pricing, eligibility, and so on.

PSP Concepts

A PSP procedure is the sequence of steps involved in transforming an input row set into an output row set. Examples of steps include a call to a business service, an instance of a transform, a conditional branch, a subprocedure call, or a terminator (an end step). A PSP procedure is any workflow that uses methods from the Row Set Transformation Toolkit business service.

A *controller workflow* is the invocation mechanism for the PSP engine. A PSP procedure is always called by a controller workflow. A controller workflow retrieves contextual information, invokes a generic PSP procedure, and then processes the results. It insulates the underlying PSP logic (such as a pricing procedure) from the calling context (such as repricing a quote or pricing an XML order passed in through a Web service). For more details on controller workflows, see *Controller Workflow*.

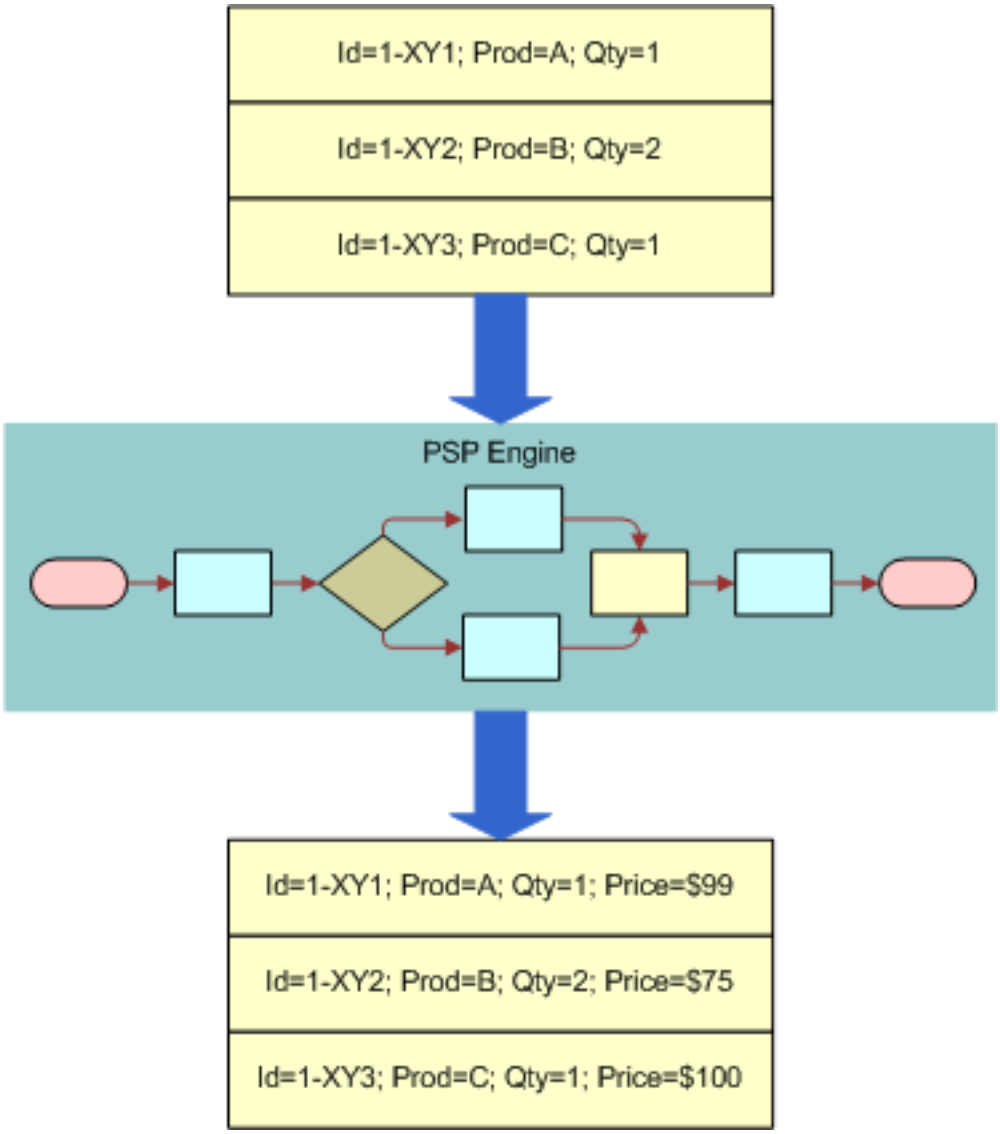
A row set is a property set that conforms to the structure defined in *About Row Sets*. It is used to represent the set of data upon which the PSP engine operates (such as data for quote line items).

The Row Set Transformation Toolkit is a business service that exposes a set of methods called transforms. Transform methods are called by steps within a PSP procedure. A *transform* accepts one or more input row sets, performs a series of operations (such as database queries), and then returns an updated version of the row sets as output. Special step input arguments called actions define the processing performed by a particular step. An action can perform a wide variety of updates to the input row set (such as setting the Net Price field value). Most transforms have a defined set of transform conditions that occur while the transform is executing (for example, the Simple Look-Up transform queries the database and then raises one or more of the following conditions: On First Match, On Match, On Last Match, On No Match). The condition raised depends on the result of the SQL query. Actions are attached to these conditions.

How PSP Procedures Are Built

PSP procedures are created in the same way that standard workflows are created, in the Business Process Designer. The Process Designer is a user interface to help you arrange process objects. You access it from the Workflow Process object in Siebel Tools. For more information about building workflows, see *Siebel Business Process Framework: Workflow Guide* .

Like a standard workflow, a PSP procedure has a start step and an end step. A PSP procedure differs from a standard workflow in that the steps of a standard workflow perform actions, while the steps of a PSP procedure transform row sets in some way, as shown in the following figure. In the following figure, a set of product information that includes data on product IDs, names, and quantities is transformed into a new set of information that includes an additional product as well as pricing information for each product.

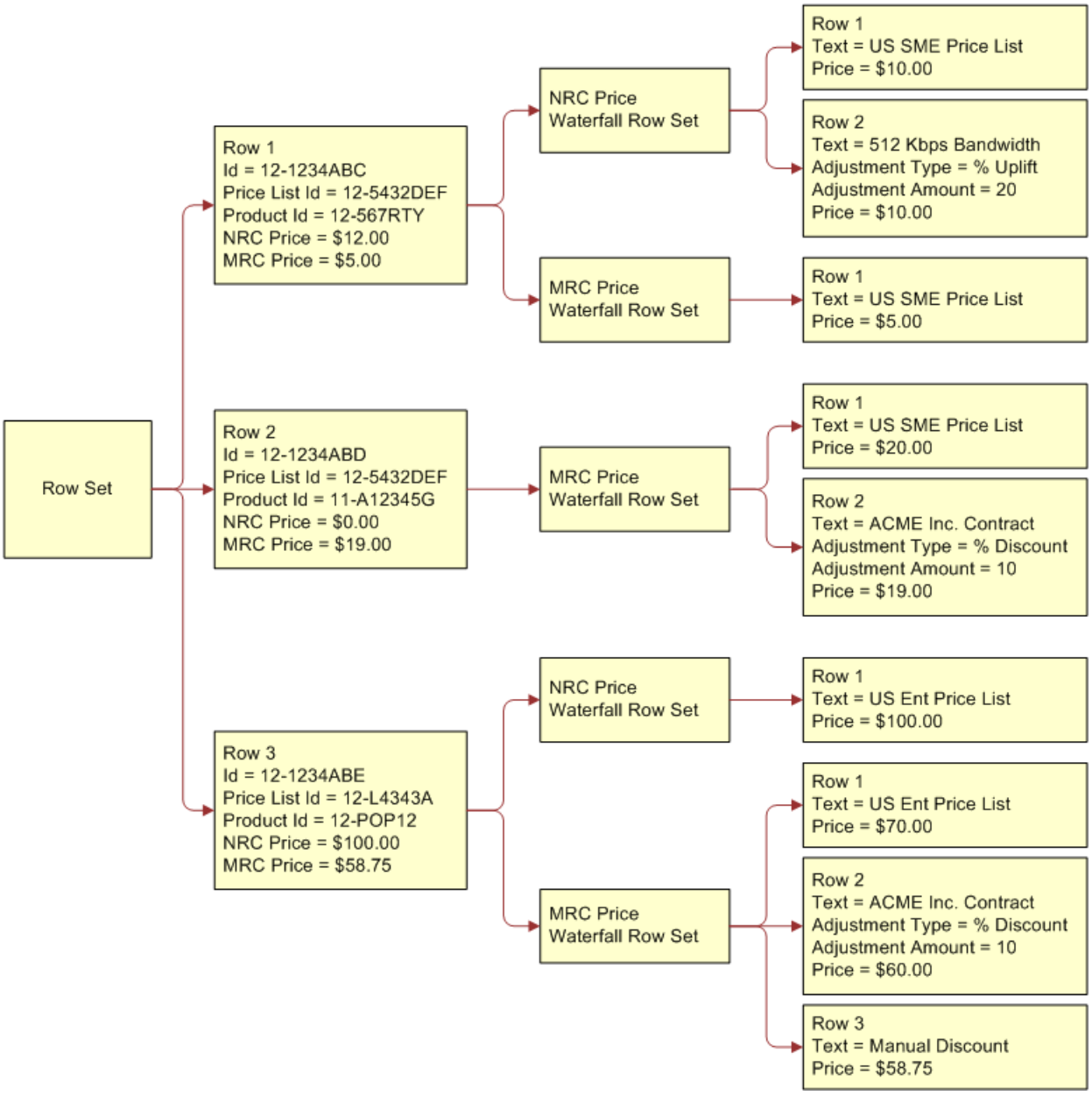


The PSP engine transforms row sets using the methods in the Row Set Transformation Toolkit business service. For more information about these methods, see *Row Set Transformation Toolkit Methods*.

About Row Sets

A type of property set, a row set is a memory structure used to pass data between business services. A row set is a group of rows where each row contains multiple name-value pairs (paired values). A row within a row set can have multiple child property sets with name-value pairs; the hierarchy goes no deeper than these child property sets.

The following figure provides a graphical example of a row set. In this figure, the second layer of boxes labeled "Row 1," "Row 2," and "Row 3" are the rows within this example row set. Each of these rows 1 through 3 have name-value pairs for data labeled "ID," "Price List ID," "Product ID," "NRC Price," and "MRC Price." Additionally, each of these rows 1 through 3 contains child property sets, called "NRC Price Waterfall Row Set" and "MRC Price Waterfall Row Set." The child property sets contain their own rows with name-value pairs, for a deeper level of pricing data.



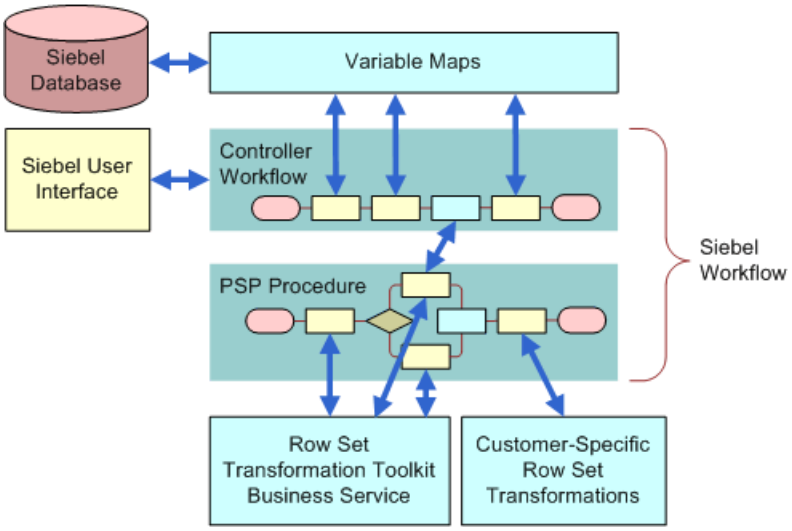
For more information about Siebel property sets, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Components of the PSP Engine

The PSP engine is comprised of the pieces shown in the following figure and described in the following topics:

- *Controller Workflow*
- *Variable Maps*
- *PSP Procedures*

- *PSP, Siebel Workflow, and Siebel Tools*
- *Row Set Transformation Toolkit Business Service*
- *Custom Business Services*



Controller Workflow

A controller workflow invokes the PSP engine every time a PSP procedure is called. The controller workflow insulates the PSP procedure from the calling context. The various calling contexts, such as Siebel Configurator, a product picklist, or a Web service, each have separate controller workflows. The PSP Driver Workflow Process is an example of a prebuilt controller workflow. You can configure your own controller workflows to meet your organization's particular needs.

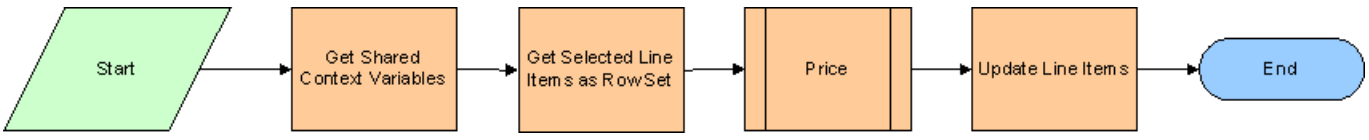
Note: PSP Driver Workflow Process is the generic controller workflow. For more information, see *PSP Driver Workflow*.

While the PSP procedure transforms row sets in memory, the controller workflow passes those row sets to the PSP procedure and then determines what to do with the PSP procedure's output. The typical flow is:

- An end user clicks a button or makes some other choice in the interface.
- This end-user action triggers a signal which executes the controller workflow.
- The controller workflow establishes the inputs for the PSP procedure by finding data and constructing this data into row sets. The controller workflow can use the variable maps mechanism to construct the row sets.
- The controller workflow calls the PSP procedure and passes the inputs to it.
- The PSP procedure transforms the inputs and sends the transformed row set back to the controller workflow.
- The controller workflow determines what to do with these transformed rows. For example, it might display the transformed rows on the screen or write them to the database.

The PSP procedure's only function is to transform row sets in memory. The controller workflow executes any other actions.

The following figure shows an example of a controller workflow.



This controller workflow and its called PSP procedure operate as follows:

1. **Get Shared Context Variables.** This first step (after the Start step) obtains the needed data by using the variable maps mechanism (the Context Service business service) to populate a property set containing context variables shared by all rows (such as Channel, Account Type, or User Role).
2. **Get Selected Line Items as Row Set.** This step instructs the Context Service business service to populate a property set containing the input row set.
3. **Price.** This Subprocess step calls the Price PSP procedure, passing the context variables and the input row set. The Price PSP procedure transforms the row set and passes the values back to the controller workflow.
4. **Update Line Items.** This step, which also uses variable maps, updates the line items with the values from the transformed row set. That is, it saves the newly calculated prices.

Variable Maps

Variable maps, using the Context Service business service, help the controller workflow to construct inputs to PSP procedures and process the output of PSP procedures. The Context Service business service optimizes the querying and updating of row set data by reading data directly from the active business component, thereby eliminating unnecessary SQL queries. Context Service provides a row-level delta that determines which line items to update. Batched SQL eliminates unnecessary network round-trips.

The Context Service business service:

- Constructs the input row sets. Context Service converts business component data, XML, or property set data to a common format. It translates from various name spaces to the PSP name space. It defines the subset of fields required by the PSP procedure.
- Writes the output row set back to its source. Context Service converts from the common format back to the business component data, XML, or property set data. It translates from the PSP name space to the target name space.

> **Note:** Most PSP procedures use the Context Service business service, but it is not required for all PSP procedures. For example, a Web service could invoke Siebel Pricer with a property set directly generated from the input XML document by XSLT, without using the Context Service business service.

For more information about variable maps and the Context Service business service, see *Variable Maps*.

PSP Procedures

A PSP procedure is any workflow that uses methods from the Row Set Transformation Toolkit business service. These methods of the Row Set Transformation Toolkit business service are called PSP transforms. A transform, such as the Simple Look-Up method, processes an input row set. There are a number of transforms that process input row sets in different ways. For example, the Simple Look-Up transform uses a simple search expression to look up each input row in a business component, while the Split transform takes an input row set, evaluates a condition for each of its rows, then splits the input row set into two output row sets.

> **Note:** In addition to invoking Row Set Transformation Toolkit business service methods, a PSP procedure can invoke methods from custom business services.

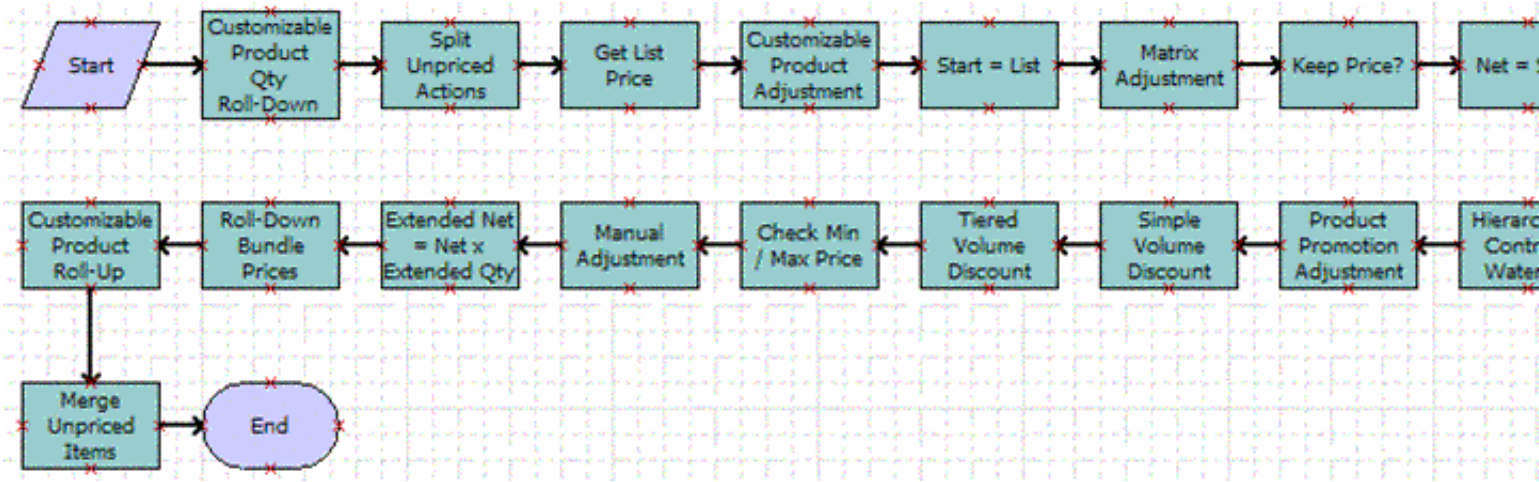
Each step in a PSP procedure is a parameterized call to a transform method. A PSP procedure can call another PSP procedure as a subprocess, to provide for modularization of logic.

The Siebel Business Process Designer interface is used to create both the PSP procedure and the controller workflow that invokes it. Like any workflow process, a PSP procedure can make use of any standard Siebel Workflow feature.

The following figure shows an example of a PSP procedure used in pricing. The steps in this procedure are: *Start, Customizable Product Qty Roll-Down, Split Unpriced Actions, Get List Price, Customizable Product Adjustment, Start = List, Matrix Adjustment, Keep Price?, Net = Start, Execute Line Specific Pricing, Hierarchical Contract Adjustment, Hierarchical Contract Waterfall, Product Promotion Adjustment, Simple Volume Discount, Tiered Volume Discount, Check Min/Max Price, Manual Adjustment, Extended Net = Net * Extended Qty, Roll-Down Bundle Prices, Customizable Product Roll-Up, Merge Unpriced Items, End.*

Each step in this procedure is a parameterized call to a method in the Row Set Transformation Toolkit.

Notice that one of the steps splits the input row set into multiple temporary subsets. Later steps perform logic on these subsets. The last step merges these subsets, so they form a single row set again.



PSP, Siebel Workflow, and Siebel Tools

Siebel Workflow—the application you use to define, manage, and enforce your organization's business processes by creating workflow processes—is also the application you use to create, edit, and execute PSP procedures. Siebel Workflow's Process Designer resides in Siebel Tools.

> **Note:** Siebel Workflow is also known as Siebel Business Process Designer, which is the configuration interface and the administrative interface for Siebel Workflow.

You configure PSP procedures and their controller workflows from the Workflow Process object, a top-level (highest level) object in the Object Explorer within Siebel Tools. In this way, you use the Process Designer to enter transforms for PSP procedure steps as input arguments.

For more information about Siebel Workflow, see *Siebel Business Process Framework: Workflow Guide* . For more information about configuring PSP procedures, see *Configuring PSP Procedures*.

Row Set Transformation Toolkit Business Service

The Row Set Transformation Toolkit is a business service that provides the following methods (also known as transforms) for manipulating and transforming row sets:

- ***Aggregate Method*.** Calculate the minimum, maximum, average, sum, or count of sub-groups of the row set.
- ***Conditional Action Method*.** Evaluate a Boolean expression for each row and perform actions based on the result.
- ***Dynamic Look-Up Method*.** Look up each input row in a business component using a dynamic search expression (example: attribute adjustment).
- ***Dynamic Subprocedure Method*.** Send each input row to the specified subprocedure for individual processing. Each row can be associated with a different subprocedure.
- ***Hierarchical Look-Up Method*.** Look up the closest, best, or accumulated value in an adjustment table for each row by considering each parent in a hierarchy (example: parent company discount).
- ***Hierarchical Method*.** Process a hierarchy of input rows from start to end or end to start (example: customizable product price roll-up).
- ***Merge Method*.** Combine two or more row sets into a single row set.
- ***Query Method*.** Query a business component and generate a row set.
- ***Row Set Look-Up Method*.** Look up each input row in the specified row set (example: check compatibility between a product and the list of products currently owned by the customer).
- ***Rule Set Look-Up Method*.** Look up the rules for each input rule set and test the rules against the row set. Perform actions if the rule set passes or fails (example: identify applicable bundles or promotions).
- ***Simple Look-Up Method*.** Look up each input row in a business component using a simple search expression (example: list price, exclusive eligibility).
- ***Split Method*.** Split an input row set into two output row sets by evaluating a condition for each row.

For each of these methods, you specify a condition and actions, as described in *Conditions and Actions for PSP Procedures*. All the methods support the same action syntax and capabilities. Each method exposes a unique set of conditions and variables.

For details on each of these methods, see *Row Set Transformation Toolkit Methods*.

Custom Business Services

In addition to calling Row Set Transformation Toolkit business service methods, PSP procedures can call custom methods that you write using Siebel VB or Siebel eScript. For more information, see *Siebel VB Language Reference* and *Siebel eScript Language Reference* .

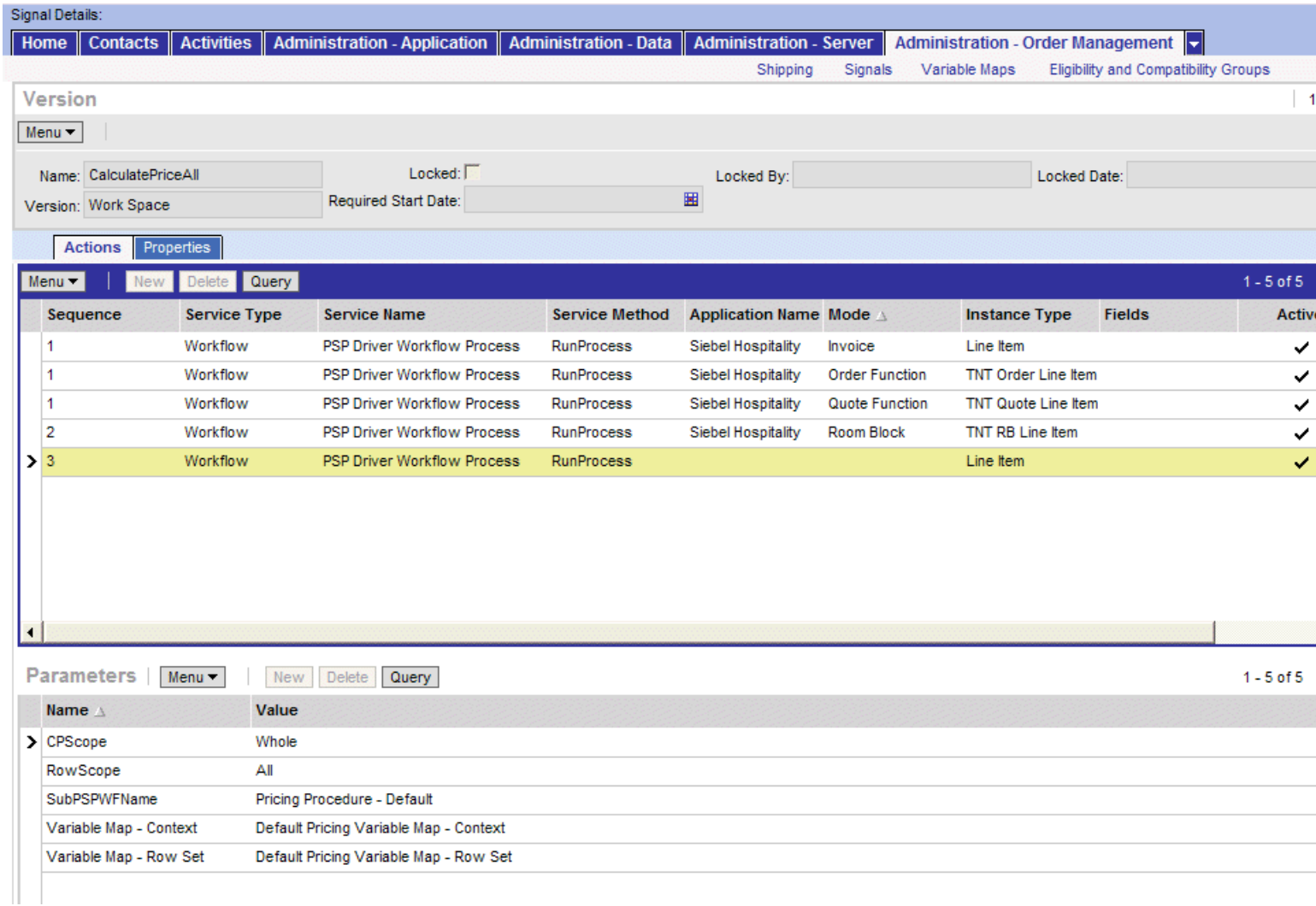
PSP Driver Workflow

A controller workflow is the invocation mechanism for each PSP procedure. The workflow called PSP Driver Workflow Process is the default controller workflow. When a signal calls the controlling workflow for a process, it passes the names of the PSP procedures to the PSP Driver Workflow.

> **Note:** PSP Driver Workflow is the default controller workflow, but you can configure your own controller workflow to replace the default if you find that modifications are necessary for your organization's requirements. You configure a controller workflow in the same way that you configure a standard workflow process. You specify the arguments of a controller workflow in the signal definition, so for your custom controller workflow, navigate to the Administration - Order Management screen, then the Signals view in order to change this definition. For information about configuring workflow processes, see *Siebel Business Process Framework: Workflow Guide* .

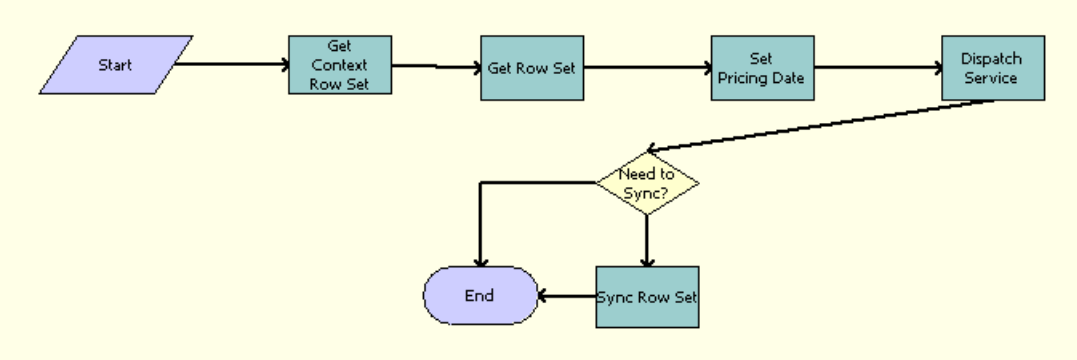
Arguments for the default PSP Driver Workflow are shown in the following figure. PSP Driver Workflow is set as the controller workflow for the signal CalculatePriceAll in the Service Name field of the Actions tab.

The PSP Driver Workflow acts as the controller workflow for the pricing and eligibility PSP procedures. It calls the Context Service to construct a property set called Row Set containing the selected rows from the source object (Quote, Order, Agreement, or Customizable Product). It also calls the Context Service to construct a property set called Context that contains header-level information shared by all rows (for example: Account Type, Credit Score). It then invokes the PSP procedure specified in its input arguments. The PSP procedure updates and returns a new version of Row Set. Finally, the controller workflow instructs the Context Service to save any changes in Row Set back to the source object.



The *PSP Driver Workflow*, shown in the following figure, does the following:

1. Asks the Context Service to generate two property sets (row sets): the shared Context and the Row Set containing individual line items. These are representations of the Line Item and Header business components using variable maps. For more information about variable maps, see *Variable Maps*.
2. Dispatches to the workflow indicated by the process property PSPWorkflowName. For example, in the event the user selected RepriceAll, this step dispatches to the Pricing Procedure - Default workflow to perform all pricing operations.
3. Synchronizes the updated PSP information back to the input data source (Quote, Order, Agreement, or Customizable Product). The synchronization can be skipped by setting the Sync process property.



The following table provides a list of the steps in the PSP Driver Workflow Process, showing also the business service and method called by each step.

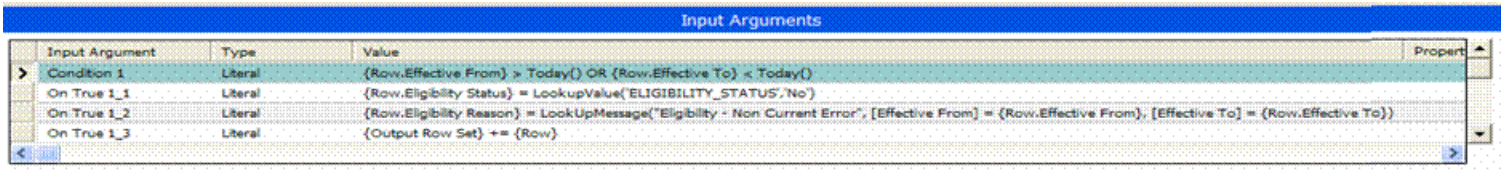
Step Name	Type	Business Service	Method	Description
Get Context Row Set	Business Service	Context Service	GetRowSetData	Generate the Context Property Set which represents the Header Buscomp.
Get Row Set	Business Service	Context Service	GetRowSetData	Generate the Row Set Property Set which represents the Line Item Buscomp.
Set Pricing Date	Business Service	Context Service	GetRowSetData	Set pricing date.
Dispatch Service	Business Service	ISS PSP Dispatch Service	CallPSPWorkflow	Dynamically dispatch to a sub-process.
Need to Sync?	Decision Point	None	None	A user decision point of whether synchronization is needed.
Sync Row Set	Business Service	Context Service	SyncRowSetData	Synchronizes information back to data sources using Context Service.

Conditions and Actions for PSP Procedures

Steps of PSP procedures can call Row Set Transformation Toolkit business service methods. Each Row Set Transformation Toolkit method (transform) performs a parameterized set of actions based on conditions that occur as the method executes.

Conditions and actions for each step are entered as input arguments in Siebel Workflow's Process Designer in Siebel Tools, as shown in an example in the following figure:

- In the Input Argument field, select the condition name.
- In the Value field, enter the processing to perform if the condition is true.
- You can specify multiple actions for one condition by using a different index number for the condition name. In the following table example, there are multiple actions for the On True 1 condition, with the condition name On True 1_1, On True 1_2, and On True 1_3.
- Some Row Set Transformation Toolkit business service methods can include one or more Boolean conditions to which the other conditions refer. In the following figure example, Condition 1 is a Boolean condition which checks the values in the Effective From and Effective To fields. The conditions On True 1_1, On True 1_2, and On True 1_3 are true if this Boolean condition is true.
- Actions are executed in the sequence specified by the index on the name. For example, the action for On True 1_1 executes before the action for On True 1_2.



PSP-Supported Action Expression Constructs

The following table defines the types of action expressions supported by PSP.

Action	Examples

Set a property of a row to the value of an expression	`{Row.Eligibility Status} = LookupValue('ELIGIBILITY_STATUS','No')`
	
	`{Parent.Roll-Up Amount} = ToNumber({Parent.Roll-Up Amount}) + ToNumber({Row.Net Price})`

Remove a property from a row	`{Row} -= {Row.Temp Roll-Up Price}`
Move a row from one row set to another	`{Output Row Set} += {Row}` `{Row Set} += {Output}`
Copy a row to another row set	`{Output Row Set} += Copy({Row})`
Construct a new row and attach it as a child to the specified row set or row	`{Row}.{Waterfall} += New('Waterfall', Text = 'Hello', Value = 10)` `{Row Set} += New('Row', Text = {Match.Text})`
Delete the current row from a row set	`{Row Set} -= {Row}` `{Output Row Set} -= {Output}`

PSP-Specific Functions Used in Action Expressions

Action expressions support the full Siebel Query Language syntax including functions such as LookupValue, IfNull, IIF, and InvokeServiceMethod. For more information about Siebel Query Language, see *Siebel Tools Online Help* .

In addition to Siebel Query Language syntax functions, action expressions can include the PSP-specific functions shown in the following table.

Function	Description
`ToNumber({Row.Qty})`	Convert the specified property value to a number. All values are stored as a string in a property set.
`ToDate({Row.Effective Start Date})`	Convert the specified property value to a date. All values are stored as a string in a property set.
`ToCurrency({Row.Net Price}, {Row.Currency Code})`	Convert the specified property to a currency. All values are stored as a string in a property set.
`AdjustPrice({Row.Net Price}, {Row.Currency Code}, {Match.Adjustment Type}, {Match.Adjustment Amount}, {Match.Currency Code}, {Match.Exchange Date})`	Apply the specified pricing adjustment. This function automatically converts the currency of monetary adjustments to match the currency of the line item.
`LookUpMessage('Pricer Waterfall - Selected Contract Adjustment', [Account] = {Row.Temp Contract Account})`	Retrieve substituted, translated text from the UMS business service. Payload variables are specified as name-value pairs after the message type. For more information about the Unified Messaging framework, see *Unified Messaging* For details on using the UMS business service in a PSP procedure, see *Using Unified Messaging with the PSP Engine*.
`GetXA({Row}, 'Color')`	Get an attribute value for a row. The GetXA method has two different signatures: 1. GetXA({Row},<Attribute Name>) 2. GetXA({Row}, "Name", <Attribute Name>, "LICValue") For more information about the second signature, see *Using the Four-Parameter GetXA Signature*.
`Sum({Row Set}, 'Extended Net Price')`	Sum the value of a field for all children of a property set.
`Avg({Row}.{Shipments}, 'Cost')`	Calculate the average value of a field for all children of a property set.
`Min({Row Set}, 'Score')`	Get the minimum value of a field for all children of a property set.
`Max({Row Set}, 'Score')`	Get the maximum value of a field for all children of a property set.

Function	Description
`Count({Children})`	Count the children of a property set.
`Round({Row.Price}, {Context.Precision})`	Round a number to the specified decimal places.

Using the Four-Parameter GetXA Signature

Use the four-parameter GetXA signature if the attribute that you want to obtain is of type integer, otherwise errors can result in calculations especially where numerous digits are involved.

To use the four-parameter GetXA signature in a workflow, you must add an additional step to retrieve LICValues for the attributes as described in the following procedure.

To use the four-parameter GetXA signature in a workflow

1. Add an additional step to retrieve LICValues for the attributes as follows:
   a. In the workflow editor, create a new "Business Service" before executing the GetXA step.
   b. Set the Business Service Name to "Pricing Manager" and Method to "PopulateAttrLICValue"
   c. Set the following input parameter:

Input Argument	Type	Property Name
Row Set	Process Property	Row Set

   d. Set the following output parameter:

Property Name	Type	Output Argument
Row Set	Output Argument	Row Set

2. Use the following four-parameter GetXA signature:

```
GetXA({Row}, "Name", <Attribute Name>, "LICValue")
```

for example, as follows:

```
{Row.Temp Attr} = ToNumber(GetXA({Row},"Name","Attribute Name","LICValue"))
```

LookUpMessage API

The Unified Messaging framework's UMS business service processes all translations using the LookUpMessage API in a PSP action script. For example, for a pricing waterfall, it might use the following script:

```
{Row}.{Net Price Waterfall} += New('Waterfall', [Text] = LookUpMessage({Row.Temp
List Price Message}, [Price List] = {Match.Price List}), [Currency Code] =
{Row.Currency Code}, [Price] = {Row.List Price})
```

For details on using the UMS business service in a PSP procedure, see *Using Unified Messaging with the PSP Engine*.

Row Set Variables Used in Action Expressions

Action expressions operate on the row set variables shown in the following table.

Variable	Description
{Row Set}	The input row set for the step (specified as an input argument).
{Context}	The input property set of variables shared by all rows (specified as an input argument). {Context} acts as a set of default values for every {Row}. If {Row.Value} is not specified, then PSP automatically returns {Context.Value}. This also works for {Parent.Value} in the Hierarchical transform. If no value is found in {Row} or {Context}, then an error is raised.
{Output Row Set}	The optional output row set for the step. Most steps allow rows to be updated and written to both {Row Set} and {Output Row Set}.
{Row}	The {Row Set} row currently being processed by the transform.
{Output}	The last row added to the {Output Row Set} property set.
{Parent}	The parent row of {Row}. (Hierarchical transform only.)
{Children}	A row set containing the child rows of {Row}. (Hierarchical transform only.)
{Match}	A property set containing name-value pairs from a joined record in a business component or other row set. (Look-Up transforms only.)
{Property Set.Name}	The value of property "Name" in the property set. (Examples: {Row.Net Price} or {Match.Discount %})
{Property Set}.{Type}	The child row set of "Property Set" of type "Type". (Example: {Row}.{Net Price Waterfall})

Conditions and Action Variables Vary by Transform

All Row Set Transformation Toolkit business service methods (transforms) support the same action syntax and capabilities. However, each method exposes a unique set of conditions and variables. For example:

- The Simple Look-Up transform joins each input row to a business component. It exposes On First Match, On Match, On Last Match, and On No Match conditions.
  Actions can reference:
  - Any field in the input row (for example, {Row.Product Id}), or
  - The joined business component (for example, {Match. List Price}).
  - The Hierarchical transform sorts the input row set into a series of tree structures and then navigates each tree from start to end or end to start. It exposes On Leaf Row, On Row, On Parent Row, and On Top Row conditions. Actions can reference:
    - The current row (example: {Row.Roll-Up Price})
    - Its immediate parent row (example: {Parent.Price})
    - Its child rows (example: {Children.Qty}).

For more information about the conditions and actions for each method, see *Row Set Transformation Toolkit Methods*.

About Temporary Variables

You can create temporary variables simply by using a name that is not defined in the Variable Maps.

A temporary variable persists for the life of the property set unless you explicitly delete it using a `{Row} -= {Row.Temp Variable}` action. In general, there is no need to delete temporary variables, because they create little overhead in having a few temp variables.

It is recommended that you use a naming convention for temporary variables, such as beginning each one with Temp (for example, Temp Roll Up Price), to make sure that they do not conflict with the names of other variables.

> **Note:** Variable names are case sensitive. For example, if you try to refer to the {Row.Net Price} variable and you mistakenly use {Row.NEt Price}, the application will not recognize the variable.

Row Set Transformation Toolkit Methods

PSP procedures use the methods in the Row Set Transformation Toolkit to manipulate and transform row sets. The Row Set Transformation Toolkit includes the following methods:

- *Aggregate Method*
- *Conditional Action Method*
- *Dynamic Look-Up Method*

- *Dynamic Subprocedure Method*
- *Hierarchical Look-Up Method*
- *Hierarchical Method*
- *Merge Method*
- *Query Method*
- *Row Set Look-Up Method*
- *Rule Set Look-Up Method*
- *Simple Look-Up Method*
- *Split Method*

Aggregate Method

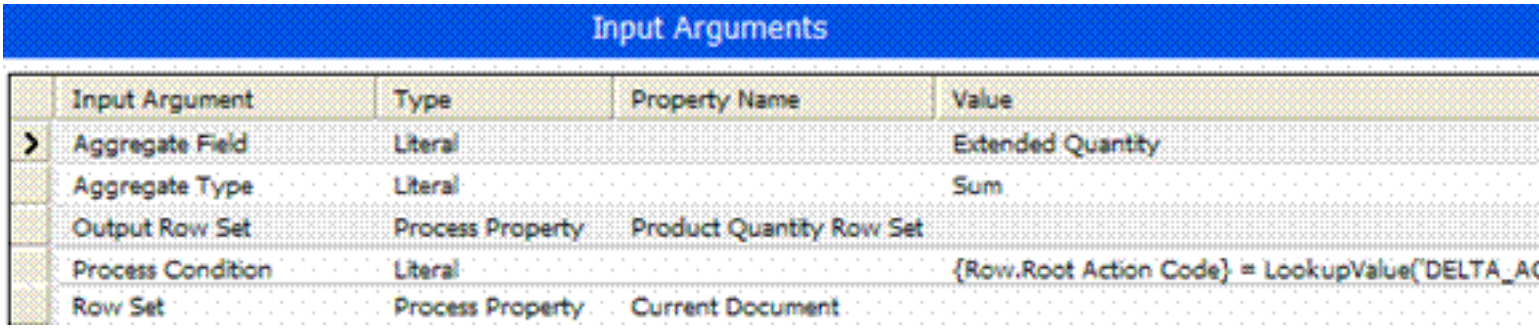
The Aggregate method calculates the sum, average, minimum, maximum, or count of subgroups of the input row set.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Output Row Set is always created and contains the aggregation results. Each row contains the Group By fields plus the Aggregate Fields (example: Price List ID, Product ID, Qty). For the Count aggregate type, the output row contains the Group By fields plus a field called Count.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Aggregate Type	Sum, Avg, Min, Max, or Count.
Aggregate Field	The field to aggregate (example: Qty). Required for all types except Count. Multiple fields can be aggregated by specifying a comma-separated list of field names.
Group By	Optional. A comma-separated list of row field names. Defines the groups of aggregates. (Example: Price List ID, Product ID).

Example

The following figure shows an example of arguments for a PSP procedure step named Sum Product Quantity, which calls the Aggregate method.



Input Arguments

Input Argument	Type	Property Name	Value
Aggregate Field	Literal		Extended Quantity
Aggregate Type	Literal		Sum
Output Row Set	Process Property	Product Quantity Row Set	
Process Condition	Literal		{Row.Root Action Code} = LookupValue(`DELTA_AC
Row Set	Process Property	Current Document	

Conditional Action Method

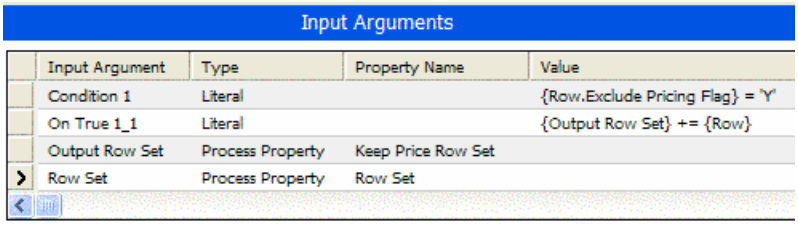
The Conditional Action method evaluates one or more Boolean expression for each row in the row set and performs actions on the row based on whether the conditions are true.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Condition [1..10]	Optional. Boolean expressions that must be satisfied to initiate corresponding actions. Executed like an If…Else If…Else If…Else statement.
On True [1..10]_[1..10]	Actions to perform if the corresponding condition is true.
On Default [1..10]	Actions to perform if none of the conditions is true.

Example

The following figure shows an example of arguments for a PSP procedure step named Keep Price, which calls the Conditional Action method.


Input Arguments

Input Argument	Type	Property Name	Value
Condition 1	Literal		{Row.Exclude Pricing Flag} = 'Y'
On True 1_1	Literal		{Output Row Set} += {Row}
Output Row Set	Process Property	Keep Price Row Set	
Row Set	Process Property	Row Set	

Dynamic Look-Up Method

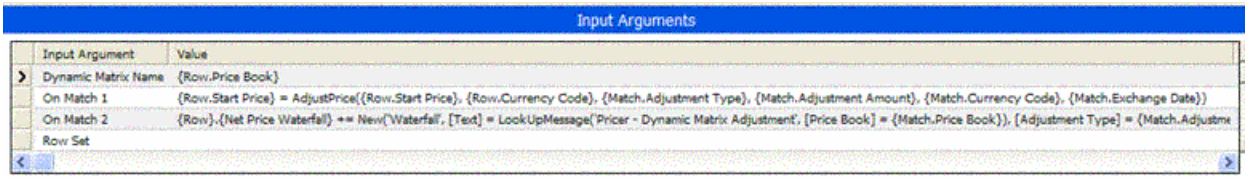
The Dynamic Look-Up method looks up each input row in a dynamic matrix using a dynamic search expression. For example, it could be used for attribute adjustments.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Dynamic Matrix Name	The name of the dynamic matrix to query.
Cache Enabled	Optional. Whether to cache query results. Y or N. The default is N.
On First Match [1..20]	Actions to perform on the first query result for each input row.
On Match [1..20]	Actions to perform on every query result for each input row.
On Last Match [1..20]	Actions to perform on the last query result for each input row.
On No Match [1..20]	Actions to perform if there are no query results for an input row.

Example

The following figure shows an example of arguments for a PSP procedure step named Matrix Adjustment, which uses the Dynamic Look-Up method.



Dynamic Subprocedure Method

The Dynamic Subprocedure method sends each input row to the specified subprocedure for individual processing. Each row can be associated with a different subprocedure.

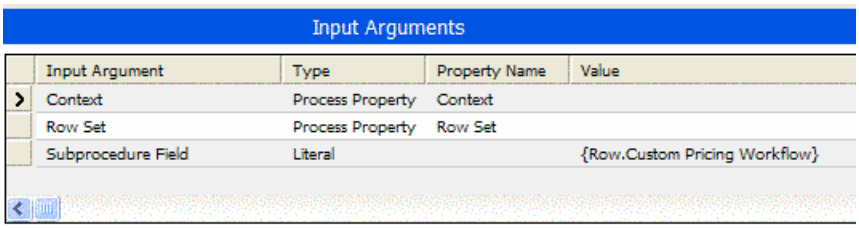
Note: Rows are grouped together into an input row set and passed to each subprocedure in a single invocation.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Subprocedure Expression	Defines the procedure to execute for the row.
[Input Arguments]	Any other input arguments to be passed to the subprocedures.

Example

The following figure shows an example of arguments for a PSP procedure step named Execute Line Specific Pricing.



Hierarchical Look-Up Method

The Hierarchical Look-Up method looks up the closest, best, or accumulated value in an adjustment table for each row by considering each parent in a hierarchy. For example, it could be used to give discounts to companies based on their parent-company discounts.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Hierarchy Business Object	The business object to query to retrieve the hierarchy (example: Account).
Hierarchy Business Component	The business component to query to retrieve the hierarchy (example: Account).
Row ID Field	The hierarchical ID field in the input row (example: “Account Id”.
HBC ID Field	The ID field of the hierarchy business component (example: “Id”).
HBC Parent Field	The parent ID field in the hierarchy business component (example: “Parent Account Id”).
HBC Visibility Mode	Optional. The visibility mode of the hierarchy business component query.
HBC Search Specification	Optional. An additional search specification that is applied to the hierarchy query.
Business Object	The business object to query for matching records (example: Agreement).
Business Component	The business component to query for matching records (example: Agreement Item).
Search Specification	A search expression comprised of business component fields, literals, and variable values from {Context} and {Row}. For example: `[Product Id] = {Row.Product Id} AND [Effective From] <= Today() AND ([Effective To] IS NULL OR [Effective To] >= Today())`
In Memory Search Specification	Optional. Additional terms that are ANDed with the Search Specification. The In Memory Search Specification is executed in memory. This can be used only if Cache Enabled is Y.
Cache Search Specification	Optional. Additional terms that define the key values for the Level 1 cache; for example: `[Price List Id] = {Row.Price List Id})` For more information, see *About PSP Cache Performance Statistics*.
Sort Specification	Optional. A comma-separated list of business component fields used to sort the query result.

Input Argument	Description
Cache Enabled	Optional. Specifies whether to cache query results. Y or N. The default is N.
BC ID Field	The hierarchy object ID field on the query business component (example: Account Id).
On First Match [1..20]	Actions to perform on the first query result for each input row.
On Match [1..20]	Actions to perform on every query result for each input row.
On Last Match [1..20]	Actions to perform on the last query result for each input row.
On No Match [1..20]	Actions to perform if there are no query results for an input row.

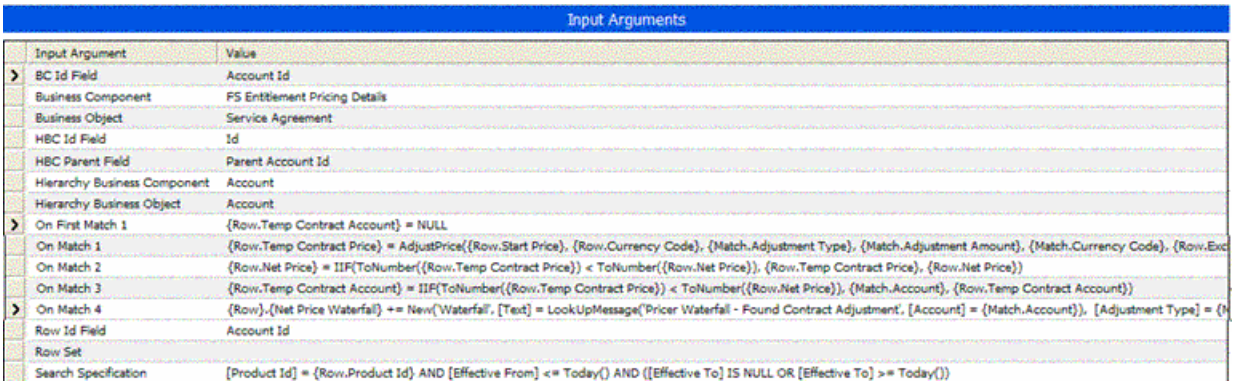
Implementing Aggregate Functions

The arguments for the Hierarchical Look-Up method are used with the aggregate functions shown in the following table. See the following table for further description.

Aggregate Function	Approach
Closest	On First Match condition, set the output row value to a match record value.
Minimum	On Match condition, set the output row value to the value of an expression: `{Row.Value} = IIF({Match.Value} < {Row.Value}, {Match.Value}, {Row.Value})`
Maximum	On Match condition, set the output row value to the value of an expression: `{Row.Value} = IIF({Match.Value} > {Row.Value}, {Match.Value}, {Row.Value})`
Accumulated	On Match condition, set the output row value to the value of an expression: `{Row.Value} = {Row.Value} + {Match.Value}` **Note:** This can be adjusted to support compounding adjustments.

Example

The following figure shows an example of arguments for a PSP procedure step named Hierarchical Contract Adjustment, which uses the Hierarchical Look-Up method.



Input Arguments

Input Argument	Value
BC Id Field	Account Id
Business Component	FS Entitlement Pricing Details
Business Object	Service Agreement
HBC Id Field	Id
HBC Parent Field	Parent Account Id
Hierarchy Business Component	Account
Hierarchy Business Object	Account
On First Match 1	{Row.Temp Contract Account} = NULL
On Match 1	{Row.Temp Contract Price} = AdjustPrice({Row.Start Price}, {Row.Currency Code}, {Match.Adjustment Type}, {Match.Adjustment Amount}, {Match.Currency Code}, {Row.Exc
On Match 2	{Row.Net Price} = IIF(ToNumber({Row.Temp Contract Price}) < ToNumber({Row.Net Price}), {Row.Temp Contract Price}, {Row.Net Price})
On Match 3	{Row.Temp Contract Account} = IIF(ToNumber({Row.Temp Contract Price}) < ToNumber({Row.Net Price}), {Match.Account}, {Row.Temp Contract Account})
On Match 4	{Row}.{Net Price Waterfall} += New('Waterfall', [Text] = LookUpMessage('Pricer Waterfall - Found Contract Adjustment', [Account] = {Match.Account}), [Adjustment Type] = {M
Row Id Field	Account Id
Row Set	
Search Specification	[Product Id] = {Row.Product Id} AND [Effective From] <= Today() AND ([Effective To] IS NULL OR [Effective To] >= Today())

Hierarchical Method

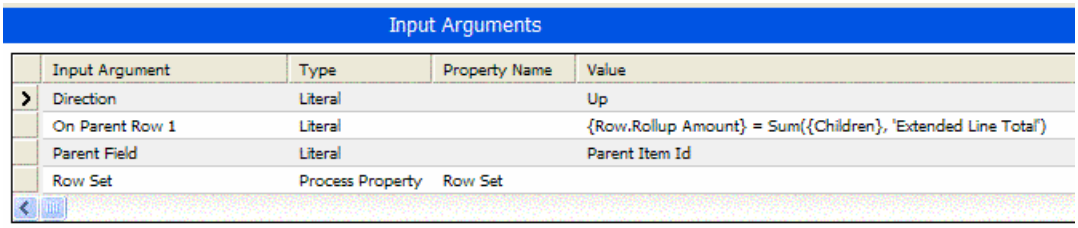
The Hierarchical method processes a hierarchy of input rows from start to end or end to start. For example, it could be used for a customizable product price roll-up.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Row Id Field	The ID field in {Row} (example: “Id”).
Parent Field	The name of the parent ID field that defines the hierarchy in the input row set.
Direction	Up or Down. Indicates the direction of traversal of the tree.
On Top Row [1..20]	Actions to perform on the first row in each tree.
On Row [1..20]	Actions to perform on every row.
On Parent Row [1..20]	Actions to perform on every row that has children beneath it in a tree.
On Leaf Row [1..20]	Actions to perform on rows that have no children.

Example

The following figure shows an example of arguments for a PSP procedure step named Customizable Product Roll-Up, which uses the Hierarchical method.



Merge Method

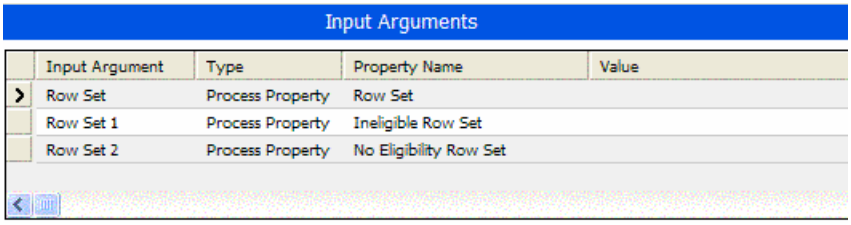
The Merge method combines two or more row sets into a single row set.

Arguments

Input Argument	Description
Row Set	The target row set into which all other row sets will be merged.
Row Set [1..20]	The row sets to merge.

Example

The following figure shows an example of arguments for a PSP procedure step named Merge Ineligible, which uses the Merge method.



Query Method

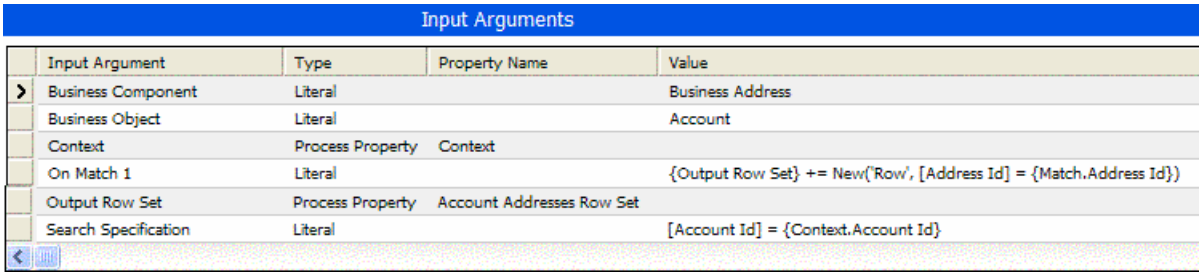
The Query method queries a business component and generates a row set.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Output Row Set	The property set used for output.
Business Object	The business object to query.
Business Component	The business component to query.
Search Specification	A search expression comprised of business component fields, literals, and variable values from {Context}. For example: `[Account Id] = {Context.Account Id}`
In Memory Search Specification	Optional. Additional terms that are ANDed with the Search Specification. The In Memory Search Specification is executed in memory. This can be used only if Cache Enabled is Y.
Cache Search Specification	Optional. Additional terms that define the key values for the Level 1 cache; for example: `[Price List Id] = {Row.Price List Id})` For more information, see *Logging of Performance*.
Process Condition	A Boolean condition that is evaluated to determine whether to process the query.
Sort Specification	Optional. A comma-separated list of business component fields used to sort the query result.
Cache Enabled	Optional. Specifies whether to cache query results. Y or N. the default is N.
On First Match [1..20]	Actions to perform on the first query result.
On Match [1..20]	Actions to perform on every query result.
On Last Match [1..20]	Actions to perform on the last query result.
On No Match [1..20]	Actions to perform if there are no query results.

Example

The following figure shows an example of arguments for a PSP procedure step named Get Account Address, which uses the Query method.


Input Arguments

Input Argument	Type	Property Name	Value
Business Component	Literal		Business Address
Business Object	Literal		Account
Context	Process Property	Context	
On Match 1	Literal		{Output Row Set} += New('Row', [Address Id] = {Match.Address Id})
Output Row Set	Process Property	Account Addresses Row Set	
Search Specification	Literal		[Account Id] = {Context.Account Id}

Row Set Look-Up Method

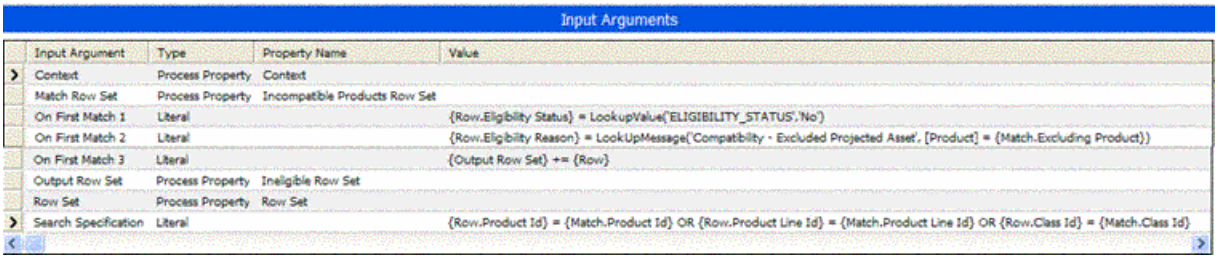
The Row Set Look-Up method looks up each input row in another row set using a specified search expression. For example, it could be used to check for compatibility.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Match Row Set	The row set that will be matched with the input row set.
Search Specification	A search expression comprised of literals and variable values from {Context}, {Row}, and {Match}. For example: `{Row.Product Id} = {Match.Product Id}`
On First Match [1..20]	Actions to perform on the first match for each input row.
On Match [1..20]	Actions to perform on every match for each input row.
On Last Match [1..20]	Actions to perform on the last match for each input row.
On No Match [1..20]	Actions to perform if there are no matches for an input row.

Example

The following figure shows an example of arguments for a PSP procedure step named Flag Incompatible Rows, which uses the Row Set Look-Up method.



Input Arguments

Input Argument	Type	Property Name	Value
Context	Process Property	Context	
Match Row Set	Process Property	Incompatible Products Row Set	
On First Match 1	Literal		{Row.Eligibility Status} = LookupValue('ELIGIBILITY_STATUS','No')
On First Match 2	Literal		{Row.Eligibility Reason} = LookUpMessage('Compatibility - Excluded Projected Asset', [Product] = {Match.Excluding Product})
On First Match 3	Literal		{Output Row Set} += {Row}
Output Row Set	Process Property	Ineligible Row Set	
Row Set	Process Property	Row Set	
Search Specification	Literal		{Row.Product Id} = {Match.Product Id} OR {Row.Product Line Id} = {Match.Product Line Id} OR {Row.Class Id} = {Match.Class Id}

Rule Set Look-Up Method

The Rule Set Look-Up method looks up the rules for each input rule set and tests the rules against the row set. Then it performs actions if the rule set passes or fails. For example, it could be used for identifying applicable bundles or promotions.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Business Object	The business object to query to retrieve the rules.
Business Component	The business component to query to retrieve the rules.
Search Specification	A search expression comprised of business component fields, literals, and variable values from {Context} and {Row}. For example: `[Required Flag] = 'Y'`
In Memory Search Specification	Optional. Additional terms that are ANDed with the Search Specification. The In Memory Search Specification is executed in memory. This can be used only if Cache Enabled is Y.
Cache Search Specification	Optional. Additional terms that define the key values for the Level 1 cache; for example: `[Price List Id] = {Row.Price List Id})` For more information, see *Logging of Performance*.
Sort Specification	Optional. A comma-separated list of business component fields used to sort the query result.

Input Argument	Description
Cache Enabled	Optional. Specifies whether to cache query results. Y or N. The default is N.
Rule Set Field	The field in the business component that groups the rules into a rule set (example: Promotion Id, Bundle Id).
Row Join Field	The field in the {Row} that is joined to a matching rule (example: Product Id). This transform tests all rows that join to see if the rule expression is satisfied.
Rule Join Field	The field in the rule business component that is joined to a matching {Row} (example: Product Id).

Rule Expression	Optional. A Boolean expression that defines whether a rule is satisfied by a row; for example:
	`"{Match.Min Qty} = 0)`
	OR
	`{Match.Min Qty} <= {Row.Qty}")`

Pass Only If All Rules Match	Y or N. This indicates whether to execute the On Pass ... conditions only if all rules in a rule set pass, or whether to execute those conditions for any rule that is satisfied.
Row Set Sort Specification	Optional. A comma-separated list of {Row} field names that determine the sequence in which rows in {Row Set} are processed as they are compared with the rules. This transform is used to make sure that the highest value item is given away in a buy-one-get-one-free scenario. This transform is required for backward compatibility.
Rule Sets	A property set containing a list of rule sets to test. Each child property set can contain the following fields: • [Rule Set Sequence Id] (optional) • [Sequence] (optional) • A value for the "Rule Set Field" (example: [Bundle Id]) • [Next Sequence on Fail] (optional) For example, the list of promotions that requires integrity checking or the list of bundles associated with the current price list. If the [Next Sequence on Pass] field is populated, then the Rule Sets transform skips to that value of [Sequence] if the current rule set passes. If the [Next Sequence on Fail] field is populated, then the transform skips to that value of [Sequence] if the current rule set fails. **Note:** There may be multiple sequences of rule sets in the Rule Sets property set. Individual rule-set sequences are identified by the optional [Rule Set Sequence Id] field. Attributes of the rule set being evaluated are exposed to action syntax as {Rule Set.Value}.
On Pass First Match	Occurs for the first row in the row set that matches a rule in a rule set for which the evaluation criteria are satisfied.

Input Argument	Description
	For 'On Pass ... [Rule]' conditions, {Row} and {Match} variables are available to actions.
On Pass Match	Occurs for every row in the row set that matches a rule in a rule set for which the evaluation criteria are satisfied. For 'On Pass ... [Rule]' conditions, {Row} and {Match} variables are available to actions.
On Pass Last Match	Occurs for the last row in the row set that matches a rule in a rule set for which the evaluation criteria are satisfied. For 'On Pass ... [Rule]' conditions, {Row} and {Match} variables are available to actions.
On Fail First Rule	Occurs for the first rule not satisfied in a failed rule set. **Note:** For On Fail ... [Rule] conditions, only the {Match} variable is available to actions.
On Fail Rule	Occurs for each rule that was not satisfied in a failed rule set.
On Fail Last Rule	Occurs for the last rule that was not satisfied in a failed rule set.

Example

The following figure shows an example of arguments for a PSP procedure step named Identify Applicable Bundles, which uses the Rule Set Look-Up method.

Input Arguments

Input Argument	Type	Property Name	Value
Business Component	Literal		Bundle Discount Component
Business Object	Literal		Admin Price List
On Pass First Match 1	Literal		{Output Row Set} += New('Bundle Discount', [Rule Set Sequence Id] = {
Output Row Set	Process Property	Applicable Bundles Row Set	
Pass Only If All Rules Match	Literal		Y
Row Join Field	Literal		Product Id
Row Set	Process Property	Product Quantity Row Set	
Rule Expression	Literal		{Match.Quantity} <= {Row.Quantity}
Rule Join Field	Literal		Product Id
Rule Set Field	Literal		Bundle Discount Id
Rule Sets	Process Property	Candidate Bundles Row Set	
Search Specification	Literal		[Required Flag] = 'Y'

Simple Look-Up Method

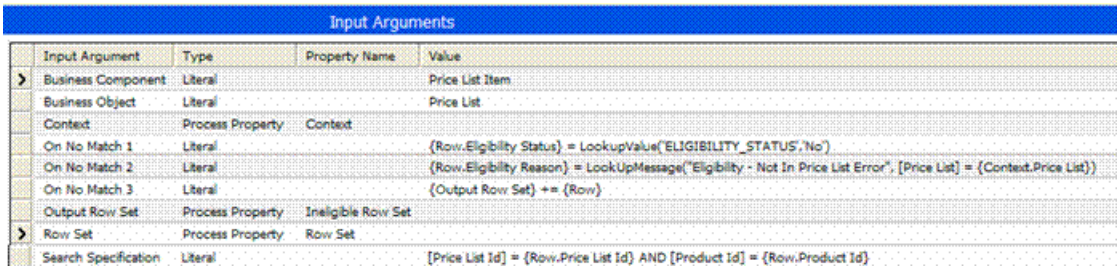
The Simple Look-Up method looks up each input row in a business component using a simple search expression. For example, it could be used to look up list price or exclusive eligibility.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Output Row Set	Optional. The property set used for output.
Process Condition	A Boolean condition that is evaluated to determine whether to process each row.
Business Object	The business object to query.
Business Component	The business component to query.
Search Specification	A search expression comprised of business component fields, literals, and variable values from {Context} and {Row}. For example: `[Price List Id] = {Row.Price List Id} AND [Product Id] = {Row.Product Id}`
In Memory Search Specification	Optional. Additional terms that are ANDed with the Search Specification. The In Memory Search Specification is executed in memory. This can be used only if Cache Enabled is Y.
Cache Search Specification	Optional. Additional terms that define the key values for the Level 1 cache; for example: `[Price List Id] = {Row.Price List Id})` For more information, see *About PSP Cache Performance Statistics*.
Sort Specification	Optional. A comma-separated list of business component fields used to sort the query result.
Cache Enabled	Optional. Specifies whether to cache query results. Y or N. The default is N.
On First Match [1..20]	Actions to perform on the first query result for each input row.
On Match [1..20]	Actions to perform on every query result for each input row.
On Last Match [1..20]	Actions to perform on the last query result for each input row.
On No Match [1..20]	Actions to perform if there are no query results for an input row.

Example

The following figure shows an example of arguments for a PSP procedure step named In Price List, which uses the Simple Look-Up method.



Split Method

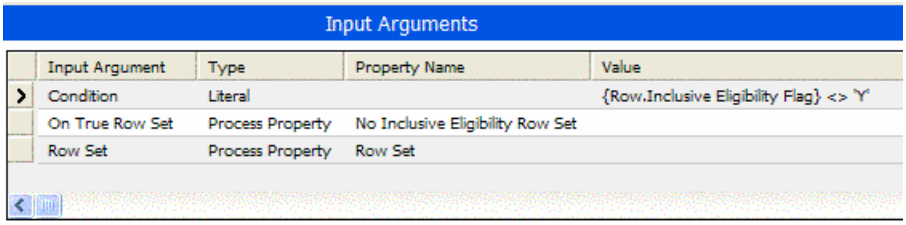
The Split method splits an input row set into two output row sets by evaluating a condition for each row.

Arguments

Input Argument	Description
Context	Optional. Property set of variables shared across all rows.
Row Set	The set of rows to process.
Condition	A Boolean expression that references {Context} and {Row} field values.
On True Row Set	The row set to which rows are moved if Condition evaluates to True.

Example

The following figure shows an example of arguments for a PSP procedure step named Inclusive Eligibility, which uses the Split method.



Configuring PSP Procedures

You configure, test, and release a PSP procedure as you would any workflow process. For information about creating workflow processes, see *Siebel Business Process Framework: Workflow Guide* .

> **Note:** Workflow processes which rely on the UI context and on variable maps and signals can only be tested in the workspace but cannot be simulated.

Use the following sequence of steps:

1. Create or edit a PSP procedure in the Siebel Tools development environment.
2. From within Siebel Tools, start the Siebel Client in debug mode.
3. In the Siebel Client, test the behavior of the PSP procedure.
4. Activate the PSP procedure in production.

The following topics contain more information about configuring PSP procedures:

- *Creating PSP Procedures*
- *Best Practices for Configuring PSP Procedures*
- *Configuring Eligibility, Compatibility, and Pricing*

Creating PSP Procedures

PSP procedures are created in the same way that standard workflow processes are created. For information about creating workflow processes, see *Siebel Business Process Framework: Workflow Guide* .

To create a PSP procedure

1. In Siebel Tools, create a workflow process.
2. To one or more of the workflow's steps, add Row Set Transformation Toolkit methods as input arguments.

> **CAUTION:** PSP requires that each input property set for a PSP procedure step must also be defined as an output property set, even though the input property set is technically not output. This is required by a workflow performance optimization that makes sure input and output property sets are not copied when the transform method is invoked.

Best Practices for Configuring PSP Procedures

To reduce the work of maintenance and to tune performance, follow these guidelines for designing PSP procedures:

- **Use the standard PSP procedures as your starting point.** Trim and tune each PSP procedure to match your business requirements. PSP procedure execution is critical to the end-user response time of your Siebel application. Always review the standard, shipped PSP procedures and trim them as necessary, for example:
  - Eliminate steps that are not required in your implementation.
  - Eliminate variables from the default variable maps that are not required to support your business logic.

> **Note:** These tuning guidelines can have a major impact on performance and scalability. For help with tuning, create a service request (SR) on My Oracle Support. Alternatively, you can phone Global Customer Support directly to create an SR or get a status update on your SR. Support phone numbers are listed on My Oracle Support.

- **Add useful subprocedures.** In every case for which the same set of steps is invoked from multiple places, consider creating a subprocedure for those steps. Then call the subprocedure rather than repeating the set of steps.
- **Remove unnecessary subprocedures.** Avoid or remove subprocedures that are not required by your organization's particular needs. This is recommended because subprocedure calls involve copying the row set, which adds to performance overhead.
- **Keep the logic generic.** Do not implement account-specific or product-specific logic in a PSP procedure. Because this logic changes frequently, it is best for it to be maintained by marketing administrators.

Configuring Eligibility, Compatibility, and Pricing

Different information is required to configure eligibility, compatibility, and pricing in the following scenarios:

- In an asset-based ordering (ABO) environment. For more information, see *Eligibility, Compatibility, and Pricing Using the Configurator in an ABO Environment*.
- In a non-ABO environment. For more information, see *Eligibility, Compatibility, and Pricing Using the Configurator in a Non-ABO Environment*.
- When using the line item UI, as opposed to the Configurator UI
    - The line item UI refers to the Line Items applet in Siebel application. Access the Line Items applet by navigating to (for example) the Sales Order screen, List, then the Line Items view.
    - This Line Items applet contains a Customize button, which allows you to start the Configurator if the product in the order line item can be configured. The Configurator creates a new view (the Configurator UI) to display the line item in a different way, and allow you to reconfigure the line item.

      For more information, see *Eligibility, Compatibility, and Pricing Using the Line Item UI*.

This topic describes the configuration that is necessary in Business Components, Integration Objects, and Business Service Properties to set up eligibility, compatibility, and pricing for these different scenarios.

For information about how to configure additional fields for use in eligibility, compatibility, and pricing, see *Configuring an Additional Field For Use in Eligibility, Compatibility, and Pricing*.

Eligibility, Compatibility, and Pricing Using the Configurator in an ABO Environment

This topic describes how to enable ABO for a Siebel Developer Web Client and a Siebel Application Object Manager.

Enabling Asset Based Ordering for a Siebel Developer Web Client

Use the following procedure to enable ABO for a Siebel Developer Web client.

To enable Asset Based Ordering for a Siebel Developer Web Client

- Set the AssetBasedOrderingEnabled parameter in your configuration file (siebel.cfg) to True.

  Then when you click the Customize button in the quote items applet or order line items applet in your Siebel Business Application:

  - No signal is raised.
  - The *SIS OM Edit Delta Quote Line Item* workflow is invoked
  - The following integration objects are used to build the Configurator instance property set:

    SIS OM Quote

    SIS OM Order

    SIS OM Asset

Enabling Asset Based Ordering for a Siebel Application Object Manager

Use the following procedure to enable ABO for a Siebel Application Object Manager.

To enable Asset Based Ordering for a Siebel Application Object Manager

1. Set the AssetBasedOrderingEnabled parameter in your Siebel Business Application by navigating to the Administration - Server Configuration screen, Enterprises, Component Definitions, and then the Parameters view.
2. Query for the object manager component where you want to enable ABO.
3. Query for the parameter Order Management - Enable Asset Based Ordering, and set the Current Value to True.
4. Save this record.

   Then when you click the Customize button in the quote items applet or order line items applet in your Siebel Business Application:

   - No signal is raised.
   - The *SIS OM Edit Delta Quote Line Item* workflow is invoked
   - The following integration objects are used to build the Configurator instance property set:

     SIS OM Quote

     SIS OM Order

     SIS OM Asset

Eligibility, Compatibility, and Pricing Using the Configurator in a Non-ABO Environment

This topic describes how to disable ABO for a Siebel Developer Web Client and a Siebel Application Object Manager.

Disabling Asset Based Ordering for a Siebel Developer Web Client

Use the following procedure to disable ABO for a Siebel Developer Web client.

To disable Asset Based Ordering for a Siebel Developer Web Client

- Set the AssetBasedOrderingEnabled parameter in your configuration file (siebel.cfg) to False.

  Then when you click the Customize button in the quote items applet or order line items applet in your Siebel Business Application:

  - The Customize signal is raised.
  - The *Configurator Load* workflow is invoked
  - The following integration objects are used to build the Configurator instance property set:

    7.7 Quote Integration Object

    7.7 Order Entry Integration Object

Disabling Asset Based Ordering for a Siebel Application Object Manager

Use the following procedure to disable ABO for a Siebel Application Object Manager.

To disable Asset Based Ordering for a Siebel Application Object Manager

1. Set the AssetBasedOrderingEnabled parameter in your Siebel Business Application by navigating to the Administration - Server Configuration screen, Enterprises, Component Definitions, and then Parameters view.
2. Query for the object manager component where you want to disable ABO.
3. Query for the parameter Order Management - Enable Asset Based Ordering, and set the Current Value to False.
4. Save this record.

   Then when you click the Customize button in the quote items applet or order line items applet in your Siebel Business Application:

   - The Customize signal is raised.
   - The *Configurator Load* workflow is invoked
   - The following integration objects are used to build the Configurator instance property set:

     7.7 Quote Integration Object

     7.7 Order Entry Integration Object

Eligibility, Compatibility, and Pricing Using the Line Item UI

When the user clicks on the Reprice (Pricing) button or Verify (Eligibility) button in the quote items applet or order line items applet in the Siebel Business Application, the following integration object is used to for Eligibility, Compatibility, and Pricing evaluation:

- ISS Quote

Configuring an Additional Field For Use in Eligibility, Compatibility, and Pricing

When RaiseSignal is called from a business component (for example, Reprice All from quote), the instance is a BusComp pointer and data is pulled from the business component that is currently in memory (that is, the business component that the applet within the view is based on). When the instance is a BusComp pointer, the integration object is determined by the 'Instance Uniform Name EAI Object*' user properties on that business component.

- Example: Reprice All from the "Quote Item List Applet (Pricing)" on the "Quote Item Detail View (Pricing)" view:

Integration Object (ABO Mode) - ISS Quote
Integration Object (Non ABO Mode) - ISS Quote
When RaiseSignal is called from the Configurator, the instance is a CxObj pointer and data is pulled from the Configurator instance property set. When the instance is a CxObj pointer, the integration object is determined by the final property set that is passed into Configurator.

- Example: Eligibility Check within configurator when launched from Quote UI:
  Integration Object (ABO Mode) - SIS OM Asset
  Integration Object (Non-ABO Mode) - 7.7 Quote Integration Object

The following procedure describes the objects that must be modified in order to involve an additional field in the eligibility, compatibility, and pricing processes. Using this procedure ensures consistent behavior between quotes and orders in the line item UI and in the Configurator UI, in both ABO mode and in non-ABO mode.

To include additional fields in eligibility, compatibility and pricing

1. Add the field to the following business components:
   MACD Quote Item
   MACD Order Entry - Line Items
   Product Eligibility BusComp
2. Add the field to the following integration objects:
   - Quote UI:
     ISS Quote
   - Order UI:
     ISS Order
   - Configurator UI (ABO Mode):
     SIS OM Quote
     SIS OM Order
     SIS OM Asset
   - Configurator UI (non-ABO Mode):
     7.7 Quote Integration Object
     7.7 Order Entry Integration Object
3. Copy the value of the additional field used in the pricing process by creating the corresponding business service user properties for the SIS OM PMT Service.
4. Add the field to the following variable maps:
   - Quote UI and Order UI:
     Default Eligibility Variable Map - Context, and Product Eligibility Variable Map - Context
     Default Pricing Variable Map - Context

- Configurator UI (ABO Mode):
  Cfg Eligibility Variable Map - Context
  Default Pricing Variable Map - Context
- Configurator UI (non-ABO Mode):
  Cfg Eligibility Variable Map - Context
  Default Pricing Variable Map - Context

5. Update the following workflow processes as necessary:
   .Product Eligibility & Compatibility - Default
   .Pricing Procedure - Default

Creating a Custom PSP Application

PSP is a general-purpose mechanism that can be used anywhere in the Siebel application. For example, you can create a custom PSP procedure to determine the allowed shipping methods for a line item on an order, or you can create a custom PSP procedure to determine the disclosures that must be read to a customer before the purchase of a product.

Your custom PSP application must include the following:

- Matrix tables, business components, and the administrative UI to capture the rules.
- Signals to invoke the controller workflow. See *Signals*.
- A controller workflow that establishes the input context and row set and processes the PSP output.
- A PSP procedure that transforms the input row set.
- If necessary, scripted business service methods that extend the set of Row Set Transformation Toolkit business service methods. See *Creating a Custom Transform*.

Creating a Custom Transform

PSP procedures can invoke any custom business service method that you create using Siebel VB or Siebel eScript. Custom business service methods for PSP follow these guidelines:

- The row set and context are passed as inputs to the custom business service method.
- Wherever possible, parameterize new methods to make them flexible and applicable to multiple situations.
- The method uses standard property set APIs to read and write from the row set.
- The row set must be returned as an explicit output argument of the business service method.

For more information, see *Siebel VB Language Reference* or *Siebel eScript Language Reference* .

Calling a PSP Procedure from an External Application

You can invoke a PSP procedure from an external application by using a Web service ASI or a business service API.

To call a PSP procedure from an external application

1. Create a Web Service ASI or expose a business service API.
2. Use the Web Service ASI or business service API to invoke a controller workflow.
3. Use the controller workflow to do the following:
   a. Convert the XML input document to a row set using the XSLT or the Siebel Data Mapper.
   b. Construct a Context property set (if required) from the XML input document using XSLT or the Siebel Data Mapper.
   c. Invoke the standard PSP procedure.
   d. Construct an XML response from the output row set.

About Logging of PSP

PSP transforms support logging for troubleshooting and performance tuning. Logging is implemented using the standard Siebel logging mechanisms. See *Siebel System Monitoring and Diagnostics Guide* .

With one exception, all PSP logging events have the primary purpose of supporting troubleshooting. The PSP logging event called PSP Cache supports performance tuning.

Troubleshooting PSP

There are several server parameters used for PSP logging. For details on using PSP logging events for troubleshooting, see *About Troubleshooting of PSP*.

Tuning Performance of PSP

For details on using the PSP Cache event and other logging events for performance tuning, see *About Tuning Performance of PSP* and *Logging of Performance*.

About Troubleshooting of PSP

To manage PSP logging-related server parameters, navigate to the Administration - Server Configuration screen, Servers, and then the Events view, and query on “PSP*”.

The server parameters for PSP logging are listed in the following table. PSP Cache Event is used for performance tuning, while all the other server parameters described in this topic are used for troubleshooting.

Event Type	Alias
PSP Cache Event	PSPCache
PSP Data Event	PSPData
PSP Parser Event	PSPParser

Event Type	Alias
PSP Transform Event	PSPTransform
PSP Pricer Service Event **Note:** The PSP Pricer Service Event logging parameter is used only in implementations that include Siebel Pricer.	PSPPricerSvc

Like all other Siebel-standard log events, the default log level is 1, and the log level can be set from 1 to 5. The higher the log level is set, the more messages are logged. Log level settings generate the types of data listed in the following table.

The data logged by the PSP logging-related server parameters is explained in the topics that follow.

Level	Type of Data Logged
1	Error messages
2	Warnings
3	Information
4	Detailed information
5	Debugging information

PSP Cache Event

Log levels and data logged for PSP Cache Event are listed in the following table. PSP Cache Event is used for logging for PSP Cache, and for tuning performance of PSP transforms.

For more information about this event, see *Logging of Performance*. For information about other events used to log performance of PSP, see *Logging of Performance*.

Level	Data Logged
3	PSP Cache miss or hit
	Keys: first-level key, second-level key

PSP Data Event

Log levels and data logged for PSP Data Event are listed in the following table. PSP Data Event is used for logging of PSP transform input arguments.

Level	Data Logged
4	Input arguments (excluding the hierarchy arguments) of each transform are logged as Name:Value pairs.
5	The whole input property set of each transform is logged as an XML string.

PSP Parser Event

Log levels and data logged for PSP Parser Event are listed in the following table. PSP Parser Event is used for logging of PSP Parser.

Level	Data Logged
5	Debugging information for PSP parser.

PSP Transform Event

Log levels and data logged for PSP Transform Event are listed in the following table. PSP Transform Event is used for logging of PSP transforms.

Level	Data Logged
3	Business service name
	Number of rows processed.
	Number of rows deleted.
4	Transform progress information.
5	Debugging information for PSP transforms.

PSP Pricer Service Event

Log levels and data logged for PSP Pricer Service Event are listed in the following table. PSP Pricer Service Event is used for logging of Pricer service APIs.

Level	Data Logged
5	Debugging information for PSP Pricer business service.

About Tuning Performance of PSP

Consider the following when tuning the performance of your preconfigured and custom PSP procedures:

Preconfigured PSP Procedures

Tune the preconfigured PSP procedures by:

- Removing steps that your implementation does not require
- Eliminating unused values from the variable maps

General Design Guidelines

Follow these general design guidelines to further improve the performance of PSP:

- Optimize eligibility PSP procedures by:
  - Executing low-cost tests first
  - Performing high-cost tests (such as Web Service calls) only in post-pick processing
- Build performance hints into the procedure definition. Use the Process Condition input argument in each step to identify the subset of rows in the row set that require processing (example: "{Row.Promotion Id} IS NOT NULL"). This can eliminate unnecessary SQL and in-memory operations.
- When multiple steps operate on the same subset of rows, split the row set, perform the operations on the subset of rows, and then merge the two split row sets afterwards.
- Avoid unnecessary subprocedures. Subprocedure calls involve copying the row set, which it is best to avoid where possible.
- Optimize external Web Service calls by:
  - Designing the Web Service interface to be set-based
  - Making sure that a single invocation will process all rows in the row set

- Use PSP Cache for caching of database query results. For more information about how to tune PSP performance with caching, see *Logging of Performance*.

SQL Queries

Use the following guidelines to improve the SQL query performance of PSP:

- Minimize the number of SQL queries executed. Consolidate multiple Simple Look-Up steps into one step if the steps use data from the same reference data business component with the same search specification.
- Tune SQL queries by:
  - Querying through thin business components to minimize Siebel Object Manager overhead and reduce query complexity.
  - Making sure that all search specifications have index coverage.

Logging of Performance

PSP provides extensive logging of performance-related data. Analyze the PSP log to determine which steps are consuming the most processing time and where the caching can be further optimized.

PSP logging that takes place in the server environment is more effective than PSP logging that takes place locally, because in the server environment there is only one environment variable controlling the log level for all the events.

You can use the events listed in the following table to log the performance of PSP.

Event Type	Alias	Description
Object Manager Business Service Operation and SetErrorMsg Log	ObjMgrBusServiceLog	Logs the performance of business service methods which include the PSP transform methods.
Workflow Performance	WfPerf	Logs the performance at the workflow level or step level.
PSP Cache Event For more information, see *Logging of Performance*.	PSPCache	Logs the performance of PSP Cache.

When debugging the pricer, the price waterfall output provides valuable clues as to which transforms and actions were executed. For more details on logging, see *About Logging of PSP*.

About PSP Cache

Siebel PSP Cache is a mechanism designed to improve performance of PSP transforms. PSP Look-Up transforms use caching to reduce the number of SQL statements executed by the database. The cache stores the results of PSP Look-Up transform queries. The cache key is the business object, the business component, the search specification, and the sort specification.

The PSP Cache of query results is shared across all user sessions on an Object Manager. A particular query is issued only once for each Object Manager and then shared by all users. This sharing maximizes the probability of a cache hit and improves performance and scalability for all users on the server.

Transforms Involving Database Queries

Of all the PSP methods (transforms) provided by the Row Set Transformation Toolkit business service, those that involve database queries are the following:

- Simple Look-Up
- Query
- Hierarchical Look-Up
- Rule Set Look-Up
- Dynamic Look-Up

There are two ways that PSP Cache is implemented: one is special for the Dynamic Look-Up transform, and the other is for the rest of the transforms:

- **PSP Dynamic Look-Up Transform Cache.** This cache is used when the Dynamic Look-Up transform performs a query. For more information, see *About Using the PSP Dynamic Look-Up Transform Cache*.
- **PSP Generic Cache.** This cache is used when all transforms other than the Dynamic Look-Up transform perform a query. For more information, see *Using the PSP Generic Cache*.

The following topics contain more information about PSP Cache:

- *Enabling PSP Cache*
- *Setting Cache Size*
- *Using the PSP Generic Cache*
- *Optimizing PSP Cache*
- *Defining a Cache Refresh Key*
- *Configuring a Clear Cache Button*
- *About Using the PSP Dynamic Look-Up Transform Cache*
- *About PSP Cache Performance Statistics*

Enabling PSP Cache

You turn the PSP Cache on or off using an input argument, at the PSP procedure step level (that is, one input argument for each step that involves caching). By default, caching is disabled.

To enable caching, add an input argument to the step involving a Look-Up transform or Query transform, as follows:

```
Cache Enabled = Y
```

Note: If Cache Enabled is not defined for a PSP procedure step, the default value is N and caching is not enabled.

Setting Cache Size

To control cache size, use the following server parameters:

- **PSP Level 1 Cache Max Item Count.** The server parameter with this display name is PSPCacheMaxItemCntLevel1. This is the maximum number of business component or cache refresh key combinations; for example:

  ```
  Price List Item/Price List Id = '12-12345')
  ```

  The default value is 10000.
- **PSP Level 2 Cache Max Item Count.** The server parameter with this display name is `PSPCacheMaxItemCntLevel2`. This is the maximum number of distinct queries cached for each PSP Level 1 Item. The default value is 10000.
- **PSP Expr Cache Max Item Count.** The server parameter with this display name is `PSPExprCacheMaxItemCnt`. This is the maximum number of PSP conditional expression that can be cached. The default value is 10000.

When either Level 1 or Level 2 cache reaches capacity, the least recently used query results are dropped to make space for new cache entries.

It is not possible to directly control the amount of memory consumed by the PSP cache by setting a total size for PSP cache, as the architecture does not count the memory of each cache item.

Note: You must restart the Siebel server for any parameter changes to take effect.

For information about setting server parameters, see *Siebel System Administration Guide* .

Using the PSP Generic Cache

The PSP Generic Cache is the cache used for all transforms that involve database queries except the Dynamic Look-Up transform. The Simple Look-Up, Query, Hierarchical Look-Up, and Rule Set Look-Up transforms use PSP Generic Cache.

Topics that relate only to PSP Generic Cache and not to PSP Dynamic Look-Up Transform Cache are the following:

- *Optimizing PSP Cache*
- *Defining a Cache Refresh Key*
- *Configuring a Clear Cache Button*

Optimizing PSP Cache

To maximize the cache hit rate (and hence, performance and scalability), partition the transform search specification into a high selectivity clause that is executed by the database and used as part of the PSP cache key (the Search Specification input argument) and a low selectivity clause that is executed by the PSP transform itself to further filter the query results (the In Memory Search Specification input argument). When you use the In Memory Search Specification input argument in combination with a Search Specification input argument, your search specification is, effectively,

"[Search Specification] AND [In Memory Search Specification]". The two search specification input arguments are divided by purpose as follows:

- **Search Specification.** Use this input argument to define highly selective search criteria executed by the database.
- **In Memory Search Specification.** Use this input argument to define low selectivity search criteria executed by the Siebel Server.

> **Note:** The Dynamic Look-Up transform does not support the In Memory Search Specification input argument. This transform dynamically generates its own search specification.

The order of search implementation is as follows: first the Search Specification input argument is applied to the database query. Next, the returned result set is further filtered in memory by applying the In Memory Search Specification input argument.

Example values for Search Specification and In Memory Search Specification are shown in the following table for the Pricer Simple Volume Discount step.

Input Argument	Value
Search Specification	`[Volume Discount Id] = {Row.Volume Discount Id} AND [Volume Discount Method] = LookupValue('VOL_DISCNT_METHOD', 'SIMPLE')`
In Memory Search Specification	`[Minimum Quantity] <= {Row.Extended Quantity Requested} AND ([Maximum Quantity] >= {Row.Extended Quantity Requested} OR [Maximum Quantity] IS NULL) AND [Volume Discount Start Date] <= Timestamp() AND ([Volume Discount End Date] >= Timestamp() OR [Volume Discount End Date] IS NULL)`

The example shown in the previous table results in one query for each volume discount that retrieves all result rows. All subsequent queries against that volume discount are served from the cache regardless of the values for [Extend Quantity Requested] or Timestamp().

> **Note:** In Memory Search Specification execution does not use sophisticated database features such as indexes. Make sure the result set searched in memory is not too large. For example, loading an entire price list in one query is not likely to improve performance; search a subset of the price list.

For information about PSP Cache performance, see *About PSP Cache Performance Statistics*.

Defining a Cache Refresh Key

One complication caused by the PSP engine's extensive use of caching is that changes to reference data (such as price list line items) that are currently in cache are not reflected immediately after an updated version of the reference data is released. The PSP Cache persists until a Siebel Server is restarted. To enable administrative updates against a running system, every administration view that maintains data cached by PSP has a Clear Cache button that causes all PSP caches on all object managers in the Siebel Enterprise to purge a subset of the cached data for that particular business component. That subset of data is defined by the cache refresh key for the business component.

When implementing PSP Cache through the PSP Generic Cache, you can control the granularity of the cache refresh by defining a cache refresh key for a business component. If no cache refresh key is defined, the business component is refreshed as a whole. To improve performance, you can use a cache refresh key to clear only a selected part of the cache.

A business component can have only one cache refresh key. Some business components have a cache refresh key that comes preconfigured. For example, the Price List Item business component has a cache refresh key of Price List Id. This means that when the user clicks the Clear Cache button in the Price List list applet, only the selected price lists are cleared from the cache.

You define a cache refresh key by adding user properties to the Row Set Transformation Toolkit business service. For each cache refresh key, one pair of user properties is required, as shown in the following table.

Name	Value
Cache Key: Price List Item	Price List ID
Cache Key: [BC Name]	[Key Field 1], [Key Field 2]

Note: You must update a workspace with the changes and then deliver it for the changes to take effect.

Search Specification and Cache Refresh Key

If a cache refresh key is defined for a business component, then every query against that business component must provide the cache key field values in its Search Specification or in the Cache Search Specification input argument.

Note: Failure to specify the cache refresh key values will result in an error.

Use the Cache Search Specification input argument to avoid unnecessary clauses in the Search Specification executed by the database. The transform will look first at the Cache Search Specification, and then at the Search Specification to identify cache key values.

The syntax and structure of the Cache Search Specification input argument are shown in the following example:

```
[Price List Id] = {Context.Price List Id} AND [Price List Item Id] = {Row.Price List
Item Id}
```

Row Set Transformation Toolkit Methods for PSP Cache Refresh

The Row Set Transformation Toolkit methods listed in the following table, with their corresponding input arguments and output arguments support refreshing of the PSP cache:

Method	Description
Get Cache Key	Checks the cache key definitions for the specified business component and constructs a property set containing the cache key values for each selected row in the active business component. This enables multiselect when clearing cache entries (such as price lists).
In: Business Component Name	
Out: Row Set	

Method	Description
Refresh Cache	Clears the entire PSP cache.
Refresh BC	Clears all PSP cache entries for the specified business component. **Note:** Refresh BC does not clear the cache if the business component has a cache key defined.
In: Business Component Name	
Refresh BC On Cache Key	Clears all PSP cache entries for the specified business component and cache key values. The input Row Set is typically the output of the Get Cache Key method.
In: Business Component Name	
In: Row Set	

When the user clicks the Clear Cache button for business components without a cache key, the Refresh BC method is called directly by a run-time event. For business components with a cache key (for example, Price List Item has one preconfigured), the methods Get Cache Key and Refresh BC On Cache Key are invoked by a workflow that is triggered by a run-time event.

Configuring a Clear Cache Button

The PSP cache persists until the Siebel Server is restarted, but you can force a refresh of cached data across all servers in the enterprise. In the Administration - Pricing views, Clear Cache buttons exist for this purpose, to allow administrative updates against a running system. All objects cached by PSP must support a Clear Cache button that forces this refresh of cached data.

Requirements for configuring Clear Cache buttons vary as follows:

- If the business component does not have a cache refresh key defined, a run-time event must be defined to refresh the cache. See *Clear Cache Button for BusComps without a Cache Refresh Key*.
- If the business component does have a cache refresh key defined, a workflow and a run-time event must be defined to refresh the cache. See *Clear Cache Button for BusComps with a Cache Refresh Key*.

Note: Although there is a Clear Cache button in the Administration - Pricing screen, then the Attribute Adjustments view, the run-time event for this button is fixed. This is the Clear Cache button associated with the PSP Dynamic Look-Up Transform Cache. Do not try to configure this Clear Cache button as you would the Clear Cache button used by the PSP Generic Cache.

Clear Cache Button for BusComps without a Cache Refresh Key

In this case, you must define a run-time event to refresh the cache.

To configure a new Clear Cache button for a business component without a cache refresh key defined

1. Create a control in the desired applet where the method invoked is EventMethodCacheRefresh.
2. Navigate to the Administration - Runtime Events screen, then the Action Sets view, and create a new record in the Action Sets list applet.
3. Give the record a name, such as “Cache Refresh BC - Applet Name”.
4. Create a new record in the middle list applet, with the following values:
   - Name is PSP Refresh
   - Action Type is BusService
5. In the More Info form applet (at the end), enter the following values:
   - Business Service Name is Row Set Transformation Toolkit
   - Business Service Method is Refresh BC
   - Business Service Context is a *list of business component names with commas as the separator, and with each business component name included in quotation marks*

     Example:

     ```
     Business Service Context ="ProcessName", "PSP Refresh Cache On Cache Key -
     Price List"
     ```
6. Navigate to the Administration - Runtime Events screen, then the Events view, and create a new record with the following values:
   - Object Type is Applet
   - Object Name is the name of the applet referred to in Step 1
   - Event is InvokeMethod
   - Subevent is EventMethodCacheRefresh
   - Action Set Name is the name of the action set created in Step 2

Clear Cache Button for BusComps with a Cache Refresh Key

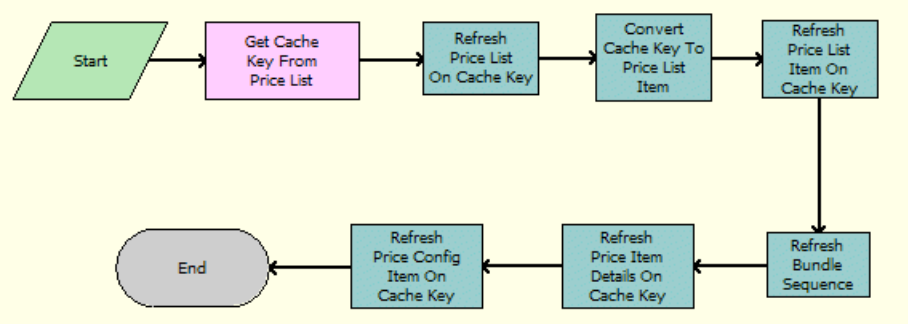
In this case, you must define a workflow and a run-time event to refresh the cache.

To configure a new Clear Cache button for a business component with a cache refresh key defined

1. Create a control in the desired applet where the Method Invoked is EventMethodCacheRefresh, add the control to the desired applet web template.
2. Define the cache keys by adding user properties to the Row Set Transformation Toolkit business service, according to the format shown in the following table, for example:

Property	Value
Name	Cache Key: Price List Item
Value	Price List Id

3. Create a workflow that does the following:
   a. Calls the Get Cache Key method to determine the selected cache key values.
   b. Invokes the Refresh BC On Cache Key method for each business component that shares the same cache key. The following figure shows an example of a Clear Cache workflow.



4. Navigate to the Administration - Runtime Events screen, then the Action Sets view, and create a new record in the Action Sets list applet with a name such as “Cache Refresh BC - Applet Name”.
5. Create a new record in the middle list applet, with the following values:

Property	Value
Name	PSP Refresh
Action Type	BusService

6. In the More Info form applet (at the end), enter the following values:

Property	Value
Business Service Name	Workflow Process Manager
Business Service Method	RunProcess
Business Service Context	Process Name

For example:

```
Business Service Context = PSP Refresh Cache On Cache Key - Price List
```

7. Navigate to the Administration - Runtime Events screen, then the Events view, and create a new record with the following values:

Property	Value
Object Type	Applet
Object Name	The name of the applet referred to in Step 1
Event	InvokeMethod
Subevent	EventMethodCacheRefresh
Action Set Name	The name of the action set created in Step 4

About Using the PSP Dynamic Look-Up Transform Cache

The Dynamic Look-Up transform has its own cache, called the PSP Dynamic Look-Up Transform Cache. You enable this for a particular step by setting Cache Enabled to Y, the same as for the generic PSP cache. No other cache-specific input arguments are supported for the Dynamic Look-Up Transform.

The Dynamic Look-Up transform supports a preconfigured Clear Cache button; do not modify this preconfigured Clear Cache button.

Note: This is the Clear Cache button in the Administration - Pricing screen, then the Attribute Adjustments view. The run-time event for this button is fixed. Do not try to configure this Clear Cache button as you would the Clear Cache button used by the PSP Generic Cache.

About PSP Cache Performance Statistics

To view statistics on PSP Cache performance, navigate to the Administration - Server Management screen, Tasks, and then the Statistics view, and query on “PSP*”. The Siebel application provides the following statistics:

- **PSP Cache Hit Total.** An integer that indicates how many times the cached query results are used.
- **PSP Cache Miss Total.** An integer that indicates how many times a query cannot be found in PSP Cache for which a database query has been conducted.

Note: The higher the value of `PSP Cache Hit Total /(PSP Cache Hit Total + PSP Cache Miss Total)`, the better performance exhibited by PSP Cache.

6 PSP Waterfall

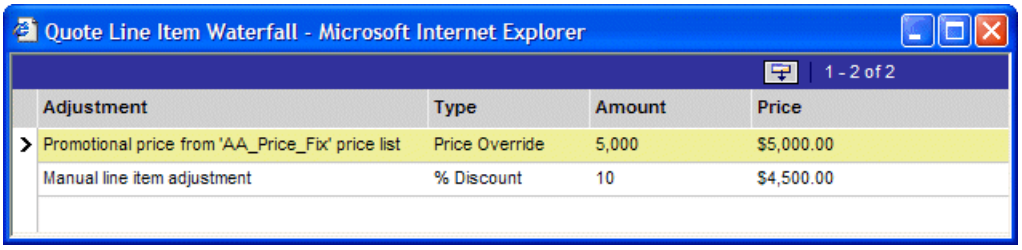
PSP Waterfall

This chapter explains the PSP Waterfall mechanism. A PSP procedure can create waterfall output to explain calculations it has made or actions it has taken. The PSP Waterfall business service displays or saves that output. This chapter includes the following topics:

- *About Waterfalls*
- *About Configuring Waterfall Output*
- *Saving Waterfall Data*

About Waterfalls

A *waterfall* is an applet or a pop-up window that provides line-item explanation about field values— displaying any combination of values such as text, numbers, or dates—such as the figures that were used to arrive at a particular value in a field. As one example of a waterfall, the pricing waterfall shown in the following figure shows the details of the calculation used to arrive at the net price. This example shows a base price used for an item ordered, minus the discount given to arrive at the net price.



As another example, you might implement a waterfall on a product's eligibility status, to show the end user all the reasons a product cannot be purchased (rather than just one reason).

All text displayed to the end user in waterfalls is translatable. the Unified Messaging Service (UMS) business service dynamically translates and substitutes waterfall text . The UMS business service processes all translations through the LookUpMessage API in PSP action script:

```
{Row}.{Net Price Waterfall} += New('Waterfall', [Text] = LookUpMessage({Row.Temp
List Price Message}, [Price List] = {Match.Price List}), [Currency Code] =
{Row.Currency Code}, [Price] = {Row.List Price})
```

For more information about UMS, see *Unified Messaging*.

A PSP Procedure Generates Waterfall Output Each Time It Executes

Waterfall output is generated on demand when the user clicks a waterfall-enabled field, but the PSP procedure generates the waterfall output every time it executes. The waterfall's output may be ignored much of the time, but when

the user drills into a waterfall-enabled field, the procedure reruns to generate and then display the waterfall output for that record and field.

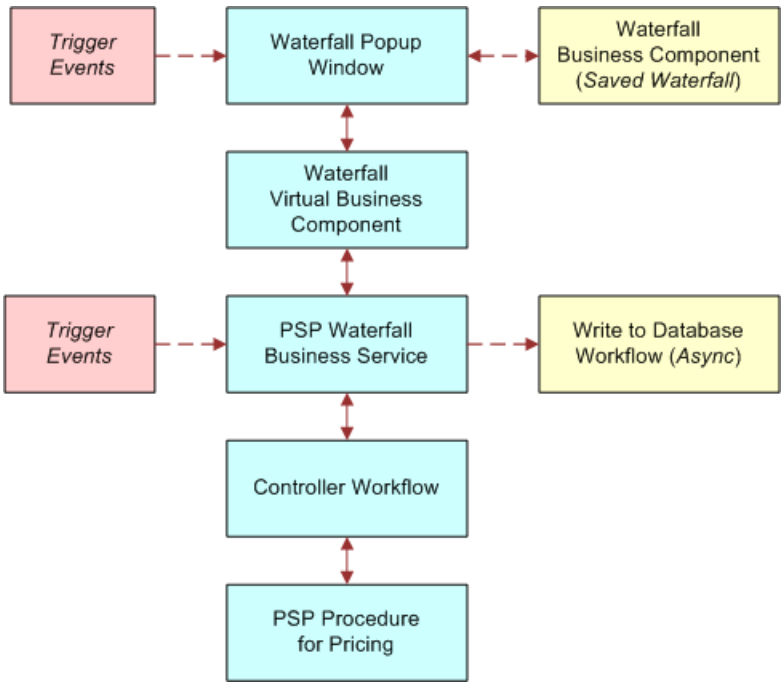
For example, the values for the pricing waterfall in the following figure are generated when the user clicks the Net Price field in a Quote or Order line item. The waterfall pop-up window appears displaying these values. The user clicks OK to hide the pop-up window. But even if the user does not click the Net Price field to view the pricing waterfall, the same waterfall output is generated (but not saved or displayed without configuration) when the PSP procedure executes.

A Controller Workflow Invokes the PSP Waterfall Business Service

The following figure shows how the PSP Waterfall mechanism generates and saves waterfall output.

- For a PSP procedure that generates waterfall output, such as the Pricing PSP procedure shown in the diagram, a controller workflow invokes the PSP waterfall business service.
- The PSP engine supports the creation of a waterfall with an unlimited number of sequenced child rows to explain each name-value pair in the row. It is the child rows' type that defines the waterfall to which they belong.

  For example, "Net Price Waterfall" is the type for each of the child rows that compose the Net Price waterfall. Multiple waterfalls can be created for one row, up to one for each name-value pair.
- Each waterfall has a configurable set of name-value pairs. For example, a pricing waterfall uses different fields than an eligibility waterfall.



Waterfalls and Performance

Waterfall output is generated every time a PSP procedure executes. For example, when the user clicks a Net Price to display its waterfall, that record is repriced and fresh waterfall output is generated for display. This process keeps the waterfall output and net price in sync. This process requires minimal overhead, because it is done in memory.

If a line item is read-only (for example, because it has already been submitted as an order), then it is not possible to reprice the item. In this case, the waterfall virtual business component looks for saved waterfall output for that line item and the waterfall displays that saved waterfall output instead.

> **Note:** The ReadOnlyOrderStatus user property is overwritten by the Business Component read-only field user property (if the value for the business component user property is active). For example, if ReadOnlyOrderStatus for a BC User Property is set to Billed, Submitted, Completed, and read-only status for Submitted is Active, then the record is read-only for the status value Submitted (not for Billed and Completed).

Waterfall output is only saved to the database upon user request. Writing waterfall records to a database can be costly. The waterfall records are written in the background to minimize end-user latency. For more information, see *Saving Waterfall Data*.

Configuration of Waterfalls

When configuring waterfalls, refer to the information provided in the following topics:

- *About Configuring Waterfall Output*
- *Saving Waterfall Data*

About Configuring Waterfall Output

The waterfall output of a PSP procedure is soft-coded in the step actions. Each line item of waterfall output is created by an action that adds a child record to a named child property set of a {Row}. The following is an example PSP action that creates a row of waterfall output:

```
{Row}.{Net Price Waterfall} += New('Waterfall', [Text] = LookUpMessage('Pricer -
Dynamic Matrix Adjustment', [Price Book] = {Match.Price Book}), [Adjustment Type] =
{Match.Adjustment Type}, [Adjustment Amount] = {Match.Adjustment Amount}, [Currency
Code] = {Row.Currency Code}, [Price] = {Row.Start Price})
```

In this example, a waterfall record is added to the Net Price Waterfall child property set of the current {Row}.

The fields written to the waterfall record are soft-coded in the action expression (in this case, Text, Adjustment Type, Adjustment Amount, Currency Code, Price). One common configuration is to add additional fields to an existing waterfall. *Adding New Fields to an Existing Waterfall* describes this configuration.

Another common configuration is to create a new waterfall output for an additional calculated value. For example, a pricing procedure could calculate the Net Price and the Cost of a line item. A waterfall explanation of the calculation of Cost could be exposed as a drill-down link on the Cost field in the UI. You can create many waterfalls for a particular

{Row} by defining different child property set names (such as, `{Row.Cost Waterfall} += New("Waterfall", …)`). See *Creating a New Waterfall* for detailed instructions.

Further configuration information appears in *Saving Waterfall Data*.

Adding New Fields to an Existing Waterfall

You can add as many new fields as you like to an existing waterfall using the following procedure as an example. This example adds an Accounting Code field to the existing Net Price waterfall.

To add a new field to an existing waterfall

> **Note:** Data Maps can be Workspace enabled in your development environment. If they have been Workspace enabled in your development environment and you are working in that environment, then you can only modify them in an editable Workspace. You do not need an editable Workspace to create and edit Data Maps in your Production environment.

1. In Siebel Tools, revise the PSP procedure (for example, revise Pricing Procedure - Calculate Net Price).
2. Add an extra comma-separated argument to the `+= New()` function in every action that generates waterfall output. For example:

```
{Row}.{Net Price Waterfall} += New('Waterfall', [Text] = LookUpMessage('Pricer -
Dynamic Matrix Adjustment', [Price Book] = {Match.Price Book}), [Adjustment Type]
= {Match.Adjustment Type}, [Adjustment Amount] = {Match.Adjustment Amount},
[Currency Code] = {Row.Currency Code}, [Price] = {Row.Start Price}, [Accounting
Code] = {Match.Accounting Code})
```

   The syntax `[Accounting Code] = {Match.Accounting Code}` adds the new field.
3. Add the new field to the waterfall virtual business component (for example, Net Price Waterfall VBC).
4. Add the new field to the waterfall pop-up applet (for example, Net Price Waterfall Popup List Applet).
5. (Optional) If the new field needs to be written to the database:
   a. Add the new field to the tables used to store this waterfall type (for example, S_QTEIT_WTR_LOG, S_ORDIT_WTR_LOG, S_AGRIT_WTR_LOG).
   b. Add the new field to the business components used to persist this waterfall type (for example,*Quote Line Item Waterfall, Order Entry Line Item Waterfall, Service Agreement Line Item Waterfall*).
   c. Navigate to the Administration Application screen, then the Data Maps view. Add the new field to the waterfall data map objects (for example, *Quote Waterfall Data Map Object, Order Waterfall Data Map Object, Service Agreement Waterfall Data Map Object*).

Creating a New Waterfall

You can add a new waterfall output to any PSP procedure. A PSP procedure can have multiple waterfall outputs.

> **Note:** It is recommended that you copy the configuration of the Net Price Waterfall when creating a new waterfall output.

The following topics contain information about creating new waterfalls:

- *Populating Child Waterfall Property Sets*
- *Exposing the Waterfall Output*
- *Saving Waterfall Data*

Populating Child Waterfall Property Sets

When creating a new waterfall output, you first populate a child waterfall property set for each output row of the PSP procedure.

To populate a child waterfall property set for each output row of a PSP procedure

1. Create new UMS message types to format text for your new waterfall output. See *Creating Message Types*.
2. Define PSP actions to create the waterfall output in your PSP procedure.
   a. Use the `{Row.Waterfall Name} += New()` syntax to create a new waterfall record and append it to the desired waterfall property set. The `New()` function has the following syntax:

      ```
      New('Waterfall', Name 1 = Value 1, Name 2 = Value 2,.)
      ```

      Name n and Value n are the waterfall output field names and values; for example:

      ```
      Currency Code = USD
      ```

   b. Use the LookUpMessage function to format text in the appropriate language with variable values substituted through a call to the UMS business service, with the following syntax:

      ```
      LookUpMessage('Message Name', Name 1 = Value 1, Name 2 = Value 2,.)
      ```

      Name n and Value n are the payload field names and values that will be used by the UMS business service to construct the message text; for example:

      ```
      Price List = Americas Price List
      ```

Exposing the Waterfall Output

Next, you expose the waterfall output on the user interface.

To expose the waterfall output as a drilldown on a field in the UI

1. Create a new virtual business component based on class CSSBCVWaterfall.
   a. Create fields for each waterfall output column.
   b. Create a field called Name, which is used internally to query the correct data by the waterfall name.
   c. Compile the virtual business component.
2. Create a new pop-up applet based on class CSSSWEFrameListPopupWaterfall using the VBC created in Step 1.
   a. Set its search specification to query the field Name with a value of the created waterfall, such as:

      ```
      Name = "Cost Waterfall"
      ```

   b. Define the column layout.
   c. Compile the applet.

3. Create a drilldown link to the new pop-up applet in each list applet that displays the field calculated by the PSP procedure.
   a. Add a drilldown object to the applet defining the field on which the drilldown is displayed and the drilldown name. Leave all other fields blank.

      Example:

      ```
      Name = "Waterfall Popup 2"; Field = "Cost"
      ```

   b. Add a user property to the applet indicating which waterfall pop-up applet to display upon each drill-down:

      Example:

      ```
      Name = "Waterfall Popup Applet 2"; Value = "Cost Waterfall Popup List Applet"
      ```

      > **Note:** Multiple waterfall drilldown links are supported by incrementing the index at the end of the Waterfall Popup Applet N drilldown name.

      Applet classes CSSSWEFrameListQuoteItemEC and CSSSWEFrameListWaterfall support waterfall drill-down.

      > **Note:** CSSSWEFrameListWaterfall is derived from class CSSSWEFrameListBase directly.

   c. Compile the applet.

Saving Waterfall Data

Waterfall data is saved using Data Transfer Utility (DTU). You can configure the application to save waterfall records manually or automatically as described in this topic.

To enable persistence of the new waterfall data in the database

1. (One option is to do this by using a command.) In the Siebel application, navigate to the Administration - Order Management screen, then the Signals view, and add a new signal to save the waterfall output. Use SaveWaterfall-Order as an example.
   a. In Siebel Tools, add a command to invoke the signal when the user selects a menu option. Use SaveWaterfall-Order as an example. Values are described as follows:

      Display Name: [Name to be displayed]

      Name: [Name of the command]

      Method: [Signal Name]
   b. Create a custom table to store the waterfall output (for example, CX_COST_WATERFALL).
   c. Create a business component (BC) based on the table with the same field names as in the virtual business component used to display the waterfall.
   d. Create a data map object. Use Order Waterfall Data Map Object as an example.

      In the Siebel application, navigate to the Administration - Application screen, then the Data Map Administration view, and create a new data map object.

      The source is the VBC and the destination is the BC.

2. (Another option is to save the waterfall data manually by using an applet menu button or an applet button.) As in Step 1, add a new signal to save the waterfall output.
   a. In Siebel Tools, add an applet menu button (call it Save Waterfall).
   b. Add an object to Applet Method Menu Item of the waterfall-triggering applet, and then expose it to the applet. Values are as follows:
   Command Name: [The command added to save this waterfall]
   Text: Save Waterfall
   Alternatively, you can add an applet button (Save Waterfall) in Siebel Tools by adding a control to the waterfall-triggering applet and then exposing it to the applet. Use the following values:
   Name: Save Waterfall
   Method Invoked: [The signal added to save this waterfall]
   Caption: Save Waterfall

For more information about creating and using signals, see *Signals* For more information about DTU, see the topic about Data Transfer Utility in *Siebel Finance Guide* .

PSP Waterfall Business Service Methods

The PSP Waterfall business service provides the methods described in the following table.

Method	Arguments	Description
ShowWaterfallPopup	Name = Popup Applet Name Value = the name of the waterfall popup applet Example: “Quote Line Item Waterfall Popup List Applet”	Display the waterfall pop-up for the current line item.
SyncToDB	See *SyncToDB Input Arguments*	Generate waterfall records for the current quote, order, or agreement and write them to the database. For more information, see *Saving Waterfall Data*.

SyncToDB Input Arguments

The SyncToDB method provides the input arguments described in the following table.

> **Note:** You may prefer to use SyncToDB in a signal, rather than directly in a command, for synchronizing waterfall data to the database. If so, use the guidance provided in *Saving Waterfall Data*.

Argument	Type	Value
Pricing Output Row Set Type	String	RowSet

Argument	Type	Value
Waterfall Data Map Object	String	Quote Waterfall Data Map Object
Waterfall Name Field	String	Name
Waterfall Parent Id Field	String	Item Id
Waterfall Parent Id Variable	String	ID
Waterfall Signal	String	PSPWaterfallAll
Waterfall Synch Process	String	PSP Waterfall Synch to DB Workflow
Waterfall Sequence Number Field	String	Sequence Num

7 Unified Messaging

Unified Messaging

This chapter describes the Unified Messaging framework used by Siebel Business Applications. It includes the following topics:

- *About Unified Messaging*
- *Components of Unified Messaging*
- *Unified Messaging Service Business Service Methods*
- *Creating Message Types*
- *Configuring the Display of Messages*
- *Implementing Multilingual Substituted Text*
- *Implementing a Custom Message-Generation Engine*
- *About Working with Message Responses*
- *About Suppressing Duplicate Messages*
- *Suppressing Duplicate Messages*
- *Migrating Message Types Between Environments*
- *Tuning Performance of Unified Messaging*
- *Using Unified Messaging with the PSP Engine*

About Unified Messaging

The Unified Messaging framework is the mechanism used by Siebel Business Applications to display messages to users. The foundation of the Unified Messaging framework is the Unified Messaging Service (UMS) business service. For Siebel order management uses, UMS messages recommend products and promotions, explain eligibility, provide price waterfalls, and display results of checks on promotion commitments and integrity. For example, a message prompts a customer service representative (CSR) to cross-sell batteries and a camera case when a customer is purchasing a camera. Messages come to the user in the form of a pop-up applet or rows in a list applet in a view.

The Unified Messaging framework supports the display of dynamic, actionable messages. The framework is an entity independent of the source and type of messages displayed. The Unified Messaging framework natively supports advanced features such as translation of message text, substitution of textual values into the message template, logging of message responses, and suppression of duplicate messages when appropriate (such as advice to a CSR against trying the same upsell if the customer has already rejected it).

In Siebel order management, a message is guidance, a recommendation, or an explanation presented to an end user in response to a button click or an action the end user takes. For example, the Order Catalog view might display the following message when an end user orders an item that is temporarily unavailable:

```
The product you have selected is on back order until [date].
```

A message's text is constructed from a fixed message template as well as substitutable text fragments, such as the name of a product.

Messages displayed are sorted by score. The message-generation algorithm sets the score. For the simple preconfigured rules-based messages, the user enters the score in an administration view. Make sure that your messages use a consistent scoring scheme so that the most important messages of any type appear at the start of the list. The default message-generation algorithms can be extended to call out to a propensity-based scoring algorithm to dynamically score the messages that are displayed based on self-learned rules.

Concepts of Unified Messaging

A *message* type stores the definition of a message, including its text, its bitmap (such as a graphic of an exclamation point), and its display mode (active or passive). A message type with an active display mode means that a dialog box is displayed to the user and the user must provide a response; a passive message type means that the message is displayed to the user without requiring a response. A message type group is a list of values used to group message types, making them available in picklists.

Payload is the contents of the message delivered to the end user. The message type payload is a list of all payload fields that must be provided with every message of a certain type.

A message type response is an allowed response to a message. The allowed responses and their associated actions are soft-coded in an administration view. For example, a preconfigured upsell message has "Accept" and "Reject" responses. You can configure an additional "Send E-Mail" response that automatically emails the upsell product details to the customer.

A message response is a logged response to a message.

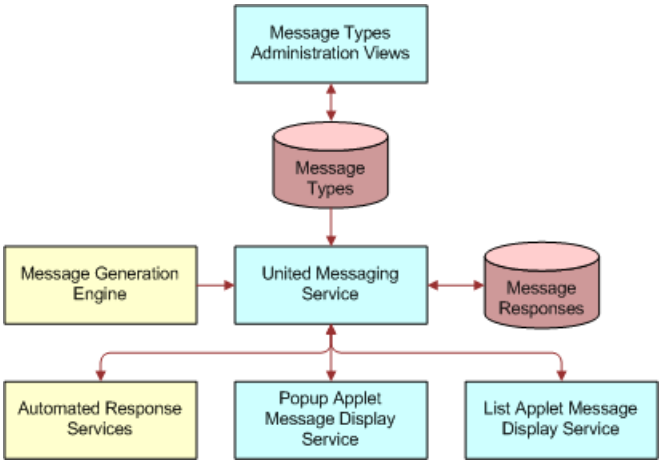
Message Types Administration

You implement Unified Messaging for Siebel order management using the Administration - Order Management screen, then the Message Types views, as follows:

- **All Message Types view.** Create new message types. Work with existing message types. Enter text for messages. Attach titles and bitmaps to messages. Specify whether messages are active or passive. Specify the group to which the message type belongs.
- **Payload view.** Create and work with payload variables, which substitute values into message text and which are used for automated response processing.
- **Responses view.** Create the possible responses for a message type. Control response logging. Control message suppression. Enable automated processing of responses.
- **Translations view.** Enter translations for message text.

Components of Unified Messaging

The following figure shows the interaction between the components of Unified Messaging.

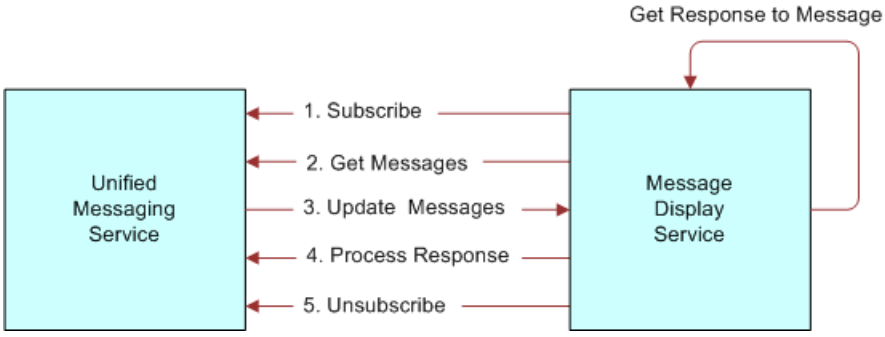


Registered Message Display Services

The UMS business service relies on pluggable message display services that are responsible for displaying messages and accepting user responses. These message display services are the following:

- List Applet Message Display service
- Pop-up Applet Message Display service

The following figure shows the interaction between UMS and registered message display services.



As shown in this image, the interaction between UMS and registered message display service (MDS) is as follows:

1. Subscribe: MDS to UMS.
2. Get Messages: MDS to UMS.
3. Update Messages: UMS to MDS.
4. Process Response: UMS to MDS.
5. Unsubscribe: MDS to UMS.

In this image, Unified Messaging Service and Message Display Service are connected by a numbered list with arrows of service communications that occur between the two entities: 1. Subscribe, 2. Get Message, 3. Update Messages, 4. Process Response, 5. Unsubscribe. In 1–2 and 4–5, arrows are pointing from Message Display Service to Unified Messaging Service. In 3, the arrow points from Unified Messaging Service to Message Display Service. There is also an arrow from Message Display Service that points back to itself with the label Get Response to Message

Message display services subscribe to updates to the message cache in the UMS. A message display service subscribes when it is included in the user interface. A message display service unsubscribes when it is not included in the user interface. More than one message display service can be subscribed at the same time.

Update Messages Method

The UMS calls the Update Messages method on the message display service every time the message cache is updated.

Process Response Method

The message display service invokes the Process Response method on the UMS when the user selects a response in the UI.

For more information about UMS business service methods, see *Unified Messaging Service Business Service Methods*.

Custom Message Display Services

If necessary, you can develop your own message display service for use with the UMS as long as it adheres to the protocol of method invocations described in *Registered Message Display Services*. Using a custom message display service, you may choose to provide a different UI layout or a filter to display only a subset of messages.

Payload Variables

A payload variable is a name-value pair associated with a message instance. It can be substituted into the final message text, saved to the database when a response is logged, used as the scope for duplicate message suppression, or used to process a response. For example:

```
[Product Id] = "12-E2345", [Product] = "Canon F150", [Account] = "Marriott HQ"
```

The payload property set is passed with the message to message display services and automated response services. The payload structure is a set of name-value pairs.

Any payload variable can be substituted into the text by enclosing its name in []. Any payload variable can be logged to the response table as long as a Response Field mapping is specified.

Payload variables are also used for message suppression. See *About Suppressing Duplicate Messages*.

You create payload variables using the Administration - Order Management screen, Message Types, and then the Payload view.

Message Property Set

Characteristics of a message's structure are described in the following table.

Attribute	Type	Description
Message Id	String	A unique identifier for the message. This is used to make sure the same message does not display twice. Every time a message is displayed, it must be assigned the same message ID to enable suppression of repeated messages. This is also used to identify the message when an action is selected.

Attribute	Type	Description
Message Type	String	The message type of the message. Refers to a row in the Message Types table.
Score	Number	An integer between 1 and 100 indicating the priority of the message. This is used to determine the order in which passive messages are displayed.
Display Mode	String	(Optional) Indicates how to display the message. Seed data values are “Active” and “Passive”. You (the integrator) can extend this list. **Note:** This input is optional. If not specified for a particular message, then the Display Mode defaults from the message type.

[Payload]	Property Set	A list of name-value pairs that are used to specialize the display and processing of the message.
		
		Examples:
		
		`Line Item Id = "12-ABC123"`
		`Upsell Product = "Canon 128 MB Memory Card"`
		
		The payload attributes are defined in the Message Type administration view. They are stored as sibling properties of Message Id, Score, and so on.

Message List Property Set

The example that follows shows the structure of a sample message list called “Message List.”

Note: Lines in boldface are required properties.

This sample message list consists of a set of messages. Each message has a child property set providing the payload properties.

```
CCFPropertySet@0013B0AC p#1 c#1 type="" vt=0 value=""
{
 p["Source"] = "Product Recommendation";
 c[0] CCFPropertySet@0BC04D00 p#0 c#2 type="Message List" vt=0 value=""
 {
 c[0] CCFPropertySet@0BDE0408 p#61 c#2 type="Message" vt=0 value=""
 {
 p["Message Type"] = "Cross-Sell Recommendation";
 p["Score"] = "66";
 p["Message Id"] = "42-4Z1RL";
 p["Display Mode"] = "Passive";

 c[0] CCFPropertySet@0BF0E578 p#14 c#0 type="Payload" vt=0 value=""
 {
 p["Doc Id"] = "42-528T1";
 p["Account Id"] = "";
```

```
p["Related Product Id"] = "99-28GJ1";
p["Contact Id"] = "";
p["Line Item Id"] = "42-528VT";
p["Related Product"] = "10GB Hard Drive";
p["Reason"] = "Test XSell";
p["Campaign Id"] = "";
p["Net Price"] = "";
p["Prod Id"] = "98-4NVN0";
p["Currency Symbol"] = "$";
p["Document Type"] = "Quote";
p["Product"] = "1006667";
p["Party Id"] = "";
}
}
c[1] CCFPropertySet@0BD18070 p#61 c#2 type="Message" vt=0 value=""
{
p["Message Type"] = "Upsell Recommendation";
p["Message Type Id"] = "04-E8VXZ";
p["Score"] = "77";
p["Message Id"] = "42-4Z1RN";

c[0] CCFPropertySet@0BE4B178 p#14 c#0 type="Payload" vt=0 value=""
{
p["Doc Id"] = "42-528T1";
p["Account Id"] = "";
p["Related Product Id"] = "99-28GSH";
p["Contact Id"] = "";
p["Line Item Id"] = "42-528VT";
p["Related Product"] = "10MB USB Home Networking Adapter";
p["Reason"] = "Test UpSell";
p["Campaign Id"] = "";
p["Net Price"] = "";
p["Prod Id"] = "98-4NVN0";
p["Currency Symbol"] = "$";
p["Document Type"] = "Quote";
p["Product"] = "1006667";
p["Party Id"] = "";
}
}
}
}
```

Message Responses

A message type with an active display mode means that a dialog box is displayed to the user, and the user must provide a response. A message type response is an allowed response to a message. Defining message type responses is part of the process of creating a message type.

You create message type responses using the Administration - Order Management screen, Message Types, and then the Responses view. For information about creating message type responses, see *Creating Message Types*.

Message Translations

The Unified Messaging framework allows for translations of message text and message response text. If you are implementing message translations, you enter the translations as part of the process of creating a message type. You

create message type translations using the Administration - Order Management screen, Message Types, and then the Translations view. For information about creating message translations, see *Creating Message Types*.

Unified Messaging Service Business Service Methods

The Unified Messaging Service (UMS) business service exposes the APIs described in the following table for updating the messages in the UMS cache, formatting messages, and attaching a new message display service. These APIs can be called from any run-time event, signal, workflow, or custom script.

Method	Arguments	Description
Add Messages	[in] Source: String [in] Message List: Hierarchy	Add a new list of messages to the message cache. Associate each message with the specified source.
Update Messages	[in] Source: String [in] Message List: Hierarchy	Replace the current set of cached messages for the specified source with the new list of messages. Associate each message with the specified source.
Delete Messages	[in] Source: String [in] Message List: Hierarchy	Delete all cached messages associated with the source or, if specified, the list of messages provided. **Note:** For deletion, only the message IDs need to be identified in the message list.
Get Messages	[out] Message List: Hierarchy	Output all unsuppressed, cached messages (regardless of source) including information derived from the message type (such as allowed responses).
Process Response	[in] Source: String Message Id: String [in] Response: String	Process the end-user response to the specified message as defined by the message type.
Reset	Not applicable	Delete all cached messages.
Subscribe	[in] Business Service: String [in] Method Name: String	Add a Message Display business service to the list of services that are informed when a change occurs to the set of messages in the cache.
Unsubscribe	[in] Business Service: String	Remove the specified business service from the list of services that are informed when a change occurs to the set of messages in the cache.
Format Message	[in/out] Message: Hierarchy	Substitute and translate the text for the input message. The message is not displayed.

Creating Message Types

A message type stores the definition of a message, including its text, its bitmap, and its display mode. The process of creating a message type involves adding a new message type record, adding message translations, defining message type responses, and creating translations for message type responses.

To implement a new message type

1. Navigate to the Administration - Order Management screen, then the Message Types view.
2. In the All Message Types list applet, create a new message type record.
3. Complete the fields for the new message type record. Message Type fields are described in the table that follows.

Field	Comments
Title	Enter the title that will be displayed on the popup applet for an active message. For example: `Recommendation`
Display Mode	Select the display mode. Options are: ○ **Passive.** The message is displayed on the screen, but it does not demand a response from the user. ○ **Active.** The message is displayed in a dialog box, and the user must select a response (such as “Accept” or “Reject”) to close the message and continue.
Group	Use this field to group related messages together. This field constrains pick applets in administration views, such as in the Administration - Product screen, then the Product Recommendations view.
Bitmap	Select the graphic that will be displayed on this message. For example, you might display an exclamation point graphic for an alert. **Note:** The Bitmap field applies only to messages with active display mode.
Short Text	Enter a short message to be displayed, using text and field names. For example: `We recommend [Related Product].` **Note:** Short Text is generally not used, except by a custom message display service.
Full Text	Enter a message to be displayed, using text and field names. For example: `We recommend [Related Product] for [Net Price]. [Reason]`

4. (Optional) If this text must be translated, click the Translations tab.
    - Add records to the Translations list, and enter the translation for the content of the Full Text field. You must add one record for each language being translated. Translations fields are described in the table that follows.

Field	Comments
Language	Enter the code for the language of this translation. For example, enter FRA for French.
Title	Enter the title that will be displayed on the popup applet for an active message. For example: `Un Recommendation`
Short Text	Enter a short message to be displayed, using text and field names. For example: `Nous recommendons [Related Product].` **Note:** Short Text is generally not used, except by a custom message display service.
Full Text	Enter a message to be displayed, using text and field names. For example: `Nous recommendons [Related Product] pour [Net Price]. [Reason]`

5. Click the Payload tab, and specify the payload variables to be substituted into the Full Text field. Payload fields are described in the table that follows.

Field	Comments
Payload	Enter the name of the payload variable to be substituted with text. For example: `Campaign Id`
Response	Choose from a picklist of values built from the fields of the UMS Response business component (which is based on the S_COMMUNICATION table). For example: `Campaign Offer Id` The response field is the field used to record the user response to messages.

6. Create a response for the message type.
    a. Write a custom business service to process the new response.

       For information about how to write a response-handler business service, see *Attaching a Business Service to a Message Response*.

       For example, the custom business service creates an activity to mail product literature to the customer.
    b. Compile a new SRF.
    c. In the Administration - Order Management screen, then the Message Types view, select the message type for which you are creating a response.
    d. Click the Responses tab.
    e. Add the new response to the message type:
        - Complete the required fields: Name, Business Service, Method.
        - Set the Log flag to indicate whether to log this response.
        - Set the Suppress Repetition flag and its corresponding field, if this response causes suppression of the message.
        - Resequence the existing responses so that buttons appear in the correct sequence on the user interface.
7. Create translations for the message type response.

   Translations for the message type response make sure that the text displayed on the actionable buttons is in the correct language.

   > **Note:** Translation records are not required if your implementation is mono-lingual. For single-language implementations, the default text in the message type and in the message type response is used.

8. Test the application by creating a situation where the message is displayed, and then clicking New Response.

Configuring the Display of Messages

When configuring the display of messages, you can choose between two preconfigured mechanisms stored in the UMS, as follows:

- **Add a list applet to a view.** If you want the messages to be displayed at all times, then include the UMF Messages list applet in a view.
- **Expose an icon on an applet.** If you do not want to give up the screen space that an applet requires, then expose the UMF “You've got messages” icon on an applet.

Adding a Message Applet to a View

Use the following procedure to add a message applet to a view.

To add a message applet to a view

1. Add the UMF Message List Applet - SI to the view.
2. Add the UMF Passive Virtual Business Component to the Business Object of the view.

Adding a Message Icon to a View

Use the following procedure to a message icon to a view.

To add a “You’ve got messages” icon to a view

1. Create a new button with the following settings:
    - HTML Bitmap = ICON_TOOLBAR_MSGS
    - HTML Disabled Bitmap= ICON_TOOLBAR_MSGS_OFF
    - HTML Type = MiniButton
    - Method Invoked = MessagePopup
2. Add a user property with the following settings:
    - Name = “Named Method 1: Message Popup”
    - Value = 'INVOKESVC','UMF UI Service','PopupListApplet'
3. Add a user property with the following settings:
    - Name = 'Named Method Check Can Invoke’
    - Submethods = MessagePopup
    - Value = Y

Implementing Multilingual Substituted Text

The UMS automatically translates the Title, Short Text, and Full Text for a message. Payload fields must be either language-independent (such as Price or Product ID), or they must be translated by the message-generation engine. In this case, you must create translatable payload text for the message.

To implement multilingual substituted text

1. Add a custom child business component and associated administration view to the payload entity to store the language overrides for each language code.
2. Add a calculated field to the payload business component to store the system language.
3. Add an outer join from the payload business component to the translation business component with a search specification that matches the parent Id and the system language code.
4. Add a joined field for the translated field to the payload business component.
5. Add a calculated field that selects the translated text if it is not null or if it otherwise defaults to the default language text in the payload business component (example: `IfNull([Language Text], [Text])`).

The message-generation engine uses the calculated field value (example: [Language Text]) to get translated text where available.

Implementing a Custom Message-Generation Engine

Any workflow or custom script can add, update, or delete messages stored and displayed by the Unified Messaging framework. The PSP engine provides a framework for implementing a custom message-generation engine. For example, you might implement a PSP procedure to generate disclosures that must be read before a product can be sold (such as “Are you 21”) and then display those messages via Unified Messaging. This topic provides the general approach to implementing a custom message-generation engine.

To implement a custom message-generation engine

1. Create new message types with associated responses and automated execution business services.
2. Create a PSP procedure that generates a Message List property set and passes that list to the UMS using the Update Messages method.
    a. Use a new source name to distinguish these messages from others that are similar.
    b. Make sure all payload variables are populated.
    c. Make sure each message ID is unique and invariant.
3. Set up run-time events or signals to execute the new message-generation engine at the desired points in the UI process flow.

    Note: Make sure to set up events that clear messages when they no longer apply.

4. Add the Message Display list applet or enable the Pop-Up Message list applet for the views where the new messages are relevant.

About Working with Message Responses

For each message type, you can specify multiple possible responses. Each response is displayed as a button labeled with the Name text (or a language-specific translation of the Name text). Buttons are displayed from left to right, sorted by Sequence #.

Further information about message responses is organized as follows:

- *Logging Message Responses*
- *Attaching a Business Service to a Message Response*

Logging Message Responses

You set logging of message responses using the Administration - Order Management screen, Message Types, and then the Responses view.

In the Responses list applet, use the Log flag to indicate whether responses are to be logged for a message type. You can use this for suppression of duplicate messages and to analyze the effectiveness of messaging.

Responses are logged to the S_COMMUNICATION table. This table stores marketing campaign responses. and you can view message responses in the existing campaign analysis views.

With response logging enabled, the following message fields are logged:

- Message Type Id
- Message Id
- Response
- Display Mode
- Score
- Language Code
- Position In Message List
- All payload fields with an associated Response Field mapping

Attaching a Business Service to a Message Response

When the user selects a message type response (for example, by clicking the Accept button for an upsell message), the active message display service informs the UMS by calling its Process Response method. The UMS then calls the business service method associated with that response.

You define which business service method will handle a response using the Administration - Order Management screen, Message Types, and then the Responses view.

The Siebel application provides prebuilt methods for handling upsell and cross-sell in the Product UpSell CrossSell Service business service. You can implement your own automated response-handling logic (such as for sending an email with product details) by scripting your own business service method. This method must process the Payload argument, as described in the following table.

Argument	Type	Description
Payload	Hierarchy	A property set containing all payload variables provided with the message when it was added to the UMS. The message-generation engine must provide values required to process the message in the payload variables.

About Suppressing Duplicate Messages

Opportunities to communicate a message (such as an upsell) to a customer are limited. It is important to deliver the message that provides the highest likelihood of a new sale. Repeating a previously rejected message to the customer is unlikely to generate a new sale; instead, you are more likely to make a sale by presenting a new message, even if it has a lower score.

The Unified Messaging framework provides a flexible mechanism for suppressing duplicate messages. You can implement message suppression for particular responses (such as, implement suppression for “Reject” but not “Accept”) and for any scope (such as, for an instruction to never show this message again to the customer, or to not show the message again for this order).

Checking for duplicate messages occurs in two instances:

- When new messages are provided to the UMS, (for example, with the Add Messages or Update Messages methods).
- After a new response has been processed (using the Process Response method).

Suppress Repetition Flag

The UMS attempts to suppress duplicate messages if the Suppress Repetition flag is set for one or more of the responses for a message type.

Suppression Scope

The Field column indicates the scope of the message suppression (for example: Party Id, Document Id, or Line Item Id).

> **Note:** Logging must be enabled and the scope variable must be mapped to a field in the response table in order for duplicate suppression to work.

All responses for a particular message suppression scope (such as Party ID) are loaded with a single query and cached until the scope changes, for example, with a new caller.

For further information, see *Suppressing Duplicate Messages*.

Suppressing Duplicate Messages

You set suppression of duplicate messages using the Administration - Order Management screen, Message Types, and then the Responses view.

A message is suppressed if a response has been logged for that message ID, response, and scope value; for example:

```
[Party Id] = '12-W123')
```

For this reason, the message ID must be unique and invariant.

To enable suppression of duplicate messages

1. Navigate to the Administration - Order Management screen, then the Message Types view.
2. In the All Messages Types list applet, select the message type for which you want to suppress duplicates.
3. Click the Responses tab to see the Responses list applet.
4. In the Responses list applet, check the Suppress Repetition flag to set suppression for the message type response.
5. In the Field field, specify the scope of the suppression, for example, the suppression may be limited to the Party ID field.

Migrating Message Types Between Environments

Message types can be exported and imported using the applet menu on the All Message Types list applet.

Tuning Performance of Unified Messaging

Note the following considerations when tuning performance of the UMS:

- Message type definitions are stored in an object manager-level cache. Messages are cached in memory and never written to the database. The Message Display list applet is based on a virtual business component that pulls data directly from the in-memory cache.
- Avoid using duplicate message suppression for business-to-business accounts. Duplicate suppression processing has to load all previous responses for the account at the beginning of the call. Loading more than 100 responses will result in a perceptible delay. If you must use duplicate message suppression in business-to-business situations, configure the message suppression to suppress duplicates by quote or by order instead of by account.
- Limit the number of messages displayed. It is generally accepted that the user will not view more than three or four recommendations at a time.
- Carefully consider the events that trigger the message-generation mechanism. In general, the message-generation mechanism will have a larger overhead than the UMS.

Using Unified Messaging with the PSP Engine

PSP applications such as Pricer and Eligibility use the UMS to format translated, substituted text for waterfall output or for eligibility reasons. The PSP action syntax provides an API, LookUpMessage, that in turn invokes the Format Message method on the UMS. This method simply returns the formatted text. It does not add the message to the UMS cache. Messages formatted in this way do not support automated responses, duplicate suppression, or logging.

To use the new message type with a PSP procedure

1. After completing Step 5 in *Creating Message Types*, add an action to the PSP procedure that invokes the LookUpMessage function for the new message type, passing the payload fields. See *Example of a LookUpMessage Call*.

   The LookUpMessage function allows the UMS to process translations of the message text.
2. Test the revised PSP procedure.

Example of a LookUpMessage Call

You can use the example that follows as a model for an invocation of the LookUpMessage function.

```
{Row.Eligibility Reason} = LookUpMessage('Eligibility - Not In Contract Error',
```

```
[Account] = {Row.Account Id})
```

The PSP engine calls the method Format Message. Arguments for this method are described in the following table.

Method	Arguments	Description
Format Message	[in/out] Message: Hierarchy	Substitute and translate the text for the input message. The message is not displayed.

8 Data Validation Manager

Data Validation Manager

This chapter discusses the data validation manager. It includes the following topics:

- *About Data Validation Manager*
- *Roadmap for Implementing Data Validation Processing*
- *Process of Administering Data Validation Rules*
- *Process of Invoking the Data Validation Manager Business Service*

About Data Validation Manager

Many companies are governed by various regulatory agencies, as well as internal processes and procedures, to verify the quality and accuracy of their transactions. Data validation is a key component of many business processes, and can involve many types of transactions, including orders, applications, claims, and various other service requests.

The Data Validation Manager business service can validate business component data based on a set of rules. In the case of a rule violation, a custom error message appears or a user-defined error code is returned. The validation rules are defined using the Siebel Query Language and can be conveniently constructed using the Personalization Business Rules Designer. The business service centralizes the creation and management of data validation rules without requiring extensive Siebel Tools configuration.

The Data Validation Manager business service reduces the need for custom scripts, decreases implementation costs, and increases application performance.

> **Note:** There is no specific encryption support built into the Data Validation Manager. The Data Validation Manager inherits the CSSQuery function at the business component level, and uses the CSSQuery functionality to actually check the expression.

The Data Validation Manager features:

- Search automatically for the proper rule set to execute based on active business objects and views.
- Write validation rules based on fields from multiple business components.
- Apply a validation rule to a set of child business component records to see if a violation occurs from one or more records.
- Invoke specific actions to be performed as a result of a validation.
- Write validation rules that operate on dynamic data supplied at run time together with data from business component fields.
- Automatic logging of data validation events.

Some example business rules which can be enforced by the Data Validation Manager business service are:

- In an insurance company, claim adjusters are required to enter a closed date whenever they close a claim. If the adjuster tries to close a claim without a closed date, an error message appears and the claim record is not committed to the database.
- In a retail bank, different data validation rules are required for each of dozens of different service request types. When a customer service representative creates a new service request, the Data Validation manager identifies the appropriate validation rule set for the specific type of service request and executes the data validation rules of that rule set.
- At a health insurance company, customer service representatives use activity plans and activities to track service requests, and all activities must be closed before the service request can be closed. When the CSR closes the SR, the DVM loops through all associated activities making sure status of each is closed. If any are still open, the SR cannot be closed.

Roadmap for Implementing Data Validation Processing

To automate data validation processing, perform the following processes:

1. *Process of Administering Data Validation Rules*.
2. *Process of Invoking the Data Validation Manager Business Service*

Process of Administering Data Validation Rules

This process is part of *Roadmap for Implementing Data Validation Processing*.

To support a given data validation business rule in your organization, you first create a data validation rule set. The rule set is a container which has one or more rule set arguments and one or more validation rules. The rules contain expressions which are evaluated as being true or false. If the expression is evaluated as being false, validation rule actions determine the appropriate error handling behavior.

To administer data validation rules, perform the following tasks:

1. *Defining Error Messages for Data Validation*
2. *Defining a Data Validation Rule Set*
3. *Defining Rule Set Arguments*
4. *Defining Validation Rules*
5. *Defining Validation Rule Actions*
6. *Activating a Data Validation Rule Set*

Defining Error Messages for Data Validation

This task is a step in *Process of Administering Data Validation Rules*.

Before defining data validation rules, you must define the error messages that these rules display. When you create a validation rule, you can choose among these messages to specify the error message that the rule displays.

Note: When creating a validation rule, you also have the option of typing in a message for the rule, provided that the Err Msg Txt field on the applet is *not* set to read-only. If the Err Msg Txt field on the applet is set to read-only, then you cannot enter a message when creating a validation rule; instead, you must select a message that you have already created.

To activate a rule set

1. Navigate to Administration - Data Validation, and then Validation Messages view.
2. In the Validation Messages list, add a record for each new rule set and complete the necessary fields. Some fields are described in the table that follows.

Field	Comments
Message Code	Enter a numeric code or error code that will be associated with the rule. This code is an alphanumeric value that the application logs in the validation history record and store in the Return Code output argument of the business service, if the expression is evaluated to be false. The existing value of that output argument will be overwritten. Therefore the Return Code output argument of the business service will contain the Return Code of the last rule that is evaluated as FALSE. Maximum number of characters for return code is 30. For more information, see *Viewing a Validation History*.
Message Level	Enter the level of the error message. This is usually something like Quote or Order, but it can be any text that describes the level.
Message Source	Enter the source of the error. This is the process that generated the error, such as Quote Validation or Quote Approval.
Message Text	Enter the text that is displayed as the error message.

Defining a Data Validation Rule Set

This task is a step in *Process of Administering Data Validation Rules*.

You define validation rule sets in the Administration - Data Validation screen. You can either revise an existing rule set or create a new one.

- *Defining a New Validation Rule Set*
- *Revising an Existing Validation Rule Set*

You can import and export validation rule sets by selecting Export Rule Set and Import Rule Set from the menu button on the Rule Sets list. The validation rule set is saved as an XML file for importing and exporting purposes.

- *Exporting a Validation Rule Set*

- *Importing a Validation Rule Set*

To create a validation rule set, specify the business object and business component you want to validate. The validation rule set will have one or more arguments and contain one or more individual rules.

Defining a New Validation Rule Set

Use the following procedure to define a new validation rule set.

To define a new validation rule set

1. Navigate to Administration - Data Validation, and then Rule Sets view.
2. In the Validation Rule Set list, add a record for each new rule set and complete the necessary fields. Some fields are described in the table that follows.

Field	Comments
Name	Enter a name for this rule set. You can execute a particular rule set by setting this name as the value of the Rule Set Name input argument of the Data Validation Manager.
Version	Displays a numeric value to differentiate various versions of the same rule set. Clicking the Revise button creates a new version of an existing rule set with the version number incremented by one.
Group	Enter the group that this rules set is in. You can group a number of rule sets together by giving them a common group name. You can then execute these rule sets in one call by setting this group name as the value of the Group input argument of the Data Validation Manager. Data Validation Manager executes these rule sets one by one in no particular order. (Note: If both the Rule Set Name and Group input arguments are specified, the Group input argument will be ignored.)
Business Component	Select the business component to be validated.
Status	Displays the rule set status. Options include: o **In Progress.** Default status that appears when the administrator first creates a new rule set or revises an existing rule set. A rule set can only be edited when its status is In Progress. o **Active.** Status that appears when the administrator clicks the Activate button. A rule set can only be invoked when its status is Active. o **Outdated.** Status that appears when the administrator activates a newer version of the rule set. o **Inactive.** Status that appears when the administrator selects Deactivate Rule Set from the applet level menu.
Business Object	Select the business object to be validated. Business object is one of the selection criteria under which a rule set is selected for execution. If the Object Search Type input argument of the business service is set to \Business Object, Data Validation Manager will check if the active business object matches the rule set's Business

Field	Comments
	Object value. If there is a match, the rule set will not be excluded based on the Business Object criteria. It may be selected or excluded based on other criteria. **Note:** This business object must have a primary business component defined.
Start Date	Enter the date when the rule set becomes effective.
End Date	Enter the last date this rule set can be used. If not populated, the rule set never expires.
Conditional Expression	Enter a selection criterion under which a rule set is selected for execution. If the Conditional Expression is specified (not NULL) for a rule set, Data Validation Manager will exclude the rule set from execution if the conditional expression is evaluated to be FALSE at run time. If the Conditional Expression is not specified, it is interpreted as TRUE. Conditional Expression provides a mechanism to perform different validations on the same business component based on certain field values. For example, you might have many different types of service requests, and each type needs to be validated in a different way. Using conditional expressions based on the Service Request type, Data Validation Manager can select the appropriate rule set to execute.
Aggregate Error	When this check box is selected, Data Validation Manager ignores the Immediate Display flag of each rule it processes. It aggregates all the error messages of the rules that are FALSE into one string, and then display the aggregated error message to the end user.

Revising an Existing Validation Rule Set

Use the following procedure to revise an existing validation rule set.

To revise an existing validation rule set

1. Navigate to Administration - Data Validation, and then Rule Sets view.
2. In the Validation Rule Set list, select a rule set and click Revise.

   Clicking Revise creates a new version of the rule set and sets the Status to In Progress.
3. Make the appropriate changes in the Validation Rule Set form and click Activate.

   Clicking Activate changes the Status from In Progress to Active and makes the record read-only. The old validation rule set still appears, but the status is now Outdated.

Note: You can delete a Validation Rule Set in the same way you delete any other record in Siebel Business Applications. When you delete a Validation Rule Set, that rule set's Validation History is also deleted.

Exporting a Validation Rule Set

Use the following procedure to export a validation rule set.

To export a validation rule set

1. Navigate to Administration - Data Validation.
2. In the Validation Rule Set list, select the rule set or rule sets that you want to export.
3. From the applet menu, choose Export Rule Set.
4. Follow the on-screen prompts to save the rule set as an XML file.

Importing a Validation Rule Set

Use the following procedure to import a validation rule set.

To import a validation rule set

1. Navigate to Administration- Data Validation.
2. In the Validation Rule Set list, choose Import Rule Set from the applet menu.
3. In the Validation Rule Set Import dialog box, locate the file you wish to import and click Import.

   The imported rule set appears having a status of In Progress.
4. To activate the imported validation rule set, select it in the Validation Rule Set list and click Activate.

   Clicking Activate changes the rule set Status to Active and makes the record read-only.

Defining Rule Set Arguments

This task is a step in *Process of Administering Data Validation Rules*.

You can write a validation expression of a rule which contains user-defined arguments using a syntax such as:

```
[Some Buscomp Field Name] = [&Argument Name]
```

These arguments must be first defined in the Arguments list. Values of these arguments can be set using the Default Value field. They can also be set at run time by supplying an input argument to the Data Validation Manager business service. The input argument name must be the same as the argument name defined in the Arguments list.

Business service input arguments will overwrite whatever default values you have specified in the Arguments list. The default values only take effect when input arguments are not provided.

You can only define arguments for validation rule sets that have a status of In Progress.

To define a rule set argument

1. Navigate to Administration - Data Validation.
2. In the Rule Sets list, select a rule set with a status of In Progress and drill down on the rule set name.
3. Click the Arguments view tab.

4. In the Arguments list, add a new record, and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Argument Name	A string that specifies the name of the argument. You use the notation [&Argument Name] to refer to the value of the argument in a rule expression.
Default Value	The value that the argument will take on in the absence of a business service input argument of the same name.
Comments	Free text field to provide explanations for the argument.

Defining Validation Rules

This task is a step in *Process of Administering Data Validation Rules*.

For each rule set, you define one or more validation rules. These rules represent the validation criteria.

You can only define rules for validation rule sets that have a status of In Progress.

To define a validation rule

1. Navigate to Administration - Data Validation.
2. In the Rule Sets list, select a rule set with a status of In Progress and drill down on the rule set name.
3. Click the Rules view tab.
4. In the Rules list, add a new record, and complete the necessary fields. Some fields are described in the following table.

Field	Comments
Sequence #	Identifies the numeric sequence of this rule in the rule set. The application evaluates rules in numerical order based on this number.
Business Component	The business component upon which the rule is based.
Expression	A statement expressed in Siebel Query Language. The application evaluates whether the expression is true or false. If true, the data validation manager proceeds to evaluate the next rule. If false, the application performs the actions defined for the rule. You can refer to a business component field value using the notion [Field]. For example, if the business component of the rule is Opportunity, then an expression [Sales Stage] IS NOT NULL means that you want to know of the Sales Stage field of the Opportunity business component contains a value or not.

Field	Comments
	You can use the syntax [BC.Field] to refer to a field of a business component different from the one of the validation rule. For example, you may have a rule which has its business component set as Opportunity. You can write the following expression stating what is valid: `[Sales Stage] IS NOT NULL AND [Account.Status] = "Active"` [Account.Status] refers to the Status field of the Account business component. Without a prefix, [Sales Stage] refers to the Sales Stage field of the business component (Opportunity) of the rule. You can also use the syntax `[&Argument]` to refer to a rule set argument. For example, you may have a rule expression `[&Answer] = "Yes"`. Here the rule set argument Answer has already been defined in the Arguments List Applet. Once defined, the argument becomes a business service input argument which you can populate with dynamic values at run time (for example, through a workflow). You can either enter the statement directly in the field or click the Expression select button to launch the Expression Designer. The Expression Designer allows you to construct an expression by pointing and clicking on a pop up window, perform syntax validation, and lookup definitions of built-in functions supported by the Siebel Query Language. For more information about the Expression Designer, see *Developing and Deploying Siebel Business Applications* .
Message	Displays the text of the error message from the Return Code field. If the expression is evaluated to be false, the application either displays the error message or writes it to a log file. The maximum number of characters is 250. For more information viewing the validation log file, see *Viewing a Validation History*.
Apply To	This field takes on two values: Current Record and All Records. When Current Record is chosen, Data Validation Manager applies the validation rule to the current active business component record. When All Records is chosen, Data Validation Manager applies the validation rule to all business component records. If the business component on the rule is the same as the one on the rule set, then this field is read only. If the business components on the two are not the same, you can choose Current Record or All Records.
Return Code	Select the return code and error message for this rule. The Validation Messages dialog box allows you to select codes and associated error messages that you defined in the step *Defining Error Messages for Data Validation*.

Field	Comments
	CAUTION: If you type a return code rather than selecting if from the dialog box, the error message is not copied into the rule. The error message for this rule will be blank.
Start Date	Corresponds to the time when the rule becomes effective. A rule is only evaluated if the Start Date is equal to or earlier than the current date.
End Date	Specifies the last date this rule can be used. If not populated, the rule set never expires. A rule is only evaluated if the End Date is equal to or later than the current date.

5. After the Rules list applet, click the Rule Detail view tab and complete the necessary fields. Some fields are described in the table that follows.

Field	Comments
Stop on Error	If the expression is evaluated to be false and this field is checked (TRUE), the application will ignore all subsequent rules.
Immediate Display	Defines error message behavior. If the expression is evaluated to be false, and both Immediate Display and Stop on Error flags are checked (enabled), the application immediately displays the specified message. **Note:** If, for the rule set, Aggregate Errors is enabled, the Immediate Display flag for each rule is ignored. Instead, the application aggregates all error messages of the rules that are FALSE into one string, and then display the aggregated error message to the end user.
Message	The text of the error message. If the expression is evaluated to be false, the application either displays the error message or writes it to a log file. The maximum number of characters is 250. For more information viewing the validation log file, see *Viewing a Validation History*.

Defining Validation Rule Actions

This task is a step in *Process of Administering Data Validation Rules*.

The Data Validation Manager business service can perform a sequence of actions when a rule expression is evaluated to be FALSE. Each action can be set to update a business component in the active business object or to execute a business service.

Each action has a sequence number. Data Validation Manager executes the actions in ascending order of their sequence numbers.

To define a data validation rule action

1. Navigate to Administration - Data Validation, and then Rule Sets view.
2. In the Validation Rule Set list, select the rule for which you want to define an action, and drill down on the Name hyperlink.
3. Click the Actions view tab.
4. In the Actions list, add a new record, and complete the necessary fields. Some fields are described in the table that follows.

Field	Comments
Sequence #	Identifies the numeric sequence of this rule in the rule action. The application executes actions in numerical order based on this number.
Type	Determines whether the action is to update the current active business component or execute a business service. Can either be Business Component or Business Service.
Business Component	Name of business component which you want to update. This field is editable only when Type is set to Business Component.
Business Service Name	Name of the business service you want to invoke. This field is editable only when Type is set to Business Service.
Business Service Method	Method of the business service you want to invoke.
Business Service Context	Name - value pairs which you can use to pass certain values as input arguments to the business service. For example, "input argument 1", "value 1", "input argument 2", "value 2".

For each action record of type Business Component, enter the field and value information as described in the table that follows.

Field	Comments
Field	Name of the business component field you want to update.
Value	Value with which you want to update the business component field. This value must be a constant and cannot be an expression.

Activating a Data Validation Rule Set

This task is a step in *Process of Administering Data Validation Rules*.

The final step in administering data validation rules is to activate the rule set. Only then can it be executed by the Data Validation Manager business service.

After you have added and defined all rule set arguments and rules, activate the rule set. After you activate the rule set, it becomes read-only and can not be edited.

You can only define arguments and rules for validation rule sets that have a status of In Progress. If you want to revise an existing rule set, see *Revising an Existing Validation Rule Set*.

To activate a rule set

1. Navigate to Administration - Data Validation.
2. In the Rule Sets list, select the rule set you wish to activate.

   Note: The status of the rule set must be In Progress in order for you to activate it.

3. Click Activate.

   Clicking Activate changes the status of the rule set to Active and makes the record read-only.

Process of Invoking the Data Validation Manager Business Service

This process is part of *Roadmap for Implementing Data Validation Processing*.

You can invoke the Data Validation Manager two different ways:

- *Invoking Data Validation Manager from a Runtime Event*
- *Invoking Data Validation Manager from a Workflow*

In either case, you can affect how the business service works by populating certain input arguments of the business service.

You can view the results of the business service execution by viewing the validation history log (see *Viewing a Validation History*).

Invoking Data Validation Manager from a Runtime Event

This task is a step in *Process of Invoking the Data Validation Manager Business Service*.

You can invoke the Data Validation Manager business service from a runtime event. When the specified runtime event occurs, the application invokes the business service. To invoke the business service from an event, you first define the event in the Administration - Runtime Events view. For more information about runtime events, see *Siebel Personalization Administration Guide* .

To define a runtime event to invoke the Data Validation Manager

1. Navigate to Administration - Runtime Events, and then Action Sets.
2. In the Actions Sets list, add a record and complete the necessary fields.

   Some fields are described in the table that follows.

Field	Comments
Action Type	Specifies the type of action. Select BusService.
Sequence	Number describing the order in which the action occurs. Execution begins with the action with the lowest sequence number. Actions with the same sequence number are executed in random order. Actions occur in sequence until all actions are completed.
Active	Check the box to indicate whether the action will be triggered or not. Inactive actions are ignored when the event occurs. This is a quick way to turn off an action without changing the start and end dates.

3. In the More Info form, complete the fields using the values described in the table that follows.

Field	Value	Comments
Business Service Name	Data Validation Manager	Name of the business service to invoke, if the conditional expression evaluates to TRUE and the type is BusService. Enter the value exactly as shown.
Business Service Method	Validate	Method to invoke on the business service. Enter the value exactly as shown.
Business Service Context	Example: "Rule Set Name", "Validation", "Enable Log", "Y" These input arguments are equivalent to those presented in Step 2.	Name-value pairs to specify the inputs to the business service method. Both the name and the value must be enclosed by quotation marks and separated by a comma and a space after the comma.

> **CAUTION:** Failure to use the syntax specified in the Business Service Context field may result in errors.

4. In the link bar, click Events to associate an event with the action set.
5. In the Events list, add a record and complete the fields as described in the table that follows.

Field	Comments
Name	Optional. Select an event alias from the drop-down list. Selecting a name automatically populates the Object Type, Object Name, Event, and Subevent fields. This is based on the event template created in the Event Aliases list. For more information about creating event aliases, see *Siebel Personalization Administration Guide* .
Sequence	Required. An event can trigger multiple action sets. Enter numbers in this field to control when the action set associated with this event in this record executes relative to other action sets associated with this event.
Object Type	Required. Select BusComp from the drop-down list.
Object Name	The name of the application, business component, or applet (depending on the object type) to which the event occurs.
Event	Required. Select from the drop-down list. The choices depend on which object type you choose. Valid values include: o Use the PreWriteRecord business component event if you want to control whether a record can be written to the database, based on the outcome from the validation. o Use the PreDeleteRecord business component event if you want to control whether a record can be deleted from the database, based on the outcome from the validation.
Action Set	Required. Select an action set to run when the event occurs. The Name name is defined in the Action Sets view tab. For more information, see Step 3.

6. Either close down and relaunch the server or mobile clients, or select Reload Runtime Events from the applet menu.

Invoking Data Validation Manager from a Workflow

This task is a step in *Process of Invoking the Data Validation Manager Business Service*.

You can invoke the Data Validation Manager business service from a workflow. This topic describes some of the possible steps you can include to enable this invocation. You may need to modify and expand on this procedure to accommodate more complex business requirements. The workflow process you create must contain the following steps:

- **Start.** Initiates the process instance. When the conditions have been met, the application initiates the process instance.
- **Business Service.** A step in a process in which an automated call is made to the Data Validation Manager service. A workflow process definition can have one or more business service steps.
- **End.** A step in a process that specifies when a process instance is finished.

Note: The workflow that makes a call to the Data Validation Service must be invoked from a runtime event. If it is invoked from a script, the script passes no record context to the Data Validation Manager. The context is passed only with a runtime event. Thus, if you try to invoke a workflow using a script, the child business component context is not passed to Data Validation Manager, so it cannot validate the data correctly.

For more information about how to create a start step, business service step, and end step in a workflow, see *Siebel Business Process Framework: Workflow Guide* .

To invoke Data Validation Manager from a workflow

1. Create the workflow in Siebel Tools.
2. When you create the business service step, include the following information:
   a. In the Business Service form, complete the fields described in the following table.

Field	Value
Business Service	Data Validation Manager
Method	Validate

   b. In the Input Arguments list, create new records to establish your Input Arguments as described in the following table.

Input Argument	Comments
Active Object	Can attain a value of Y or N. If the value is N or if this input argument is not entered into the list applet, the Business Object and Object Id input arguments must be established and cannot be NULL.
BusObj	The name of the business object (that is, the functional area) to which the event occurs. It is required if an Active Object has not been specified or has a value of N. By default, the business service uses the primary business component of the business object (if defined).
Enable Log	Valid options include: - Y - Application logs all instances when the rule set runs. - N - Application does not track any instances of when the rule set runs. For more information about the Validation log file, see *Viewing a Validation History*.
Object Id	The row ID of the principle business component of the business object. It is required if an Active Object has not been specified or has a value of N.

Input Argument	Comments
Object Search Type	Value is View or Business Object and determines which of these two arguments is used as criteria.
Group	Group name to which to restrict data validation rule set selection.
Rule Set Name	In the Value field, enter the name of the rule set to be invoked. For more information, see *Defining a New Validation Rule Set*.

If the data is valid, both the Return Code and the Return Message field remain empty.

Note: If the data is invalid, in addition to filling in the Return Code and Return Message, the workflow engine also generates the generic error message "Error invoking business service." This error message is an expected result of how the workflow engine treats content populated into the Error Message process property by a business service. It does not indicate that the Data Validation Manager failed.

Viewing a Validation History

You can view a history of validation events in the Validation History view. All events display in chronological order.

To view the validation history

- Navigate to Administration - Data Validation, and then Validation History view.

  The Validation History view appears, displaying validation events. Some fields are described in the table that follows.

Field	Comments
BusComp Name	The business component that was validated.
Date	The date the validation event happened.
Last Step #	Sequence number of the rule evaluated to be false or the last rule in the rule set.
Return Code	The rule's Return Code field value.
Return Message	The rule's Message field value.
Started By	The login name of the user who executed this rule.

Field	Comments
Status	Specifies the status of the validation result: ○ **Errored Out.** Indicates Stop on Error is True. The current rule is evaluated to be false and further rule evaluation is halted. ○ **Error Proceed.** Indicates Stop on Error is False; the current rule is evaluated to be false and the application proceeds to evaluate the next rule. ○ **Completed.** Indicates the application has reached the last rule of the rule set and the rule is evaluated to be True.

For more information about defining rule sets and configuring the Return Code and Return Message fields, see *Defining Validation Rules*.

9 Approvals Manager

Approvals Manager

Using the Approvals engine, you can define a set of approvers who must approve a requested item. The Approvals engine processes the defined set of approvers by notifying each approver with a Universal Inbox record or an email.

Approval processes happen in one of two flow types: sequential or parallel. For sequential approval processes, the Approvals engine maintains the context of a specified sequence of approvers, and alerts the next approver after the previous approver has approved the request. For parallel approval processes, multiple approvers can take action at the same time.

The Approvals engine allows the requester to monitor the approval process using the Approvals view in the Quotes, Orders, and Agreements screens. Navigate to the Administration - Application screen, then the Approval Admin view to access the Approvals view.

In order to integrate Universal Inbox with Siebel order management, you use the Approvals view and the Approval Manager business service.

This chapter includes the following topics:

- *About Approval Processing*
- *ISS Approval Business Service Methods*
- *Defining Approval Items and Approval Stages*
- *About Invoking the Approvals Manager Business Service from a Workflow*
- *Approving or Declining Approval Stages (End User)*

About Approval Processing

In Siebel order management, administrators can define a number of approval levels without the need for programming, scripting, or configuring. You can define both basic or multiple-step approval processing levels based on the needs of your organization. You can invoke approval processing from a script, a workflow, or a run-time event.

Approval Item

An *approval item* is an approval process invoked by the Approvals engine. An approval item can be one of two types: parallel or sequential. An approval item with an Approval Flow Type of *Parallel* is set so that all approvers receive the approval request at the same time. An approval item with an Approval Flow Type of *Sequential* is set so that each approver receives the approval request only after the prior person approves. If an approver rejects the request, no other approvers further along in the sequence sees the approval request.

Approval Stage

The *approval stage* is the set of individuals who must approve the approval item. An approval item can have multiple approval stages.

Approval Types

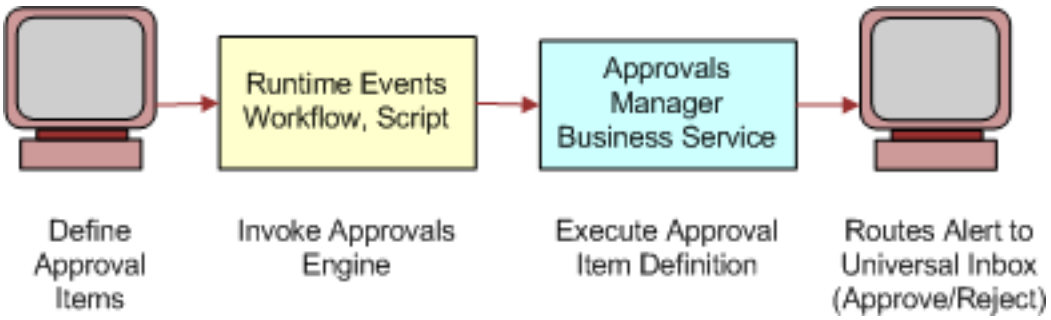
The individual that is set as an approver can be one of the following approval types:

- Employee. A specific employee within the organization.
- Position. A general position within the organization.
- Either of the aforementioned. Employee and Position are both multi-value groups. The third option is Either. If there is more than one owner or position specified against an approval item, then either one of them is allowed to approve or deny the Inbox item.
- **Note:** If your organization defines the relationship between Employee and Position as 1:1, the distinction between Approval Types is irrelevant. If your organization chooses to use the approval type Position, only the primary employee for the position is alerted.

Automating an approval process involves the following steps:

- **Administrator defines approval items and approval stages.** For more information, see *Defining Approval Items and Approval Stages*.
- **Administrator invokes approval processing.** For more information, see *About Invoking the Approvals Manager Business Service from a Workflow*.
- **End users approve or decline an approval stage.** For more information, see *Approving or Declining Approval Stages (End User)*.

Interaction of the Approvals engine parts is shown in the following figure.



ISS Approval Business Service Methods

The ISS Approval business service has the following APIs:

- *CreateNewApprovalTasks Method*

- *GetApprovalStatus Method*
- *SetApprovalDecision Method*

CreateNewApprovalTasks Method

This method creates new approval task instances by copying a template from Approval Stage. You must pass in the Approval Level Name, Requesting Bus Comp, Inbox Type, and Object ID.

For more information about this method, including a description of all method arguments, see the topic about order management interface methods reference in *Siebel Order Management Guide* .

GetApprovalStatus Method

This method returns the approval status for the inbox item. The status is one of the following: Approved, Declined, or In Progress. You must pass in the Approval Level Name, Approval Item ID, Inbox Type, and Object Id.

For more information about this method, including a description of all method arguments, see the topic about order management interface methods reference in *Siebel Order Management Guide* .

SetApprovalDecision Method

This method sets the approval status for a given stage level. You must pass in the Stage ID, Inbox Type, Object ID, Seq Num, Inbox Item Id, Owner Info Id, and Action LIC.

> **Note:** Only Approval Type EMPLOYEE is supported. Position-based and Either approval is not supported with the Universal Inbox. These are only supported with the My Approval Inbox (Siebel Industry Applications). However, Position Type Approval is supported in the sense that it dynamically routes the approval to the primary employee for the position. But it is not the standard Siebel position-based visibility.

For more information about this method, including a description of all method arguments, see the topic about order management interface methods reference in *Siebel Order Management Guide* .

Defining Approval Items and Approval Stages

An administrator defines the approval process by creating approval items and approval stages using the Administration - Application screen, then the Approval Admin view.

To define approval items and stages

1. Navigate to the Administration - Application screen, then the Approval Admin view.
2. In the Approval Item list, add a record and enter a name in the Approval Item field.

3. In the Approval Flow Type field, select one of the following:
    - **Sequential.** Indicates this approval item is distributed to approvers one after another in the sequence specified in the Approval Stage list applet. The application routes the approval item to the next approver only if the current approver approves the request. If any one approver in the approval chain declines the request, the approval item is rejected, and no further routing is conducted.
    - **Parallel.** Indicates this approval item is distributed to all approvers simultaneously for approval. The approval item is rejected if at least one approver declines the approval request.

    After you have defined an approval item, the next step is to define the appropriate approval stages. The Approval Item and Approval Stage list applets have a parent-child relationship.
4. In the Approval Stage list, add a record for each approval stage and complete the necessary fields.

    Some fields are described in the following table.

Field	Description
Sequence #	Identifies the numeric sequence of this approval stage in the current approval item. The application executes approval stages in numerical order based on this number.
Approval Type	Specifies whether the approver is a position or an employee.
Owner Login Name	Indicates the login name tied to this approval stage. Relevant only if Approval Type is Employee.
Owner Position	Indicates the position tied to this approval stage. Relevant only if Approval Type is Position.

End users use the Inbox screen to approve an approval item. For more information, see *Approving or Declining Approval Stages (End User)*.

About Invoking the Approvals Manager Business Service from a Workflow

You can invoke the FINS Approval Item Service from a seed data workflow. To do this, click the Generate Approvals menu option on the Quote, Order, or Agreement header applet. The workflow shown in the following is the one invoked through the signal ApproveItem.

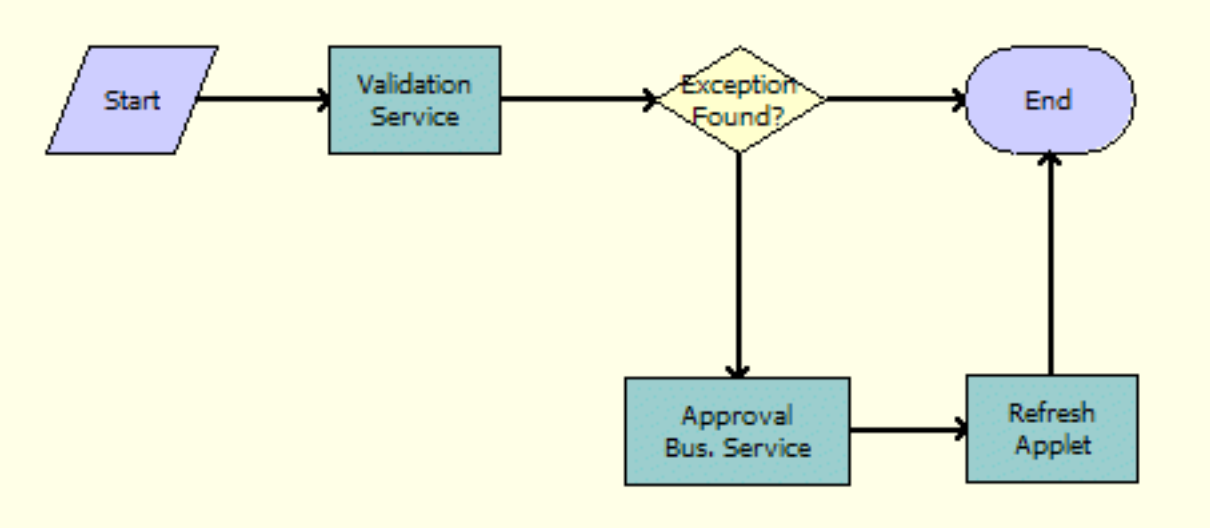
You can also invoke the FINS Approval Item Service from a workflow by creating a workflow process in Siebel Tools containing the following steps:

- **Start.** Initiates the process instance. When the conditions have been met, the application initiates the process instance. See *Configuring the Start Step for a Workflow That Invokes the Approvals Manager Business Service*.
- **Business Service.** A step in a process that makes an automated call to the FINS Approval Item Service. A workflow process definition can have one or more business service steps. See *Configuring the Business Service Step for a Workflow That Invokes the Approvals Manager Business Service*.

- **End.** A step in a process that specifies when a process instance is finished.

In order for your workflow to execute correctly, the Start and Business Service steps must meet the minimum requirements described in the referenced sections. For more information about workflows, see *Siebel Business Process Framework: Workflow Guide* .

An example of a workflow that invokes the Approvals Manager business service is the ISS Approval (Order) workflow, shown in the following figure.



Configuring the Start Step for a Workflow That Invokes the Approvals Manager Business Service

The following table details some of the start step parameters for the workflow process.

Field	Comments	Example
Event	The specific event that happens to the object. The set of available events is different for different object types.	Use the WriteRecord business component event if you want to trigger the approval process after the record is written to the database. Use the WriteRecordNew business component event if you want to trigger the approval process after a new record is written to the database.
Event Object	The name of the application, business component, or applet to which the event occurs.	Contact
Event Object Type	The type of object to which the event occurs. This can be an application, business component, or applet.	BusComp
Name	The name of the Next step branch.	

Field	Comments	Example
	The name of the branch must be unique or you cannot import or export the workflow process.	
Next Step	The name of the step that follows when conditions are met.	Picklist of existing process steps.
Type	The type of branch.	The value can be one of the following: • Condition. This value indicates that a condition is defined for the branch. • Default. This value indicates that if nothing else is satisfied, then this branch is followed. Additionally, if this value is used, any conditions defined for the branch are ignored.

Configuring the Business Service Step for a Workflow That Invokes the Approvals Manager Business Service

The first and second table in this topic detail some of the business service step parameters and input arguments for the workflow process.

Field	Value
Business Service	ISS Approval Business Service
Methods	CreateNewApprovalTasks GetApprovalStatus SetApprovalDecision

Input Argument	Property Name	Comments
Approval Identifier	Object Id	Row Id of the object (for example, a Service Request) that needs approval processing.
Approval Item Name	Approval Item Name	Name of the Approval Item defined in the Administration - Application screen, then the Approval Admin view.
Requesting Buscomp	Requesting Buscomp	Name of the buscomp object (for example, a Service Request) that needs approval processing.

Approving or Declining Approval Stages (End User)

End users approve approval items in the Inbox views. Users can view approval items by login name or position. For more information about setting up approval processing, see *Defining Approval Items and Approval Stages*.

The ISS Post Approval workflows (query on ISS Post Approval Workflow *) are invoked to execute the ISS Approval service after the approver takes action in the Inbox views.

To approve or decline an approval stage

1. Navigate to the Inbox views.
2. Select one of the following views:
    - **My Approvals.** Displays all approval items associated with the user's login name.
    - **My Position Approvals.** Displays all approval items associated the current user's position.
3. To view additional details about an approval item, drill down on the Approval Identifier hyperlink.
4. In that Status field, select Approve or Decline.

    Once you select a status, the application populates the Approval By and Approval Date field and sets the record to read-only.

10 Asset-Based Ordering Methods Reference

Asset-Based Ordering Methods Reference

This chapter is a reference that explains the methods developed for the business services used for the asset-based ordering parts of C/OM. It includes the following topics:

- *Product Manipulation Toolkit Business Service Methods*
- *Order Entry Toolkit Business Service Methods*
- *Account Administration Toolkit Business Service Methods*
- *Complex Product Auto Match Business Service Method*

Product Manipulation Toolkit Business Service Methods

The Product Manipulation Toolkit (PMT) business service is a set of methods that can be linked to implement order processing workflows. These workflows maintain the service profile as orders are provisioned.

The two primary methods in this toolkit are:

- **Delta.** Creates a Quote or Order that defines the changes required to convert the initial state of an Asset into the final state of an Asset.
- **Apply.** Applies changes defined in Quotes and Orders to an Asset, putting the Asset into a new state.

The toolkit also provides a number of methods to support Delta and Apply.

This topic begins with a description of *User Properties Used by PMT Methods*. Then this topic describes all the methods that the PMT business service calls, which are summarized in the following table.

Method	Comment
Delta Method	Generates the actions necessary to change an existing product with components (asset) into a new product with components. The set of actions can be written to a quote or an order.
Apply Method	Applies changes defined by a Sales order line item to a customizable asset.
Trim Method	Eliminates line items from a delta quote or delta order if they do not meet the requirements specified in the input arguments. This action produces a new trimmed quote or order. The method determines which changes in a customizable order item to apply to the service profile stored in Assets.
Explode Method	Creates multiple instances of a product. The number of instances is determined by the value of the field defined by the ExplodeOnField argument. For each new instance, the value of ExplodeOnField is set to 1. An existing instance is considered for explosion only if it meets the conditions specified by ConditionFieldNames and ConditionValues.

Method	Comment
Explode Siebel Object Method	Functions like Explode except that it also loads the SiebelMessage integration object from the Siebel database with a specified business component and synchronizes it back to the database after the explosion.
Find Orders Method	Given the asset integration ID of a root line item, this method finds all instances of order items that have the same asset integration ID. The order header, matching line item, its child items and attributes are returned as part of the output. Other lines item in the same order header with a different integration IDs are not returned.
Logical Delete Method	Converts any item of a product instance that has a Deleted action code to an Update action code and an Inactive status. Logical Delete only works with a product instance of the Order type. In other words, the Integration Object passed in the Siebel Message is based on the Order Entry business object.
Assign New Service IDs Method	Assigns a service point ID, associated with a specified premise, to each item of the input complex object where the service point type matches the service type of the product.
Convert Product Instance Method	Converts a product instance of one type to another; for example, quote to order.
Get Instance Method	Gets a complex product instance from the Product Configurator.
Get Profile Attribute Method	Returns the value of the specified attribute of the user profile.
Is Fully Ex Method	Checks a product instance to determine if an explode operation is required, based upon the value specified by ExplodeOnField. If the field value is greater than 1 for any component of the product instance, the method returns N. Otherwise, the method returns Y.
Is Module Licensed Method	Determines whether or not the specified module is licensed.
Merge Method	Merges the components of one integration object (product instance) under the header of another integration object.
Quote To Revenue Method	Generates revenue line items for each line item in a quote that matches the criteria specified by the input conditions. The line items are associated with the opportunity from which the quote was created.
Reconfigure Product Instance Method	Displays the asset that was passed to the Product Configurator as input, in the Configurator UI.
Reset Method	Clears out all cached product instances.
Retrieve Next Object From List Method	Given a hierarchical integration object with multiple root components at the second level, this method returns an integration object that contains the header, one root component, its children and their attributes.
Set Action Method	Sets the Action Code field of all items in the hierarchy of a given product instance to the specified value.
Set Exception Error Message Method	Called from the workflow to get the localized error message text that is associated with the input error code.

Method	Comment
Set Field Value Method	Sets a specified field to the given values for all items in the product instance that meet an optional condition.
Set Multiple Field Values Method	Sets specified fields to the given values for all items in the product instance.
Set Output Header Method	Caches the output header that will be used by the Apply and Delta methods.
Set Product Instance Method	Caches a product instance that will be used as an input arguments for Apply and Delta methods.
Set Profile Attribute Method	Assigns values to attributes in a user profile.
Synchronize Method	Synchronizes product instance to the database. Optionally, this method also reprices the instance after it is synchronized by calling Pricing Manager Reprice or RepriceAll. This method calls the EAI Siebel Adapter Execute method to synchronize or upsert.
Update Multi Object List Method	After a root integration component is stripped from the integration object by the Retrieve Next Object From List method, this method returns the resulting integration object.
Update Order Line Item Completed Flag Method	Sets the Order Item Processed Flag of the root order line item to Y, if its status and that of all its child items is Complete, Rejected, or '-'.
Get Cfg Button Click Information Method	Identifies the button the user has clicked in the Complex Product view.
Refresh Business Component Method	Reexecutes all instances of the specific buscomp to get data from the database.
Invoke BC Method	Allows a business component-based method to be invoked from a workflow. Acts as a bridge to pass the business component name and method name, along with the parameters, and returns the value required from the workflow to the specified business component
Iterate Process For Selected Rows Method	Loops through all selected rows in the active business component and initiates the specified workflow process for each row.
Get Selected Row Count Method	Returns the number of rows selected in the active business component (for example, the business component that initiated the workflow).
Get First Selected Row Values Method	Queries the active business component for a given set of field values (specified by the Fields argument) to be assembled and returned in the output property set.
Ungroup Method	Creates multiple instances of a product. The number of instances is determined by the value of the field defined by the ExplodeOnField argument. For each new instance, the value of ExplodeOnField is set to 1. An existing instance is considered for explosion only if it meets the conditions specified by ConditionFieldNames and ConditionValues.

User Properties Used by PMT Methods

The following user properties are used by PMT methods:

- **Alias Action Code.** Used by Delta and Apply to extend the standard set of action codes by creating aliases.

  Syntax:

  ```
  Alias Action Code = "<action code>","<alias action code>","<expr to satisfy on
  Delta>"
  ```

  Example:

  ```
  Name = Alias Action Code 1

  Value = "Update", "Suspend", "[Old Asset Status] = "Active" AND [Asset Status] =
  "Suspended"
  ```

- **Asset Integration Object Name.** Name of the integration object that is based upon the Asset business object.
- **Attribute Integration Component Name.** Name of the integration component that is based on the extended attribute business component. For example, Quote Item XA is a line item's extended attribute. This value must be the same for all three integration objects: asset, quote, and order.
- **Attribute Item Map.** Used by the Convert Product Instance, Delta, and Apply methods to map Asset, Quote, and Order attribute fields. It allows the methods to transform one data type (Asset, Quote, or Order) to another data type (Asset, Quote, or Order).

  Syntax:

  ```
  Name = Src Int Obj Name.Src Int Comp Name:Dest Int Obj Name.Dest Int Comp Name
  Map #

  Value = [Src Field]:[Dst Field]
  ```

  Example:

  ```
  Name = SIS OM Quote.XA:SIS OM Order.XA Map 20

  Value = [Name]:[Name]
  ```

- **Cancel Button Return.** Output value of the Get Cfg Button Click Info method when the Cancel button is clicked in the Complex Product view.
- **Delta Line Item Compare Field.** Used by the Delta method to determine which Asset line item fields are compared to determine if two line items are different.

  Syntax:

  ```
  Delta Line Item Compare Field = [Asset line item Integration Field]:[Quote/Order
  line item Integration object field]
  ```

- **Delta Old Field.** Used by the Delta method to capture the old value of a line item field when it is changed by a Modify Order.

  Syntax:

  ```
  Delta Old Field # = [field name]:[field name to store old value]
  ```

  Example:

  ```
  Name = Delta Old Field 1

  Value = [Status]:[Old Status]
  ```

- **Delta XA Compare Field.** Used by the Delta method to determine which Asset Line Item's attribute fields are compared to determine if two line item's attributes are different.

  ```
  Delta Line Item Compare Field = [Asset line item Integration Field]:[Quote/Order
  line item Integration object field]
  ```

  Example: If an Order line item's Account Id field is mapped to the Asset Line item's Owner Account Id, PMT user property Quote Integration Object Name is set to SISOM Order, and user property Asset Integration Object name is set to SIS OM Asset, the following user property is created:

  ```
  SIS OM Order Line Item:SIS OM Asset Line Map 20 [Account Id]:[Owner Account Id]
  ```

- **Delta XA Old Field.** Used by the Delta method to capture the old value of an XA field when it is changed by a Modify Order.

  Syntax:

  ```
  Delta XA Old Field # = [field name]:[field name to store old value]
  ```

  Example:

  ```
  Name = Delta XA Old Field 1

  Value = [Value]:[Old Value]
  ```

- **Done Button Return.** Output value of the Get Cfg Button Click Info method when the Done button is clicked in the Complex Product view.
- **Header Integration Component Name.** Name of the integration component that is based on header business components. A Quote is a header of a Quote, an Order is a header of an Order, and so on. This value must be the same for all three integration objects: asset, quote, and order.
- **Header Map.** Similar to the Attribute Item Map except that this user property maps header fields.

  Syntax:

  ```
  Name = Src Int Obj Name.Src Int Comp Name:Dest Int Obj Name.Dest Int Comp Name
  Map #

  Value = [Src Field]:[Dst Field]
  ```

- **Line Item Integration Component Name.** Name of the integration component that is based on line item business components. Quote Item is a line item component, Order Item is a line item component, and so on. This value must be the same for all three integration objects: asset, quote, and order.

- **Line Item Map.** Similar to the Attribute Item Map except that this user property maps line item fields.

  Syntax:

  ```
  Name = Src Int Obj Name.Src Int Comp Name:Dest Int Obj Name.Dest Int Comp Name
  Map #

  Value = [Src Field]:[Dst Field]
  ```

  Example: If an Order line item's Account Id field is mapped to the Asset Line item's Owner Account Id, PMT user property Order Integration Object Name is set to SIS OM Order, and user property Asset Integration Object name is set to SIS OM Asset, the following user property is created:

  ```
  Name = SIS OM Order.Line Item:SIS OM Asset.Line Item Map 20

  Value = [Account Id]:[Owner Account Id]
  ```

- **Order Integration Object Name.** Name of the integration object that is based on an Order business object.
- **Quote Integration Object Name.** Name of the integration object that is based on a Quote business object.
- **Workflow Product Configuration View.** Specifies which view the Product Configurator is to use when PMT method Reconfigure Product Instance is invoked.

  > **Note:** The name of the view must be added to both of the following views in the Siebel client: (1) Application Admin screen, then the Views view (2) Application Admin screen, then the Responsibilities view

  Syntax:

  ```
  SIS OM Reconfigure Complex Product View Name: Account
  ```

  SIS OM Complex Product Runtime Instance View - Account

Delta Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It generates the actions necessary to change an existing product with components (asset) into a new product with components. The set of actions can be written to a quote or an order.

Delta compares two complex assets (original and modified) and returns a quote or order. The return contains line items that specify the actions required to change the asset from the original state to the final state.

> **Note:** An update occurs if a field in the product or any of its attributes changes. The list of fields being compared is defined by the Delta Line Item Compare Field user properties. This list of fields is configurable to support customer extensions to the database.

Arguments

Argument	Description
SiebelMessage	[in] Hierarchical property set containing the final Asset (output returned from call to PMT business service method *Reconfigure Product Instance Method*).
SiebelMessage	[out] Hierarchical property set containing a quote or order header, complex line items, and attributes.
SplitQtyChange	[in] Value is N. Controls whether a new line item is created upon an increase in the quantity of an asset component. This argument is optional. By default the component line item is split.

Returns

Property Set containing the complex quote or order.

Remarks

Because Delta is used frequently, the following additional information about the method is useful.

User Properties

The Delta method uses the following user properties:

- Asset Integration Object Name
- Quote Integration Object Name
- Order Integration Object Name
- Delta Line Item Compare Field
- Delta XA Compare Field
- Delta Old Field
- Delta XA Old Field
- Line Item Map
- Attribute Item Map
- Alias Action Code

For descriptions of these user properties, see *User Properties Used by PMT Methods*.

Before Invocation

Before Delta is invoked, the system must call two other methods:

- Set Product Instance

  Saves the original asset's configuration before the Product Configurator is called. For more information, see *Reconfigure Product Instance Method*.
- Set Output Header

Saves the quote or order header that will be the Delta output. If a line item or attribute is associated with the Quote or Order property set, it is stripped from the property set returned by the Delta method. For more information, see *Set Output Header Method*.

Processing

During Delta processing, the method:

- Compares the before and after images to determine the correct action codes for output.
- Passes all fields in the new customizable asset through to the delta quote or delta order. This includes all custom fields.

  Delta compares a user-configurable set of fields. This includes the parent component ID to make sure that changes to the product with components structure are reflected as an update.

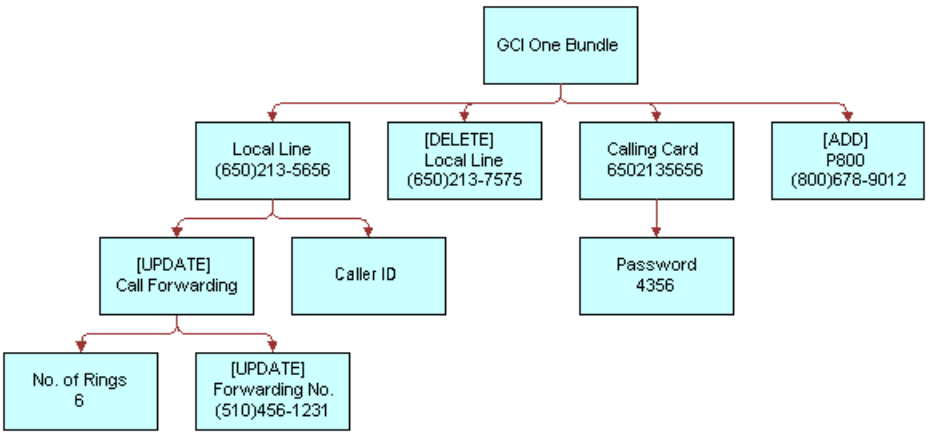
Increasing Quantities of an Asset Component

If the user edits a customizable asset and increases the quantity of an existing component, the result is two line items. The first line item represents the original asset. The second line item adds new copies of that asset. If the original line item is changed, the Delta action is Update or NULL.

Action Field in the Quote and Order Attribute Tables

Delta logic populates an Action field in the quote attribute and order attribute tables. This field allows order provisioning logic to determine which of the attributes of a service product has changed.

For example, a delta quote can be represented as shown in the following figure. In this example, the call forwarding number changed but the number of rings did not.



Action Codes Reset Upon Delta Line Item or Attribute Changes

When a delta-enabled field in a line item changes (because of direct user input or a process such as repricing) or an attribute of a line item changes, the action code is automatically set. This is shown in the following table.

Original Action	New Action
None	Update

Original Action	New Action
Add	Add
Update	Update
Delete	Delete

Note: Make the Action field Read-Only to avoid possible violations of configuration rules that could be caused by changing the action code of a line item.

Alias Action Codes

The Delta method has been extended to support *Alias Action Codes*. Delta replaces one of the standard action codes (Add, Update, Delete, -) with an alias action code if a certain condition is met. For example, an action code of *Update* may be replaced by *Suspend* if the status field changes from *Active* to *Suspended*. Alias action codes are evaluated for components but not attributes. Alias action codes are specified by the Alias Action Code user properties.

Old Value Support

When performing a modify order in Siebel Customer Order Management versions prior to 7.7, you can view the changes made to a product but only the end state, and values prior to the modify are lost. Downstream provisioning systems require both the prior and current values. For example, a change in bandwidth from 56K to 1024K might require a new piece of equipment to be installed at the wire center whereas a change form 2048K to 1024K is simply a downgrade using the existing equipment.

The Delta method has been extended to store the values of fields prior to their being changed. The prior value is the value of the field in the initial property set being considered by Delta.

Service Item Unique Keys (Asset Integration Id)

The Delta and Apply method operations depend upon the unique keys to each service item. Typically, the unique key is an invariable combination of fields in the service item record. Because no combination of user-entered fields is certain to be unique or invariable, the Siebel application provides a hidden Asset Integration Id field that stores a unique identifier for each service item.

The asset integration ID links the service item to the quotes and orders that modify it. On creation of a quote to add a new service item a new asset integration ID is generated from the row ID of the quote line item The quote is converted to an order at which time a new asset integration ID is generated from the row ID of the order line item. This occurs only if the action code of the quote line item is ‘Add’ to enforce uniqueness if multiple orders are created from the same quote.

When the completed order is converted into an asset the asset integration ID is copied from the order line item to asset. When the asset is subsequently modified (Modify or Disconnect) the asset integration ID is copied to the quote and order line items.

Action Types

Each action types is implemented as a soft-coded list of values. This soft coding supports a multilingual user interface and allows for industry specific terminology. The action types supported by the Siebel application are listed in the following table.

Action Type	Comments
Add	None
Update	None
Delete	None
None	No action

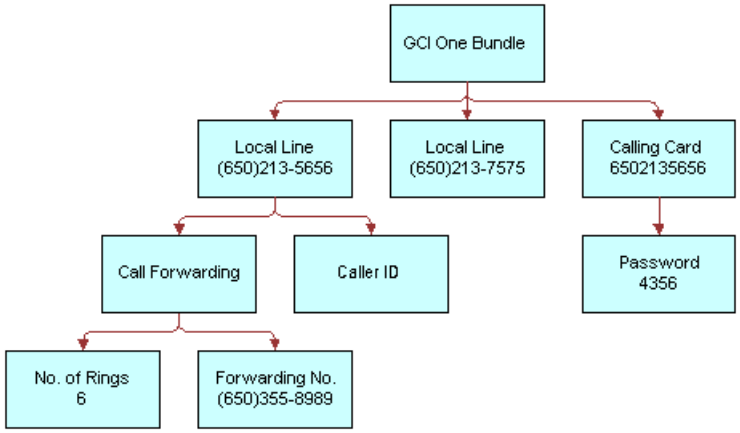
Examples

Review the following examples in the following sections:

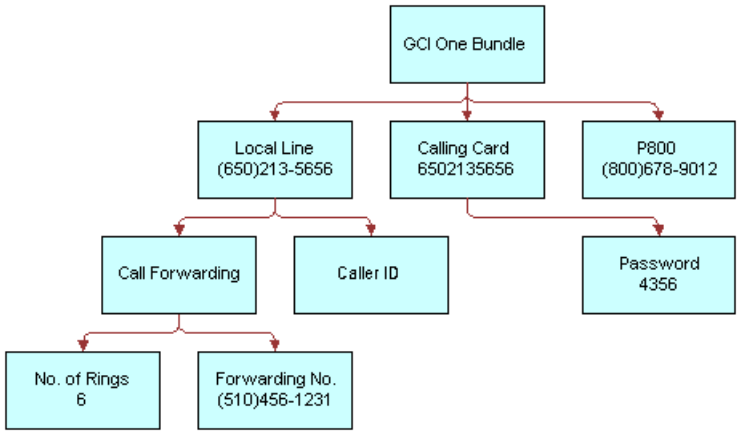
Generating a Delta Quote to Update an Asset

The following example shows how this method generates a delta quote to update an asset.

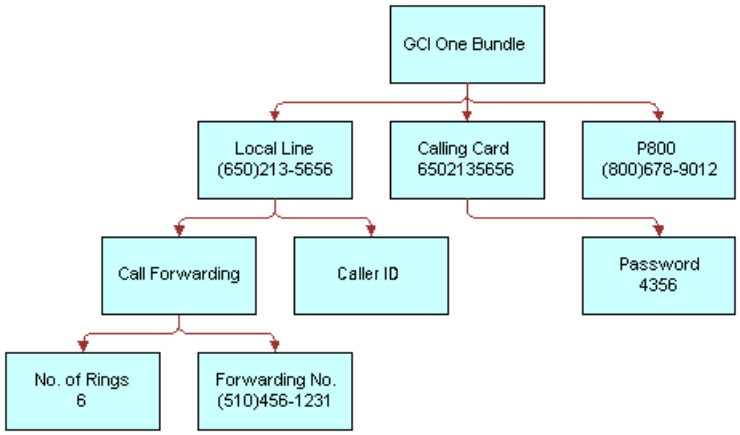
1. A configuration session starts with the GCI One Bundle in the state shown in the diagram that follows.



2. A CSR updates the customizable asset, as in the diagram that follows.



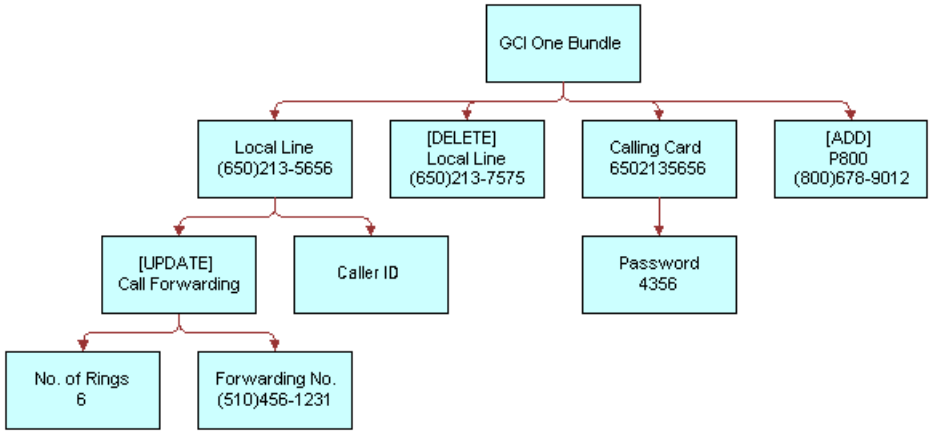
3. The Delta method generates the delta quote shown in the diagram that follows.



Generating a Delta Quote to Add a New Asset

The following example shows how this method generates a delta quote to add a new asset.

1. A configuration session starts with no existing asset. The user configures a new customizable product, as in the diagram that follows.

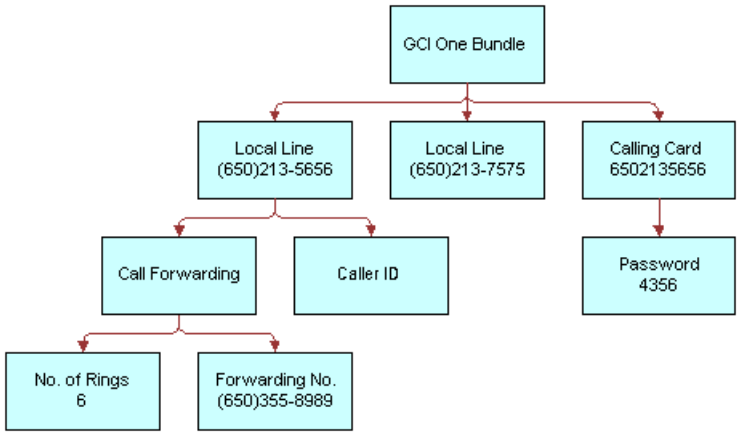


2. The Delta method generates the following delta quote.

Generating a Delta Quote to Disconnect an Asset

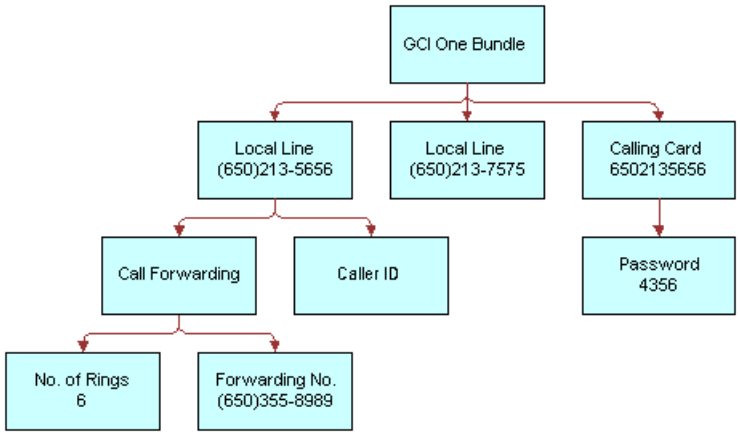
The following example shows how this method generates a delta quote to disconnect an asset.

1. The user selects a customizable asset in the service profile view, as in the diagram that follows.



2. The user clicks Disconnect.

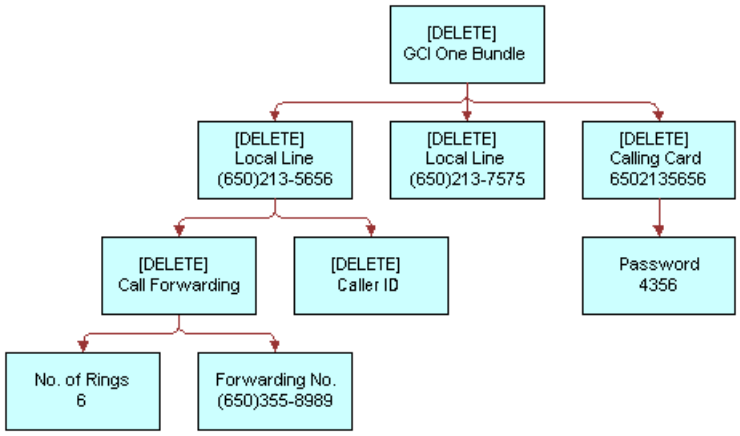
   A workflow runs Delta with the current state of the customizable asset and an empty customizable asset as input arguments. The resultant delta quote is shown in the diagram that follows.



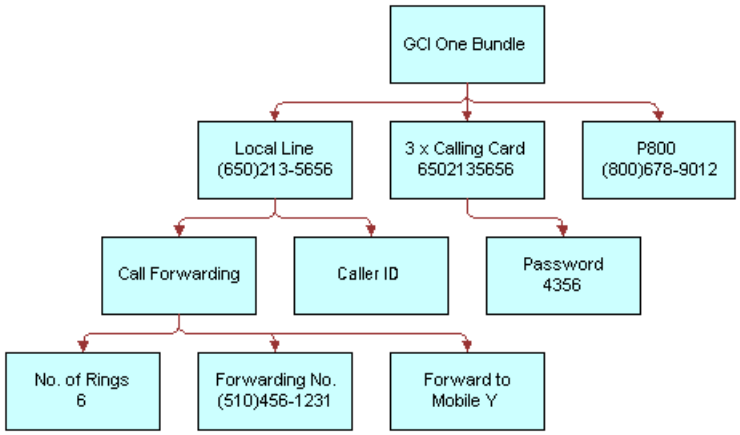
Generating a Delta Property Set to Add More Assets

The following example shows how this method generates a delta property set to add more copies of an asset.

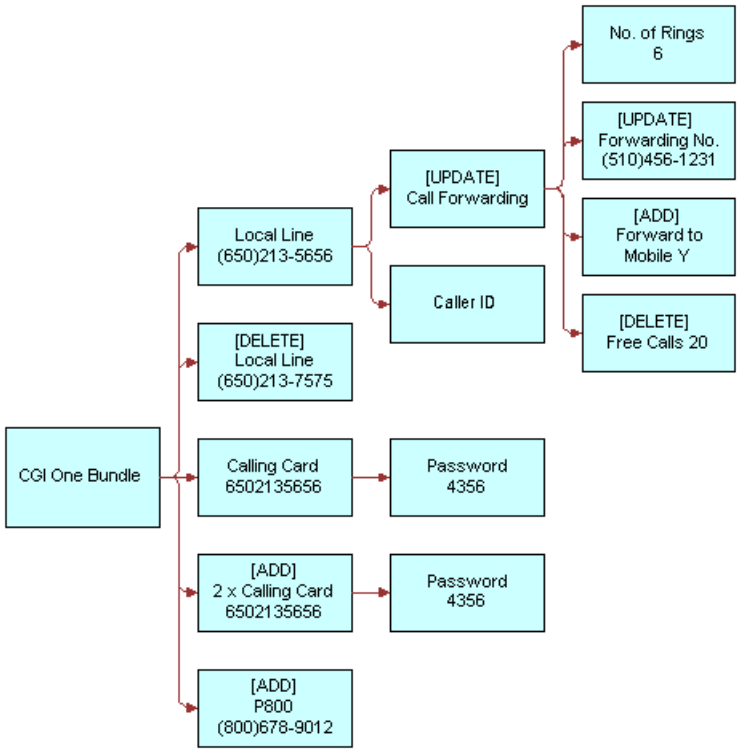
1. The user selects a customizable asset in the service profile view, as in the diagram that follows.



2. The user makes various changes including changing the quantity of Calling Card from one to three, as in the diagram that follows.



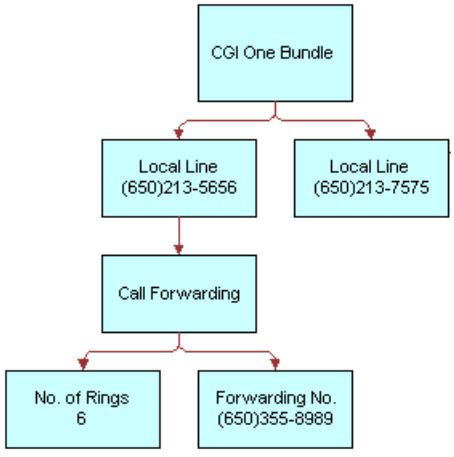
3. Delta generates the delta property set as shown in the diagram that follows. The calling card record is split out into the original, unchanged asset and an action to add the new copies of the original calling card.



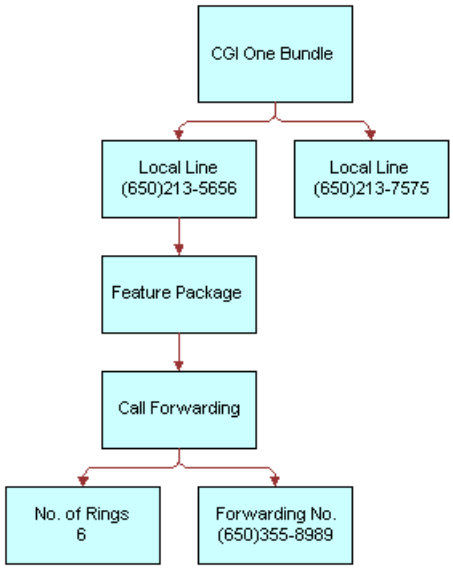
Identifying Changes in Product Structure

The following example shows how this method is used to change a product structure.

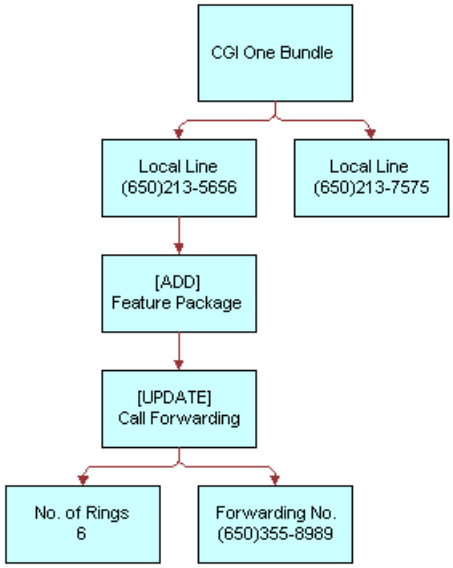
1. The user selects a customizable product in the customer profile view, as in the diagram that follows.



2. Since this asset was created, the customizable product structure has changed to group all features beneath a Feature Package component. When the product is loaded into the Configurator, it is relinked and displayed as shown in the diagram that follows.



3. When the new structure is saved, Delta identifies the new Feature Package component and marks the Call Forwarding feature for update because its parent has changed. This is shown in the diagram that follows.



Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Apply Method*
- *Trim Method*
- *Reconfigure Product Instance Method*
- *Set Output Header Method*

- *Set Product Instance Method*

Apply Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It applies changes defined by a Sales order line item to a customizable asset. This method uses, as a base, an asset that is cached as a result of a call to set the Product Instance and optionally, a header (asset, quote, or order), passed in during the Set Output Header.

Arguments

Argument	Description
OpenOrders	[in] Output result of a call to Business Service Find Orders. (Optional). For more information, see the section about input arguments. **Note:** Either OpenOrders or SiebelMessage is acceptable as input but not both.
SiebelMessage	[in] Contains a single complex Open Order or an Open Quote Line Item. (Optional) **Note:** Either SiebelMessage or OpenOrders is acceptable as input but not both.
SiebelMessage	[out] Output asset image representing a future configurable asset.
Is Apply Result Empty	[out] Y if all the line items are removed from the result, or if the information supplied to create an asset is insufficient information. **Note:** Either SiebelMessage or Is Apply Result Empty is returned as output but not both.

Returns

An asset PropertySet that represents the original input asset plus the changes defined in the input quote or order line item.

Remarks

The following sections about the Apply method are useful.

Input Arguments

To meet its requirements as a general-purpose tool for processing throughout the Asset-Quote-Order life cycle, the Apply method can accept a variety of arguments as input. All input parameters are optional to a varying degree, and the combination of parameters will be determined by the data present and the desired operation.

Apply handles four possible input parameters:

- OpenOrders [input] PropertySet representing a series of Open Orders

OpenOrders can be passed as one of two arguments directly in the Apply method invocation. When a single OpenOrder is to be processed, this argument can be supplied through a standard SiebelMessage PropertySet, obtained through a call to a standard Siebel Adapter. It can be either an Order or a Quote subtype (Quote only on Modify Quote Workflows).

When more than one Open Order is involved in creating the Output Asset, OpenOrders is supplied by a multiple hierarchy OpenOrders type, obtained by invoking the Find Orders Business Service method. Apply checks for the presence of OpenOrders first, and only looks for the single-order SiebelMessage if OpenOrders is not supplied. If both are supplied, only OpenOrders is processed. If neither is supplied and Input Asset is supplied, the Apply method passes the Input Asset PropertySet back as the Output Asset PropertySet.

- SiebelMessage [input] PropertySet

  This input represents a single Open Order. See the previous description.

- Asset [input] PropertySet

  This argument is passed through the Set Product Instance method invocation before Apply is invoked. The Input Asset PropertySet is the base Asset upon which all changes from Open Orders are applied. If no Assets related to the Open Orders are being applied, the call to Set Product Instance is skipped.

- Header[output] PropertySet

  This argument is passed through the method invocations before Apply is invoked. Ordinarily, the Output Header normally is not supplied. However, if it is supplied, it is passed into the Business Service by a separate invocation of Set Output Header immediately before Apply is invoked.

  Under most operating conditions, Apply determines the contents of the Output Header from the Input Asset or the Input Orders. However, when the Output Header is supplied, it is passed into the Business Service by a separate invocation of SetOutputHeader immediately before Apply is invoked. The Output Header can be a SiebelMessage PropertySet of type Asset, Order or Quote. It can be either an empty header without subordinate data or a fully formed hierarchy with associated child item data. When child item data is carried with the Output Header, the child item data is removed.

  Generally, the Output Header gives the Apply method specific data to create an update Output Header for later synchronization by a Siebel Adapter. Use it only if the Output Header that results from Input Asset or the Input Open Order processing is insufficient for resynchronization.

It is also possible (and occasionally valid) to invoke Apply without passing any arguments at all. If no input is specified at all, Apply returns a value of Y in the Is Apply Result Empty Process Property. This result is also returned when the resulting Asset contains only a header, but no items.

Creating a hybrid asset order

Apply creates a hybrid asset-order to simulate the future configuration of a complex product. Taking an asset representing a complex product as input, Apply overlays all unprocessed items and attributes of that product from all its open orders onto the asset. Because the asset's items and attributes are already provisioned, their action codes will carry the internationalized equivalent of the *(blank) value.

Service Item Unique Keys

The Apply and Delta method operations depend upon the unique keys to each service item. For more information, see the description of *Delta Method*.

Apply assumes that the asset used as a base on which to apply open orders was set using Set Product Instance. If no asset is supplied, either the first Open Order or the single (SiebelMessage) Open Quote or Order will be used as the basis for creating a new complex asset. If neither asset nor Open Order is supplied, the method returns an Empty result.

Exception Handling

Apply handles all service quote or sales order actions even if they include possible conflicts. For example, if a service quote line item instructs the method to modify a service item that is already disconnected, Apply logic ignores the service quote line item. The exception conditions handled by Apply are listed in the following steps.

Apply is executed in two steps:

1. SetProductInstance (Asset PropSet)

   This action initializes internal structures and stores the passed PropertySets that are the result of an earlier invocation of Siebel EAI Adapters. Because a business service is limited to a single hierarchy for each invocation, the PMT business service is invoked twice to pass both PropertySets.

   > **Note:** The Asset PropertySet is assumed to be a single hierarchy representing a single complex item, keyed by the integration ID for the root of the complex item.

2. Apply (OpenOrders PropSet)

   This action does the following:

   - Retrieves the Asset PropertySet from its internal storage (established by calling Set Product Instance) and instantiates the output complex object from it.
   - Instantiates a complex object from the OpenOrders PropertySet input parameter.
   - Iterates through the OpenOrder PropertySet, applying each item in turn, repeating for each open order in ascending chronological sequence.
   - Whenever the hierarchical structure is altered, Apply fixes the output hierarchy to reflect the OpenOrder.
   - Returns the output property set.

   > **Note:** The OpenOrders PropertySet is assumed to be one of a Null hierarchy, a single hierarchy representing one complex item, or a container of iterations of a complex item, each representing a change over time. The integration ID for the root of the complex item is the key for the item.

The Apply method handles the exception conditions listed in the table that follows.

Exception	Action	Reason
Instruction to add an item that already exists.	Ignores the add instruction. Attributes and the price are not updates.	The instruction is outdated. Therefore, the attributes are unreliable.
Instruction to update an item that no longer exists.	Ignores the update instruction.	The instruction is outdated. It cannot be performed.
Instruction to delete an item that no longer exists.	Ignores the delete instruction.	The action has already occurred.

Exception	Action	Reason
Instruction to do nothing to an item that does not exist.	No action.	A sequencing problem may have occurred.

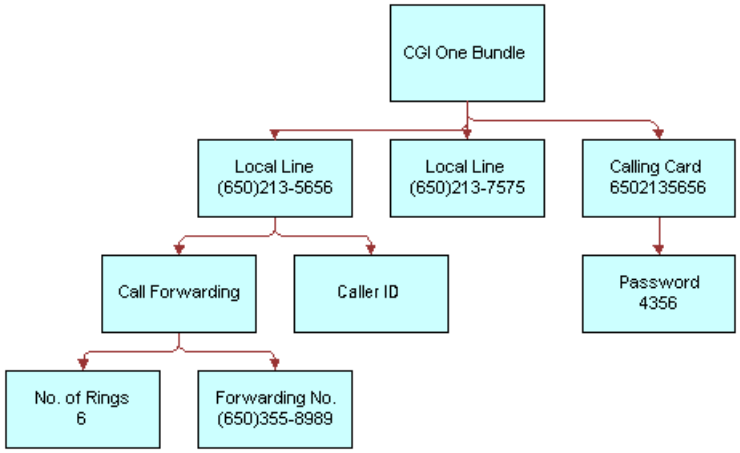
Examples

Review the following Apply method examples.

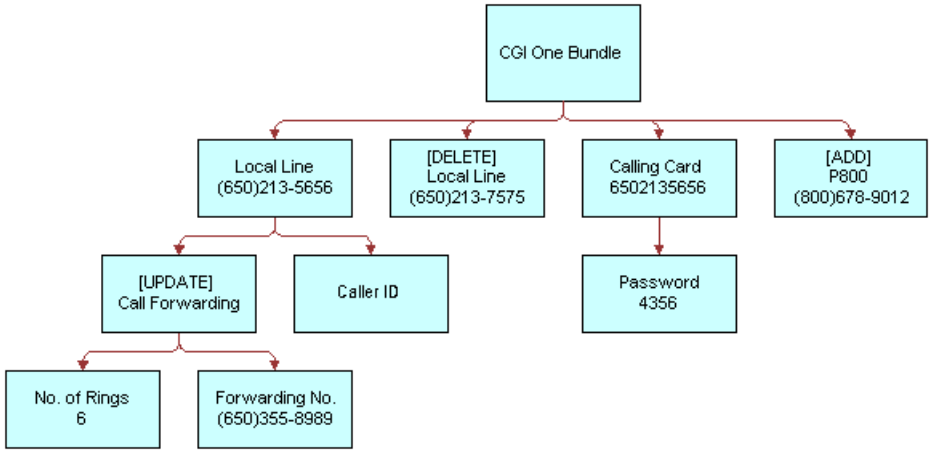
Add, Update, Delete a Complex Order

The following example shows how this method applies add, update, and delete instructions on an order to an existing asset.

1. Start with a customizable asset, as in the diagram that follows.

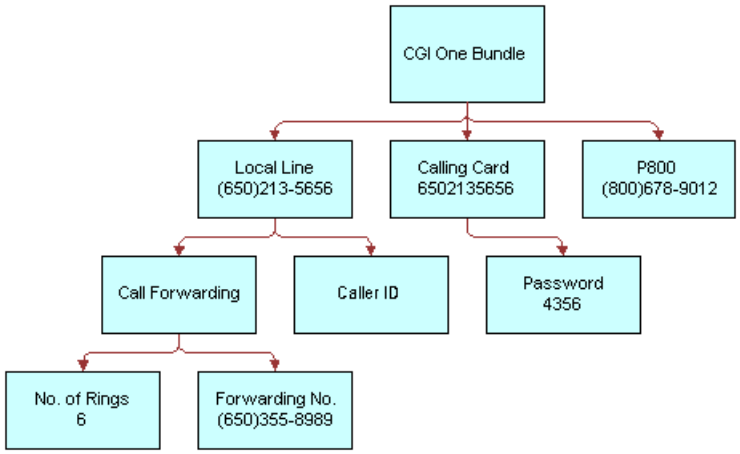


2. Apply a delta order, as in the diagram that follows.



   For more information, see *Delta Method*.

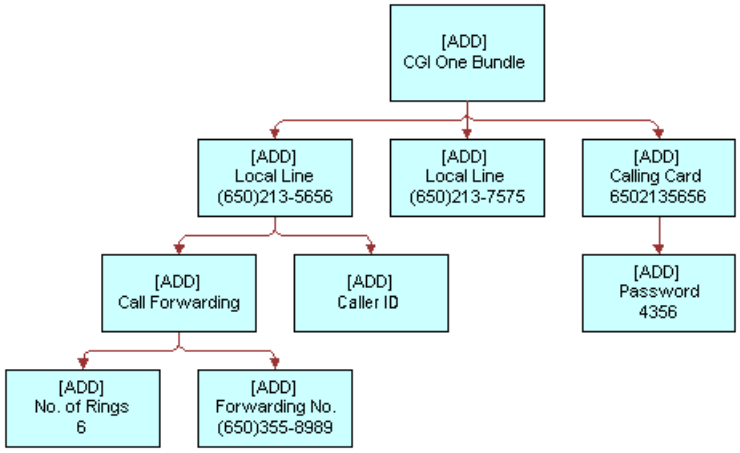
3. A new customizable asset is created, as in the diagram that follows.



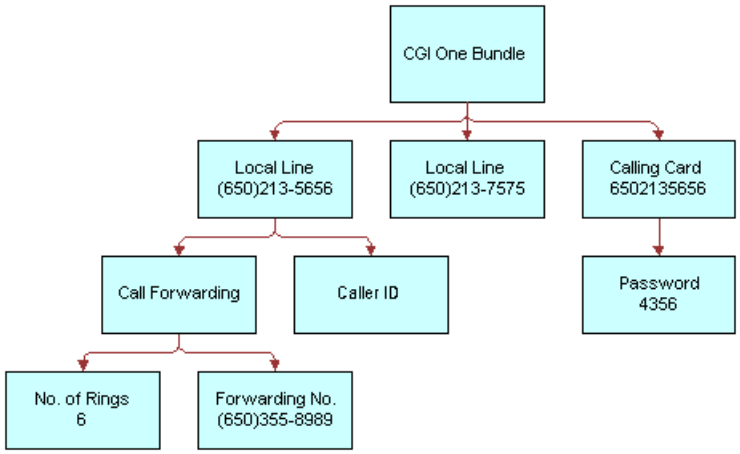
Process a new installation

The following example shows how this method is used to process a new installation.

1. Start with no asset.
2. Apply a new installation.



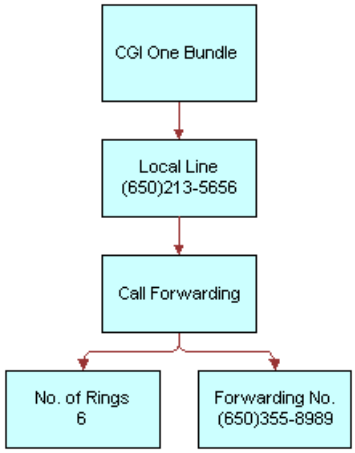
3. A new customizable asset is created, as in the diagram that follows.



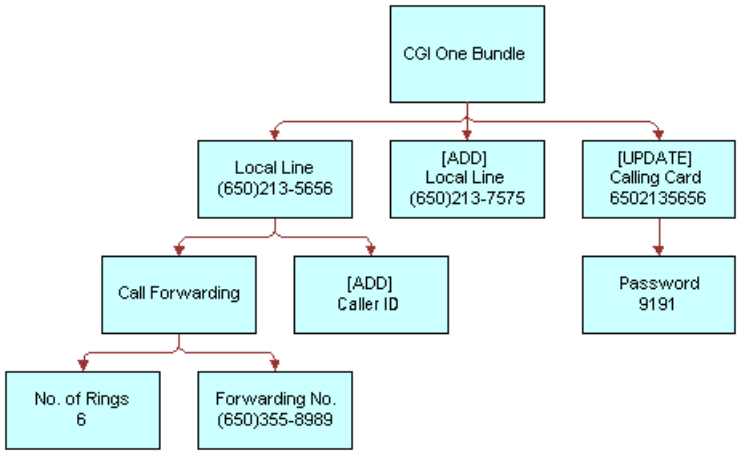
Ignores Instructions to Process Absent Items

The following example shows how this method is used to process a delta quote that includes an update to an absent item.

1. Start with a customizable asset from an external profile management system, as in the diagram that follows.

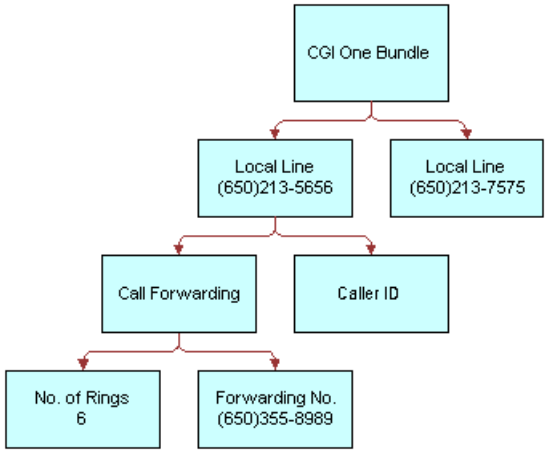


2. Apply a delta quote that was generated a week before, as in the diagram that follows.



> **Note:** The calling card referred to in the delta quote was removed from the profile after the quote was created. The [UPDATE] Calling Card branch is ignored.

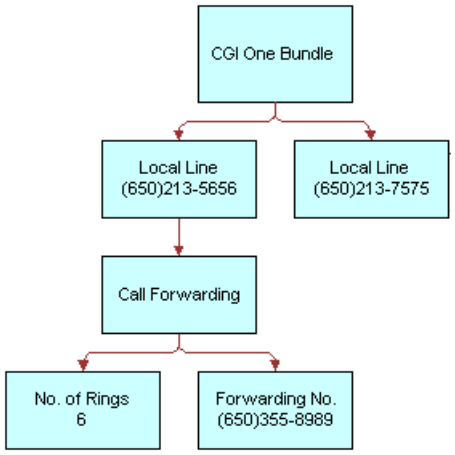
3. The Apply method ignores updates to the service item that no longer exists, but successfully executes the remaining changes. This is shown in the diagram that follows.



Ignores Instructions to Add an Already Existing Item

The following example shows how this method is used to process a delta quote that contains an invalid add instruction.

1. Start with a customizable asset from an external profile management system, as in the diagram that follows.



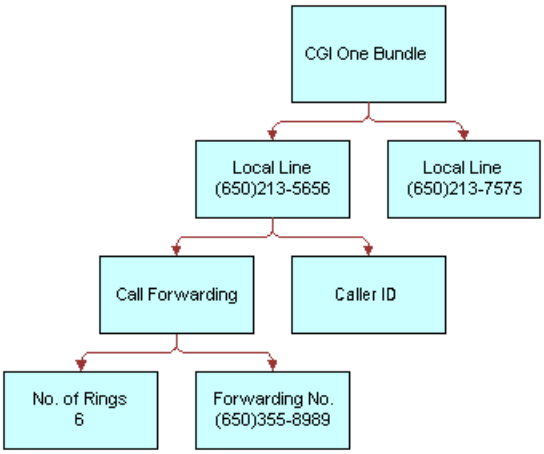
2. Apply a delta quote that was generated a week before.

   > **Note:** The second local line, (650) 213-7575, already exists in the service profile. It was provisioned by an external system user.

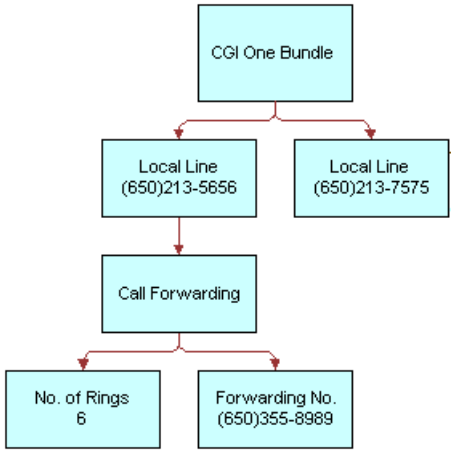
3. Apply ignores add commands where the service item already exists and successfully executes the remaining changes.

Process Instructions to Update the Parent of a Component

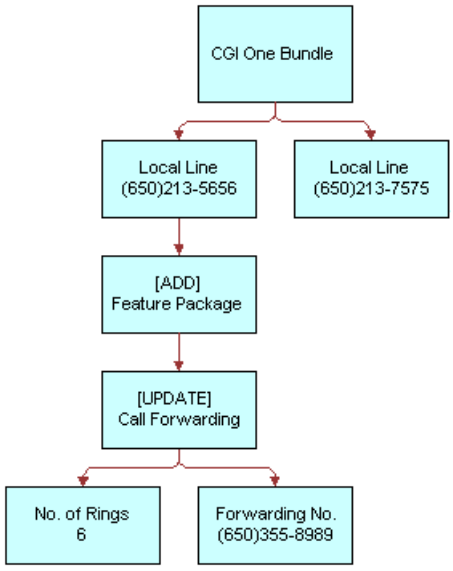
The following example shows how this method is used to process a delta quote that updates the parent component.



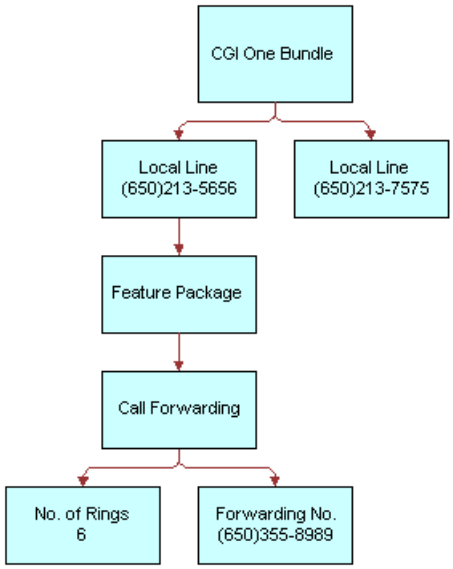
1. Start with a customizable asset in the old product format, as in the diagram that follows.



2. Apply a delta order that updates the parent component of the Call Forwarding feature, as in the diagram that follows.



3. The Apply method adds the Feature Package product beneath the local line and reattaches the existing Call Forwarding feature to the Feature Package, as in the diagram that follows.



Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Delta Method*
- *Trim Method*
- *Explode Method*
- *Set Product Instance Method*

Trim Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It eliminates line items from a delta quote or delta order based on a soft coded rule or Keep Specification. This method is used, in the Order to Asset workflow, to identify changes in an order item that are ready to apply to the service profile stored in Assets.

For a line item to be kept in the product instance hierarchy, KeepSpec must be TRUE for that line item. All children of the line item will also be removed if the parent is removed.

Arguments

Argument	Description
KeepSpec	[in] A Boolean expression based on fields in the current line item. If the line item is to be retained, KeepSpec must return True. (Required)
Object Id	[in] Row Id of the root line item that is used to load the hierarchy if a SiebelMessage is not passed in. (Optional)

Argument	Description
Input Object Type	[in] Type of object to which Object Id relates. Must be specified is Object Id is specified. (Optional)
SiebelMessage	[in] Hierarchy to be used if an Object Id is not supplied. (Optional)
SiebelMessage	[out] Resulting product instance.
Is Trim Result Empty	[out] Y or N value. Y if all line items are removed in the result. Otherwise, N.

Returns

Removes selected line items from the product instance.

Remarks

If the KeepSpec input is TRUE for a line item, it is kept in the product instance hierarchy. If not, it is eliminated. All children of the line item are removed if the parent is removed.

When Trim is called, the method starts at the highest item in the product hierarchy and works recursively down through its children. If the KeepSpec evaluates to TRUE for a line item, it is kept in the product instance hierarchy. If not, it and all of its children are eliminated. For example, the KeepSpec for the Order to Asset workflow is:

```
(([Status] = LookupValue('FS_ORDER_STATUS', 'Complete')) OR ([Action Code] =
LookupValue('DELTA_ACTION_CODE', 'Existing'))) AND ([Convert To Asset Flag] = 'Y')

To process items with status other than Complete or Existing, add the status values to the KeepSpec
Input Argument.
```

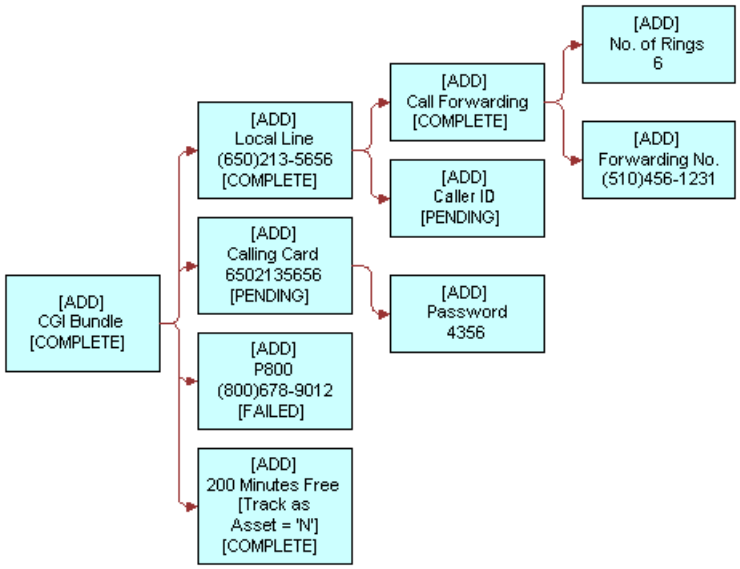
Examples

Review the examples in the following sections:

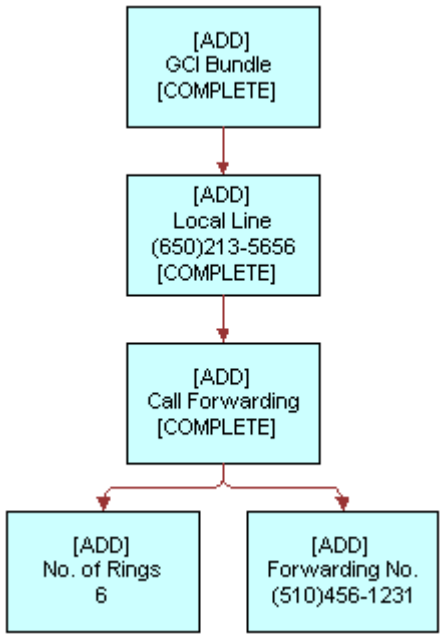
Trimming Pending and Failed Items

The following example shows how this method is used to eliminate pending and failed items.

1. A new installation is partially complete, as in the diagram that follows.



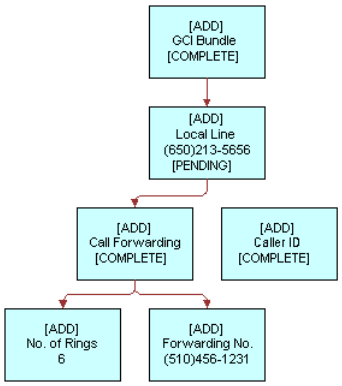
2. Trim eliminates all Pending and Failed items. It also eliminates the 200 Minutes Free product because that product has set Track As Asset to N. This is shown in the diagram that follows.



Trimming Orphaned Items

If an item fails to meet the KeepSpec criteria, this method removes all of its children. The following example shows this situation.

1. A user starts a new installation in which a parent item is Pending and a child item is Complete, as in the diagram that follows.



2. Trim eliminates all Pending or Failed items and their children, Complete or not, as in the diagram that follows.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Delta Method*
- *Apply Method*

Explode Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It creates multiple instances of a product. The number of instances is determined by the value of the field specified by the ExplodeOnField argument. For each new instance, the value of ExplodeOnField is set to 1. An existing instance is considered for explosion only if it meets the conditions specified by ConditionFieldNames and ConditionValues.

Note: Explode works for a quantity set at any level of the product hierarchy.

To exclude fields from being copied from the existing instance to the new instance, add user properties to the SIS OM PMT Business Service. You can use the ExclusionFieldsUserPropertyTag input argument to identify the User Properties series used for this purpose.

Arguments

Argument	Description
RootItemId	[in] Root Item Id. Only the subcomponents of the root line item with a Row Id specified by the RootItemId are considered for Explode. (Optional)
ExplodeOnField	[in] Value of the field specified by ExplodeOnField determines the number of instances created by Explode. For each new instance, the value of the ExplodeOnField is set to 1. (Required)
ConditionFieldNames	[in] Comma separated list of component field names. An existing instance is exploded only if the conditions specified by ConditionFieldNames and ConditionValues are met. (Optional)
ConditionValues	[in] Comma separated list of condition values. Standard Siebel expressions (such as LookupValue) are supported. An existing instance is exploded only if the conditions specified by ConditionFieldNames and ConditionValues are met. (Optional)
ExclusionFieldsUserPropertyTag	[in] Name of the series of user properties that identify fields to exclude when the object instance is copied. The user property name is configurable and specified by ExclusionFieldsUserPropertyTag. (Optional)
SiebelMessage	[in] Product instance to be exploded. (Required)
SiebelMessage	[out] Product instance (integration object) representing the exploded business component. (Required)
Is Exploded	[out] Status flag (Y or N) which indicates whether the SiebelMessage has been exploded or not. (Optional)

Returns

Product set containing multiple copies of the original component.

Remarks

Explode copies any product component whose quantity is greater then (>) 1. It creates multiple copies, each with quantity set to 1. By default, products with the Convert to Asset flag set to N are ignored. This method inputs and outputs a property set containing product changes.

A user configurable list identifies fields that are excluded during the copy. For example, a user would not create multiple copies of a unique identifier such as a telephone number.

The following information about the Explode method is useful.

Excluded Fields

All fields, including prices, are copied as they are into each new instance of the service item, except the following columns that cannot be copied, by default:

- Asset Integration Id

- Conflict Id
- Created
- Sequence Number
- Updated
- Id
- Integration Id
- Quantity
- Service Point Id
- Extended Quantity

User Properties

This method uses the default user properties, listed here, to define a list of integration component fields that are not copied when the parent integration object is exploded.

- Exclude From Explode.SIS OM Order.Line Item 11 to Exclude From Explode.SIS OM Order.Line Item 20
- Exclude From Explode.SIS OM Quote.Line Item 1 to Exclude From Explode.SIS OM Quote.Line Item 10

The general format for all these user properties is:

```
<User Prop Name>.<Integration Object Name>.<Integration Component Name>#
```

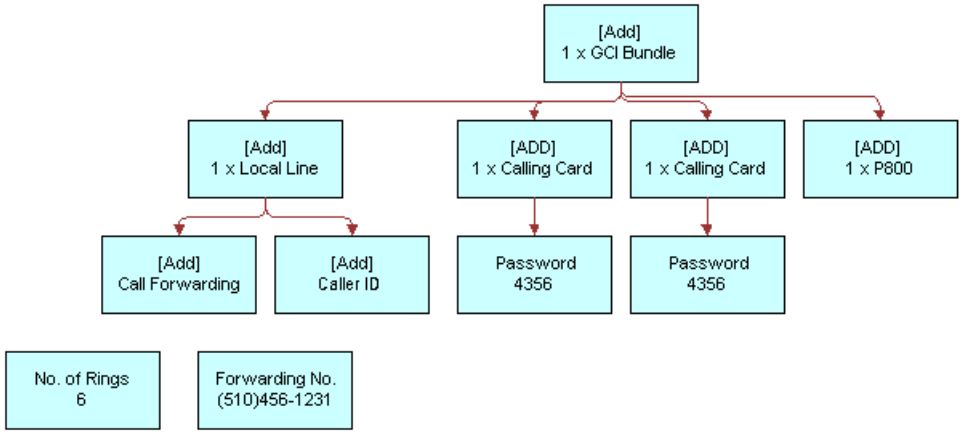
- Examples

Review the following Explode method example.

Copying Components Whose Quantity Exceeds 1

The following example shows this method creates multiple copies of a component.

1. Start with an order to add multiple Calling Cards as part of a GCI One Bundle.
2. Explode copies all components with quantity > 1. The diagram that follows provides an example.



Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Delta Method*
- *Apply Method*
- *Trim Method*
- *Explode Siebel Object Method*
- *Is Fully Ex Method*

Explode Siebel Object Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It functions like *Explode Method* except that it also loads the SiebelMessage integration object from the Siebel database with a specified business component and synchronizes it back to the database after the explosion.

Arguments

Argument	Description
IntObjectName	[in] Name of the integration object representing the business component that will be exploded. (Required)
PrimaryRowId	[in] Siebel object row ID of the business component that will be exploded. (Required)
RootItemId	[in] Root Item Id. Only the subcomponents of the root line item specified by the RootItemId are considered for Explode. (Optional)
ExplodeOnField	[in] Value of the field specified by ExplodeOnField determines the number of instances created by Explode. For each new instance, the value of the ExplodeOnField is set to 1. (Required)
ConditionFieldNames	[in] Comma separated list of integration field names. An existing instance is exploded only if the conditions specified by ConditionFieldNames and ConditionValues are met.
ConditionValues	[in] Comma separated list of condition values. Standard Siebel expressions (such as LookupValue) are supported in each comma separated value. An existing instance is exploded only if the conditions specified by ConditionFieldNames and ConditionValues are met.
ExclusionFieldsUserPropertyTag	[in] Name of the series of user properties that identify fields to exclude when the object instance is copied. The user property name is configurable and specified by ExclusionFieldsUserPropertyTag. (Optional)
SiebelMessage	[out] Product instance (integration object) representing the exploded business component. (Optional)
Is Exploded	[out] Status flag (Y or N) which indicates whether the SiebelMessage has been exploded or not. (Optional)

Argument	Description

User Properties

This method has the following default user properties:

- Exclude From Explode.SIS OM Order.Line Item 11 to Exclude From Explode.SIS OM Order.Line Item 20
- Exclude From Explode.SIS OM Quote.Line Item 1 to Exclude From Explode.SIS OM Quote.Line Item 10

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Explode Method*
- *Is Fully Ex Method*

Find Orders Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. Given the asset integration ID of a root line item, this method finds all instances of order items that have the same integration ID. The order header, matching line item, its child items and attributes are returned as part of the output. Any other line item in the same order header that does not have a matching integration ID will not be returned.

Arguments

Argument	Description
Asset Integration Id	[in] Root asset integration ID that is used to open order items to an asset. (Required)
Search Spec	[in] Additional search specification used to look for open orders. This is a business component search spec that will be applied to the 'Order Entry - Line Item (Asset Based) BC. (Optional)
Sort Order Item By	[in] Comma separated list of field names. Each field name is optionally followed by the string (DESCENDING). For example, Last Name (DESCENDING), First Name. This forces the method to sort the order line item it locates by the given field names. (Optional)
Open Orders	[out] A single hierarchy of type OpenOrders that has child hierarchies for each open order that is found.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Logical Delete Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It converts any item of a product instance that has a Deleted action code to an Update action code and an Inactive status. Logical Delete only works with a product instance of the Order type. In other words, the Integration Object passed in the SiebelMessage is based on the Order Entry business object.

Arguments

Argument	Description
ObjectId	[in] ID of the object to be loaded. If this optional argument is provided, the SiebelMessage argument is ignored. (Optional)
SiebelMessage	[in] Primary argument if there is no Object Id. This must be an Order type input. (Required)
SiebelMessage	[out] Result of the logical delete.

Remarks

This method takes a complex object as input. It goes through the hierarchy of the complex object and changes all Deleted action codes to Update. Then, it sets the status of the associated line items to Inactive.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Get Profile Attribute Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It returns the value of the specified attribute of the user profile.

Arguments

Argument	Description
Profile Attribute Name	[in] Name of the user profile attribute to be retrieved.(Required)
Profile Attribute Value	[out] Value of the profile attribute. This value is NULL if the attribute is not set. (Required)

Returns

Value of the user profile attribute.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Set Profile Attribute Method*.

Get Instance Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It gets a complex product instance from the Product Configurator.

Arguments

Argument	Description
Object Id	[in] Key used to return the preloaded complex asset. The argument Instance Id returned by the Reconfigure Product Instance method is passed here.
Instance Id	[out] Passed to this method as output from Reconfigure Product Instance, this key is used to return a complex asset that was loaded into the Product Configurator when Reconfigure Product Instance was invoked.
SiebelMessage	[out] Complex product instance returned by the Configurator runtime session.

Returns

Complex product instance.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Reconfigure Product Instance Method*.

Convert Product Instance Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It converts a product instance of one type to another; for example, quote to order.

Arguments

Argument	Description
Output Object Type	[in] The input product instance to be converted to this type. (Required)

Argument	Description
Object Id	[in] ID of the object to be converted. If Object Id is specified Input, Input Object Type must also be specified. (Optional)
Input Object Type	[in] Type of the input product instance. (Only required if Object Id is specified)
SiebelMessage	[in] Product instance to be converted. Not required if Object Id and Input Object Type are specified. (Optional)
Generate New Item Integration Id	[in] If the line item's action code is Add (Y or N value), this argument forces the system to generate a new unique ID for the Asset Integration Id field. (Optional) **Note:** The Integration Id and the Service Id are not the same thing. The Integration Id is the internal unique identifier. The Service Id is a free text field that the user may use for telephone numbers, and so on.
Upsert Result	[in] Insert and synchronize the resulting product instance back to the database (Y or N value). (Optional)
SiebelMessage	[out] Product instance to be converted. Not required if the Object Id and Input Object Type are specified.

Returns

Product type change.

Remarks

This method uses the mapping of integration component fields as user properties. The name has the following format:

Source Int Obj Name.Source Int Comp Name:Dest Int Obj Name.Dest Int Comp Name Map #

The user property value format is:

```
[Src FieldName]:[Dest Field Name]
```

Note: Src Field Name must be unique for each group of user property mappings.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Assign New Service IDs Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It assigns a service point ID, associated with a specified premise, to each item of the input complex product for which the service type of the service point matches the service type of the product.

If a free service point is not available for a product component, a service point is not assigned to it. On the other hand, if multiple service point IDs are available for the same service type, the system will pick one of them randomly.

Arguments

Argument	Description
Premise AddressId	[in] Row Id of the address to which services are moving. (Required)
SiebelMessage	[in] Service Point Ids are set for this product instance. (Required)
SiebelMessage	[out] Product instance with the newly assigned service point IDs. (Required)

Returns

New service point IDs.

User Properties

This method uses the following user properties:

- Line Item Integration Object Service Account Id Field Name
- Line Item Integration Object Service Point Id Field Name
- Line Item Integration Object Service Type Field Name
- Service Point BC Address Id Field Name
- Service Point BC Owner Account Id Field Name
- Service Point BC Service Point Id Field Name
- Service Point BC Service Type Field Name
- Service Point Business Component Name
- Service Point Business Object Name

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Is Fully Ex Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It checks a product instance to determine if an explode operation is required, based upon the value specified by ExplodeOnField. If the field value is greater than one for any component of the product instance, the method returns N. Otherwise, the method returns Y.

Arguments

Argument	Description
RootItemId	[in] If supplied, only subcomponents of the root item specified by RootItemId are considered for Explode processing. (Optional)
ExplodeOnField	[in] Field (name) that is checked to determine whether explosion is necessary. (Required)
ConditionFieldNames	[in] Comma separated list of integration component field names. (Optional)
ConditionValues	[in] Comma separated list of values. Standard Siebel expressions (such as LookupValue) are supported in each comma separated value. (Optional)
SiebelMessage	[in] Product instance to be checked for explode processing. (Required)
Result	[out] Y or N flag indicating whether the input SiebelMessage has been exploded. (Required)

Returns

Y or N.

Remarks

Primarily used in the Apply Completed Service Order Line Item to Service Profile workflow, this method double checks to determine if the service order line items created from the Siebel database (earlier in the workflow) have been fully exploded or not. In other words, it determines whether all line items and the subcomponents were previously processed by the Explode method.

Related Information

See the following methods:

- *Explode Method*
- *Explode Siebel Object Method*

Is Module Licensed Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It determines whether the specified module is licensed.

Arguments

Argument	Description
Module Name	[in] Name of the module being checked. (Required)
Result	[out] Y if the module is licensed; otherwise N.

Returns

Y (module licensed) or N (module not licensed).

Related Information

See *ViewCart Method*.

Merge Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It merges the components of one integration object (product instance) under the header of another integration object.

Note: Before this method is called, Set Product Instance must be called to cache the target product instance.

Arguments

Argument	Description
SiebelMessage	[in] Source product instance to be merged. (Required)
SiebelMessage	[out] Merged product instances. (Required)

Returns

A single product instance containing the merged assets.

Remarks

This method receives two property sets as input, each containing a complex object with hierarchical assets, quotes, or order items. It copies all the line items from the source complex object to the target (cached) complex object. The target object's header information (quote or order headers) are retained. The merged complex object is returned in an output argument property set.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Set Product Instance Method*.

Quote To Revenue Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It generates revenue line items for each line item in a quote that matches the criteria specified by the input conditions. The line items are associated with the opportunity from which the quote was created.

Arguments

Argument	Description
SiebelMessage	[in] Contains a product instance hierarchy.
RootItemId	[in] Root item ID.
ConditionFieldNames	[in] Names of fields whose value must equal that specified by ConditionValues. In these cases, the quote line item will be converted to a revenue line item. In the SIS OM Update Revenue workflow, the condition fields are action code, price type and extended amount.
ConditionValues	[in] Values that fields must have to satisfy the condition. In the SIS OM Update Revenue workflow, the action code must be Add or Update, price type must be One-Time or Recurring, and extended amount must be non-zero.
ExcludedFieldsUserPropertyTag	[in] User properties tag identifying fields that are not copied from the quote line item to the revenue line item.

Returns

Revenue line items.

Remarks

The following discussions list user properties associated with this method. They also indicate how the method adds revenue and determines: revenue amount, revenue dates, number of revenue items, frequency of revenue line items, annually recurring charges, quarterly recurring charges, monthly recurring charges, weekly recurring charges, and daily recurring charges.

User Properties

This method used the following user properties:

- **Quote To Revenue.Quote Item.Due Date Field**. Quote Line Item business component field that determines the first date on which revenue will be added. Out of the box, the quote line item is the due date.
- **Quote To Revenue.Quote Item.Amount Field**. Quote Line Item business component field used as the revenue amount. Out of the box this is the extended amount.

- **Quote To Revenue.Quote Item.Item Price Field**. Quote Line Item business component field containing the item price.
- **Quote To Revenue.Quote Item.Price Type Field**. Quote Line Item business component field containing the price type.
- **Quote To Revenue.Quote Item.Unit of Measure Field**. Quote Line Item business component containing the unit of measure.
- **Quote To Revenue.Quote Item.Occurence Field**. Quote Line Item business component field containing the number of revenue occurrences.
- **Quote To Revenue.Quote Item.Extended Quantity Field**. Quote Line Item business component field containing the extended quantity.
- **Quote To Revenue.Quote Item.Description Field**. Quote Line Item business component field containing the description.
- **Quote To Revenue.Quote Item.Product Id Field**. Quote Line Item business component field containing the product ID.
- **Quote To Revenue.Revenue.Quantity Field**. Revenue business component field containing the quantity.
- **Quote To Revenue.Revenue.Quotable Field**. Revenue business component field indicating whether the revenue is quotable.
- **Quote To Revenue.Revenue.Date Field**. Revenue business component field containing the revenue date.
- **Quote To Revenue.Revenue.Price Field**. Revenue business component field containing the product price.
- **Quote To Revenue.Revenue.Revenue Field**. Revenue business component field containing the revenue.
- **Quote To Revenue.Revenue.Description Field**. Revenue business component field containing the description.
- **Quote To Revenue.Revenue.Product Id Field**. Revenue business component field containing the product Id.

Adding Revenue

This method:

- Adds revenue only for quote line items with an Add or Update action code. Quote line items '-', and Deleted action codes are ignored.
- Adds revenue only for quote line items that have an extended amount not equal to zero.

Negative extended amounts are added to revenue.

- Adds revenue only for price types that are one-time and recurring. It is not calculated for usage.
- Adds revenue based on product components (in a quote line item).

Determining Revenue Amount

This method:

- Uses a user property to define the Quote Item business component field that is used for the revenue amount. The default is the Extended Amount field.
- Uses the value of this field as the revenue amount for all periods.

Determining Revenue Dates

This method:

- Uses a user property to define the Quote Item business component field that, in turn, is used to calculate the first revenue date. The default is the Due Date field.

Determining Number of Revenue Items

The forecast number of revenue occurrences for a product is defined in product administration. When a quote line item is created the number of forecast revenue occurrences is copied from the product into the quote line item. There, it can be overridden through the UI or by configuration.

This method:

- Adds revenue for products with one time price types once on the due date of the quote line item, regardless of the number of occurrences defined.
- Adds revenue for products with recurring price types as many times as the number of occurrences.

Determining Frequency of Revenue Line Items

This method:

- Adds revenue as it occurs (weekly, monthly, quarterly or annually) instead of grouping it into monthly totals.
- Adds the first revenue, for any quote line item, on the due date plus one UoM.

The following UoMs that are allowed: Per Year, Per Month, Per Quarter, Per Week, and Per Day.

Determining Annually Recurring Charges

This method:

- Adds revenue on the same day every year, starting on the end date of the first period. For example, if the due date is 7-11-01, the default date of the first billing cycle is 7-11-02 and revenue is added for 7-11-02, 7-11-03 and so on, for as many occurrences as the quote line item specifies.

If the end date of the first period falls on February 29th, the revenue date for nonleap years is February 28th.

Determining Quarterly Recurring Charges

This method:

- Adds revenue on the same day every three months, starting on the date of the first billing cycle, the default value of which is 3 months after the quote line item due date. For example, if the due date is 7-11-01, revenue is added for 10-11-01, 1-11-02, 4-11-02, 7-11-01 and so on, for as many occurrences as product specifies.

If the end date of the first period falls on the 29th, 30th or 31st of a month, the revenue date for months that have fewer days is the last day of the same month.

Determining Monthly Recurring Charges

This method:

- Adds revenue on the same day every month, starting on the date of the first billing cycle which defaults to one month after the quote line item due date. For example, if the due date is 7-11-01, revenue is added for 8-11-01, 9-11-01 and so on, for as many occurrences as the product specifies.

If the due date falls on the 29th, 30th or 31st of a month, the revenue date for months with fewer days is the last day of the same month.

Determining Weekly Recurring Charges

This method:

- Adds revenue every 7 days starting on the date of the first billing cycle which defaults to 7 days after the quote line item due date. For example, if the due date is 7-11-01, revenue is added for 7-18-01, 7-25-01, 8-1-01 and so on, for as many occurrences as the product specifies.

Determining Daily Recurring Charges

The method:

- Adds revenue every day, starting on the date of the first billing cycle which defaults to one day after the quote line item due date. For example, if the due date is 7-11-01, revenue is added for 7-12-01, 7-13-01, 7-14-01 and so on, for as many occurrences as the product specifies.

Reconfigure Product Instance Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It displays, in the Configurator UI, the asset that was passed to the Product Configurator as input.

> **Note:** This method does not return the asset updated by Product Configurator. Instead an event occurs for the primary business component when the Done button is clicked. At that time, you can invoke the Get Instance method to obtain the updated asset from the Product Configurator.

Arguments

Argument	Description
Complex Product	[in] This product instance, based on Asset, is used as input to the Configurator.
Row Id	[in] Row Id of the Asset.
Event Name	[in] Name of the event that is triggered when the user clicks the Done button.
Primary Business Component Name	[in] Name of the primary business component of the business object associated with the workflow that calls this method. This business component receives the event specified by Event Name.

Argument	Description
Pricing Business object	[in] Name of the business object to be used for pricing.
Price List Id	[in] ID of the price list to be used.
Currency Code	[in] Currency code.
Exchange Date	[in] Date of the exchange.
Instance Id	[out] Returned key. This output can be passed (as input) to the Get Instance method to return a complex asset, loaded into the Product Configurator.

Returns

Product Configurator display of the reconfigured complex asset.

User Properties

This method applies the user properties listed in the following steps.

Note: This view must use the same business object as the workflow that invokes the Reconfigure Product Instance method.

- Asset Integration Object Name:

  Name of Integration Object based on Asset business components.

- Complex Product Runtime View Name

  Name of view for Product Configurator UI.

Getting an Updated Asset

This method does not return the Asset updated by the Product Configurator. Instead, an event occurs for the primary business components, passed as parameters to this method, when the Product Configurator's Done button is clicked. At that time, the system can call PMT business service method Get Complex Asset to obtain the updated Asset from the Product Configurator.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Get Instance Method*.

Reset Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It clears out all cached product instances.

Arguments

None

Returns

There are no cached products.

Remarks

This method has no input or output arguments.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Set Product Instance Method*
- *Set Output Header Method*

Retrieve Next Object From List Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. Given a hierarchical integration object with multiple root components at the second level (for example, Asset), this method returns an integration object that contains the header, one root component, its children and their attributes.

Arguments

Argument	Description
SiebelMessage	[in] Integration object to retrieve the root component from. (Required)
Integration Id	[out] Integration Id of the retrieved root integration component. (Optional)
Object Id	[out] Row Id of the retrieved root integration component. (Optional)
Remaining Number of Objects	[out] Number of root integration components left in the input integration object. (Required)
SiebelMessage	[out] New instance of the integration object containing the header and first root component (including its children and attributes) of the object retrieved. (Required)

Remarks

This method can be called multiple times with the same input argument, each time it returns the next root component. And, it is used in conjunction with Update Multi Object List to form a loop control mechanism.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Update Multi Object List Method*.

Set Action Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It sets the Action Code field of all items in the hierarchy of a given product instance to the specified value.

Arguments

Argument	Description
Action Code	[in] Set the action codes of all line items in the hierarchy SiebelMessage to this value. (Required)
SiebelMessage	[in] Product instance whose action code will be updated. (Required)
SiebelMessage	[out] Updated product instance.

Returns

Newly set action codes.

Remarks

This method takes a property set containing a complex item as input along with an action code parameter. It goes through the complex item and sets the action code to the value of the action code argument.

Related Information

See the following methods:

- *Set Field Value Method*
- *Set Multiple Field Values Method*

Set Exception Error Message Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It is called from the workflow to get the localized error message text that is associated with the input error code.

Arguments

Argument	Description
Error Code	[in] Error code defined in the repository. (Required)
Error Message	[out] Localized error message text. (Required)

Dependencies

Strings corresponding to the supplied Error Code must be defined in the Siebel Database. The seven predefined error messages are defined in the Siebel repository with the message key prefixed with IDS_SISOM_ERR_MOVEWF.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Set Field Value Method

It is used optionally to configure conditions so that updates are only run on the subset of items in the hierarchy that satisfy the conditions.

Arguments

Argument	Description
Field Name	[in] Name of the field to be changed. (Required)
SiebelMessage	[in] Product instance. (Required)
Value	[in] Literal. (Required)
ConditionFieldNames	[in] Comma separated list of integration component field names. (Optional)
ConditionValues	[in] Comma separated list of values. Standard Siebel expressions (such as LookupValue) are supported. (Optional)
Generate new Id	[In] Y or N flag indicating whether to generate a new Row Id for each item.
SiebelMessage	[out] Updated product instance. (Required)

Returns

New field values.

Remarks

As input, this method receives one property set containing a complex object and two strings representing a field name and field value. The method goes through the line items hierarchy of the complex object wrapped by the property set, and for each item that satisfies the optional conditions, locates the named field of each line item, and sets it to the value provided.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Set Action Method*
- *Set Multiple Field Values Method*

Set Multiple Field Values Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*.

It sets specified fields to the given values for all items in the product instance.

Arguments

Argument	Description
Field Names	[in] Comma separated list of names of fields whose values are to be set. (Required)
Values	[in] Comma separated list of values to which the fields are set. (Required)
SiebelMessage	[in] Product instance hierarchy whose field values are to be set. (Required)
ConditionFieldNames	[in] Comma separated list of integration component field names. (Optional)
ConditionValues	[in] Comma separated list of values. Standard Siebel expressions (such as LookupValue) are supported. (Optional)
SiebelMessage	[out] Updated product instance. (Required)

Returns

Product instance with updated field values.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Set Action Method*
- *Set Field Value Method*

Set Output Header Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It caches the output header that will be used by the Delta method.

Arguments

Argument	Description
SiebelMessage	[in] Product instance containing the header to be used for the Delta method output.

Returns

Cached output header.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Delta Method*
- *Set Action Method*
- *Set Product Instance Method*

Set Product Instance Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*.

It caches a product instance that will be used as an input arguments for Apply and Delta methods.

Arguments

Argument	Description
SiebelMessage	[in] Product instance being saved. (Required)

Returns

Cached product instance.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Delta Method*
- *Apply Method*
- *Set Output Header Method*

Set Profile Attribute Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*.

It assigns values to attributes in a user profile.

Arguments

Input Argument	Description
Profile Attribute Name	[in] Name of the attribute being set. (Required)
Profile Attribute Value	[in] Value to which the attribute will be set. A NULL value clears the attribute. (Required)

Returns

New attribute values.

Related Information

See *Get Profile Attribute Method*.

Synchronize Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*.

It synchronizes product instance to the database. Optionally, this method also reprices the instance after it is synchronized by calling the Pricing Manager Reprice-RepriceAll. This method calls the EAI Siebel Adapter Upsert method to synchronize.

Arguments

Argument	Description
InMemoryPricing	[in] With this argument set to TRUE, only the instance of the CP (root and subcomponents) in memory is repriced. If it set to FALSE, the CalculatePriceAll method is called, which reprices the entire document. If there is a pricing relationship between the line items, such as a total discount, then you need to set it to FALSE in order to call CalculatePriceAll. Note that this parameter has a severe performance impact if it set to FALSE for large quotes or orders with many line items. Default is FALSE. (Boolean, Optional)
Message Id	[in] Passed through to the EAI Siebel Adapter Upsert method. (Optional)
PrimaryRowId	[in] Row Id of the business component to be synchronized. (Required)
Reprice	[in] Y or N flag indicating whether to reprice or not. (Optional)
RootItemId	[in] If this input is given, only reprice the root line item with a Siebel Object Row Id that corresponds to this RootItemId and any new line items that were created from it after an Explode operation. (Optional)
SiebelMessage	[in] Product instance to be synchronized.
StatusObject	[in] Passed through to EAI Siebel Adapter Upsert method. (Optional)
SiebelMessage	[out] Synchronized product instance.

Returns

Synchronized product instance.

Remarks

This method is used when the object to be synchronized has modified quantity or price fields, requiring a repricing. It is primarily used after Explode.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Explode Method*.

Update Multi Object List Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*.

After a root integration component and its children are stripped from the integration object, this method (in conjunction with Retrieve Next Object From List) returns the resulting integration object.

Arguments

Argument	Description
SiebelMessage	[out] Integration object left behind after the first root component is retrieved. (Required)

Returns

New integration object.

Remarks

This method is used in conjunction with Retrieve Next Object From List to form a loop control mechanism.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Retrieve Next Object From List Method*.

Update Order Line Item Completed Flag Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It sets the Order Item Processed Flag of the root order line item to Y, if its status and the status of all its child items is one of the values passed in Complete Statuses, or if it is '-'.

Note: This method only works with product instance of type Order.

Arguments

Argument	Description
SiebelMessage	[in] Product instance being updated. (Required)
Synchronize	[in] Defaults to N. (Optional)
Complete Statuses	[in] The default values are Complete or Rejected. To set the Processed Flag for other status values, add those Status values to this input argument.
Update Order Items	[out] Comma separated list of row IDs for line items that were updated by this method.

Returns

Order Item Processed Flag set to Y or N.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Get Cfg Button Click Information Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It is used to identify whether a user clicks on the Cancel or Done button from Complex Product View.

Arguments

Input Argument	Description
Business Object Name	(Required) Name of business that Business Component belongs to.
Result	[Out] Either Cancel or Done depending on the button clicked by the user. The actual string value returned is specified by the Cancel Button Return and Done Button Return user properties respectively. (Required).

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Reconfigure Product Instance Method*
- *Get Instance Method*

Refresh Business Component Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It reexecutes all instance of the specific buscomp to get data from the database, and also queries all records again.

Arguments

Argument	Description
Business Object Name	[in] Name of business the buscomp belongs to.
Business Component Name	[in] Name of the buscomp you want to refresh with data from database.
Refresh Result	[out] Either Fail, NoRefresh, or Succeed. Fail means the method could not refresh because of insufficient input argument. NoRefresh means the method did not find any instance of the specified buscomp. Succeed means it refreshed at lease one instance of the specified buscomp. (Optional)

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Invoke BC Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It is a generic method that allows one to invoke a Business Component-based method from Workflow. A Business Service method is invoked from a workflow by default. This method acts as a bridge to allow one to pass in the Business Component name and the method name, along with the parameters and return value required from Workflow to the Business Component specified.

Arguments

Argument	Description
BC Name	[in] A string to specify the name of Business Component on which you want to invoke its method. (Required)
Method Name	[in] A string to specify the name of the method in the specified Business Component that you want to invoke. (Required)
Param 0	[in] A string to pass in the first argument to the method. (Optional)
Param 1	[in] A string to pass in the second argument to the method. (Optional)
Param 2	[in] A string to pass in the third argument to the method. (Optional)
Param 3	[in] A string to pass in the fourth argument to the method. (Optional)
Return Property Name	[out] A string to pass out the output of the method. (Optional)

Iterate Process For Selected Rows Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It loops through all the selected rows in the active business component and invokes the specified workflow for each row. Input arguments to the workflow come from the fixed inputs plus the values of specified field names are transformed into workflow argument names based upon the specified mappings.

Arguments

Input Argument	Description
All Asset List	[in] A property set containing a list of Row IDs.

Input Argument	Description
Fields	[In] Comma separated list of field names in the active business component. (Required)
Fixed Inputs	[In] Comma separated list of name-value pairs (Required). For example: `'Active Document Id='+[&Active Document Id] '+'Price List Id='+[&Price List Id]`
List Type	[in] String
Mappings	[In] Comma separated list of field mappings of the form: `[Bus Comp Field Name]=[Workflow Input Argument]`
Process	[In] Name of the workflow process to be initiated for each row of the active business component. (Required)
Delete Connection	[In] Y or N flag indicating whether to cascade the process to the connections associated with selected nodes in a network scenario. (Optional)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Get Selected Row Count Method*
- *Get First Selected Row Values Method*

Get Selected Row Count Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It returns the number of rows selected in the active business component, that is, the business component that initiated the workflow.

Arguments

Argument	Description
Row Count	[Out] The number of selected rows. (Required)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Iterate Process For Selected Rows Method*
- *Get First Selected Row Values Method*

Get First Selected Row Values Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It queries the active business component for a given set of field values for the first selected row. The fields to be retrieved are specified by the Fields argument. If the Mapping input argument is specified the values of the fields in the query are remapped to different field names in the output property set.

Arguments

Argument	Description
Fields	[In] Comma separated list of field names in the active business component for which values are to be retrieved.
Mappings	[In] Comma separated list of mappings of the form: `[Bus Comp Field Name]=[Property Set Field Name]`
SiebelMessage	[Out] Property set containing the requested values.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Iterate Process For Selected Rows Method*
- *Get Selected Row Count Method*

Ungroup Method

This method is one of the *Product Manipulation Toolkit Business Service Methods*. It is a business component-based version of Explode. It creates multiple instances of a product. The number of instances is determined by the value of the field specified by the Quantity Field argument. For each new instance, the value of the Quantity Field is set to 1. An existing instance is considered for ungrouping only if it meets the conditions specified by the Condition Field Names and Condition Values arguments. The updated business component instances are written to the database.

Arguments

Argument	Description
Line Item BC Name	[In] Name of the line item business component to be ungrouped. (Required)
Extended Attribute BC Name	[In] Name of the XA business component associated with the line item business component. (Required)
Quantity Field	[In] Name of field in the line item business component that is used to determine the number of instances to be created. (Required)
Header Id	[In] Row Id of the header business component instance. (Required)
Header Id Field	[In] Name of the field in the header business component that stores the Row Id. (Required)
Root Item Id	[In] Id of the root item in the line item business component. (Required)
Root Item Id Field	[In] Name of the field in the line item business component that stores the Root Item Id. (Required)
Parent Item Id Field	[In] Name of the field in the line item business component that stores the Parent Item Id. (Required)
Line Number Field	[In] Name of the field in the line item business component that stores the Line Number. (Required)
XA Header Id Field	[In] Name of the field in the XA business component that stores the Header Id. (Required)
XA Parent Root Id Field	[In] Name of the field in the XA business component that stores the Parent Root Id. (Required)
XA Line Item Id Field	[In] Name of the field in the XA business component that stores the Line Item Id. (Required)
Condition Field Names	[In] Comma separated list of field names. An existing instance is ungrouped only if the conditions specified by Condition Field Names and Condition Values are met. (Optional)
Condition Values	[In] Comma separated list of condition values. Standard Siebel expressions (such as LookupValue) are supported. An existing instance is ungrouped only if the conditions specified by Condition Field Names and Condition Values are met. (Optional)
Integer Fields to Split	[In] Comma separated list of fields of type Integer for which the value is to split between the multiple instances. For example, if an instance has a field value of 12 and a quantity of 4, the integer field will have a value of 3 in each of the multiple instances. (Optional)
Number Fields to Split	[In] Comma separated list of fields of type Number for which the value is to split between the multiple instances. (Optional)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and the following methods:

- *Explode Method*
- *Explode Siebel Object Method*

Order Entry Toolkit Business Service Methods

The Order Entry Toolkit (OET) business service is a set of methods that allow Siebel order management processes to be implemented in eSales workflows. The business service includes methods to manipulate the user's account information, validate payment information, and navigate to eSales views. These methods summarized in the following table.

Method	Comment
CreateAccount Method	Creates a new account, associates it with the User and associates specified addresses to that account. The method also sets a specified field in the Quote BC, if it is required.
CreateOrder Method	Converts a quote to an order.
GetBCCount Method	Gets the number of rows and first row ID in a BC for the input Search Spec. If Parent and Child BC names are provided, the search spec is applied to the Parent BC. If no Parent BC is provided, the Search Spec in applied to the one input BC.
GotoView Method	Navigates to the view specified in the input argument.
SelectPrimary Method	Selects a record in the picklist into a field.
SetLIAccounts Method	Rolls down the Service and Billing Account field values from the Quote or Order Header to the line items, if the value is NULL.
SubmitOrder Method	Submits the Pending Order by changing the Order Header and Line Items status to Open. Optionally, sets the Order ID to a defined (user property) Profile Attribute.
ValidatePayment Method	Validates the payment method, verifying that only one payment method at a time is specified for a quote.
ValidateQuote Method	Sets the Invalid Flag for all line items that have a base price of 0 except those that have an Action Code set to Delete.
ViewCart Method	Navigates to the CME Shopping Cart if licensed; otherwise, to the standard Shopping Cart.

CreateAccount Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It creates a new account, associates it to the user and associates specified addresses to that account. The method also sets a specified field in the Quote BC, if it is required.

Arguments

Argument	Description
Account Name	[in] Name of the new account. (Required)
Account Type	[in] Type of new account. (Required)
Address Id 1	[in] ID of an existing address associated with the new account. (Optional)
Address Id 2	[in] ID of an existing address associated with the new account. (Optional) You can add more Address IDs by incrementing the number.
Quote Account Field	[in] Quote business component field to be populated with the Account Id. (Optional)
New Account Id	[out] Row ID of the newly created account.

Returns

Row ID of new account.

User Properties

This method uses the following user properties:

- CreateAccount: Account BC Name

  Name of the business component that is used to create the new account. The default is Account.

- CreateAccount: Account and Address Intersection BC Name

  Name of the business component based on the Account-Address Intersection table that is used to associate addresses to the new account. The default is Com Account Address Intersection.

- CreateAccount: Intersection Account Field Name

  Account foreign key field in the intersection business component. The default is Account Id.

- CreateAccount: Intersection Address Field Name

  Address foreign key in the intersection business component. The default is Address Id.

This method invokes AssociateAccountToUser method in the CUT Account Administration Toolkit Service business service.

Related Information

See the following methods:

- *GetBCCount Method*
- *ValidatePayment Method*

CreateOrder Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It converts a quote to an order.

Arguments

Argument	Description
Quote Id	[in] Quote identifier. (Required)
Return Error Code	[in] Direction to return an error code. (Optional)
Order Id	[out] Order identifier. (Optional)
Error Message	[out] Error message. (Optional)

Returns

A new Order.

Dependencies

This method first invokes the Shopping Service's CreateOrder Method, and then it invokes SubmitOrder.

GetBCCount Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It gets the number of rows and first row ID in a BC for the input Search Spec. If Parent and Child BC names are provided, the search spec is applied to the Parent BC. If no Parent BC is provided, the Search Spec in applied to the one input BC.

Arguments

Argument	Description
BC Name	[in] Name of the business component whose rows will be counted. (Required)
BC SearchSpec	[in] Free text search specification. (Optional)
BusObj Name	[in] The business components belongs to this business object. If a BusObj Name is not specified, the business service business object is used. (Optional)
Parent BC Name	[in] Name of the parent business component to which the search criteria is applied. (Optional)
Field Name	[in] Field name to be used as additional input for the search specification. (Optional)
Field Value	[in] Value to be used as additional input for the search specification. (Optional)
Count	[out] Number of rows. (Optional)
First RowId	[out] First rowId of the rows. (Optional)

Returns

Number of rows and first row Id.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *CreateAccount Method*.

GotoView Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It navigates to the View specified in the input argument.

Arguments

Argument	Description
View	[in] Name of the view to navigate to. (Required)
KeepContext	[in] Set this to TRUE to maintain the context. The destination view must be based on the same business object as to originating view to keep the context.

SelectPrimary Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It selects a record in the picklist into a field.

Arguments

Argument	Description
PickList Field	[in] Name of the picklist field. (Required)
Primary Row Id	[in] Primary rowId (Optional)
Primary ID Field	[in] Name of the field that stores the primary Id. Not required if Primary Row Id is specified. (Optional)
Business Component Name	[in] Name of the business component to which the field belongs. (Optional)
IntersectionTable Field	[in] Name of the field in the intersection table that stores the primary Id. (Optional)
Execute BusComp at Finish	[in] TRUE if Base BC is executed after this operation; otherwise, FALSE. The default is TRUE (case sensitive). (Optional)
ReturnVal	[out] Success or Fail.

Returns

Success or Fail.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

SetLIAccounts Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It rolls down the Service and Billing Account field values from the Quote or Order Header to the line items, if the value is NULL.

Arguments

Input Argument	Description
Parent BC Name	[in] Parent BC name. (Required)

Input Argument	Description
Parent Row Id	[in] Parent row Id. (Required)
Line Item BC Name	[in] Line item BC name. (Required)

Returns

New line item values.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *GetBCCount Method*.

SubmitOrder Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It submits the Pending Order by changing the Order Header and Line Items status to Open. Optionally, it sets the Order Id to a defined (user property) Profile Attribute.

Arguments

Argument	Description
Order Id	[in] Order identifier. (Required)
Parent Fieldmap LHS	[in] LHS value of the field map used by user properties for field names in the Parent business component. (Optional)
Parent Fieldmap RHS	[in] RHS value of the field map used by user properties for field values in the Parent business component. (Optional)
Line Item Fieldmap LHS	[in] LHS value of the field map uses by user properties for field names in the Line Item business component. (Optional)
Line Item Fieldmap RHS	[in] RHS value of the field map uses by user properties for field names in the Line Item business component. (Optional)
Return Error Code	[in] Direction to return an error code. (Optional)
Error Message	[out] Error message. (Optional)

User Properties

The following user properties are associated with this method:

- Order Field|Value FieldMap X—Field map value. See the next user property definition.
- Order Item Field|Value FieldMap X—Field map value.

Replace X with a numbers starting from 1 and increments of 1. The last FieldMap must have a value of End.

- SubmitOrder: Order Header Buscomp—Default = Order Entry - Order.
- SubmitOrder: Line Item Buscomp—Default = Order Entry - Line Items.
- SubmitOrder: Line Item Set Field Condition. Default is Status=FS_ORDER_STATUS Pending.

ValidatePayment Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It validates the payment method, verifying that only one payment method at a time is specified for a quote.

Arguments

Argument	Description
Bill To Account	[in] The account whose payment is being validated. (Required)
Credit Card Number	[in] Credit card number associated with the account. (Required)
Credit Card Type	[in] Type of credit card associated with the account. (Required)
Expiration Month	[in] Expiration month of the credit card. (Required)
Expiration Year	[in] Expiration year of the credit card. (Required)
PO Number	[in] PO number for the account. (Optional)
Return Error Code	[in] Direction to return an error code. (Optional)
Error Message	[out] Error message. (Required)

Returns

Error messages.

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *CreateAccount Method*.

ValidateQuote Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It sets the Invalid Flag for all line items that have a base price of 0 except those that have an Action Code set to Delete.

Arguments

Argument	Description
Quote Id	[in] Quote identifier. (Required)
Return Error Code	[in] Direction to return an error code. (Optional)
Invalid	[out] Indicates an invalid quote. (Optional)
Error Message	[out] Error message. (Optional)
ReturnVal	[out] Indicates that the quote is valid. (Optional)

Dependency

Invokes the Shopping Service's ValidateQuote method.

ViewCart Method

This method is one of the *Order Entry Toolkit Business Service Methods*. It navigates to the CME Shopping Cart if licensed; otherwise, to the standard Shopping Cart.

Arguments

No input or output arguments.

Remarks

The following user properties may be specified for the Shopping Service:

- Module Name

Licensed Module Name. The default is CME eSales.

- Default Shopping Cart View

Name of the view to display if a module is not specified or if the module is specified but not licensed. The default is Current Quote View (eSales).

- Licensed Shopping Cart View

Name of the view to display if the module identified by module name is licensed. The default is CUT Current Quote View (e Sales).

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *ValidatePayment Method*.

Account Administration Toolkit Business Service Methods

The Account Administration Toolkit (AAT) business service is a set of methods that are used to manipulate accounts for eSales workflows.

These methods are summarized in the following table.

Method	Comment
PickAccount Method	Sets an Account to current for a user session and sets its price list as the default price list.
SetPrimary Method	Picks a record from a picklist and puts it into the set of fields specified in the picklist field's pickmap.
AssociateAccountToUser Method	Associates an Account with a User.
EstablishMtoM Method	Establishes an Account to Contact M:M relationship.

PickAccount Method

This method is one of the *Account Administration Toolkit Business Service Methods*. It sets an account to current status for a user session. It also sets the account's price list as the default price list.

Arguments

Argument	Description
Account Id	[in] Row Id of the Account that is set to current status. (Optional)
Account Name	[in] Name of the account. (Optional)

Argument	Description
Master Account Id	[in] Row Id of the ultimate parent of the account. (Optional)
BusComp Name	[in] Name of the account business component. (Optional)

User Properties

- SelectCurrent:Login BO/BS—Default = CUT Account Login|CUT Account Login
- SelectCurrent:BusComp Name to Refresh—Default = Account

SetPrimary Method

This method is one of the *Account Administration Toolkit Business Service Methods*. It picks a record from a picklist and puts it into the set of fields specified in the picklist field's pickmap.

Arguments

Argument	Description
BusComp Name	[in] Name of the business component into which the picklist record will be picked. (Required)
PickList Field	[in] Field name in the business component that has the picklist defined. (Required)
Primary Id	[in] Row Id of the Picklist record that is picked. (Required)
Row Id	[in] Row Id of the business component into which the picklist record is picked. (Required)

User Properties

- SelectPrimary: Source Buscomp—Default = User Profile (eApps)
- SelectPrimary: Source Picklist Field Name—Default =Primary Account Name

AssociateAccountToUser Method

This method is one of the *Account Administration Toolkit Business Service Methods*. It establishes an Account to Contract M:M relationship.

Arguments

Argument	Description
Account Id	[in] Account identifier. (Required)
Contact Id	[in] Contact identifier. (Required)
BusComp Name	[in] Name of the business component from which this method is to be invoked. (Optional)
BusComp ToBe Refreshed	[in] Name of the business component that is refreshed after the operation in order to show updated data. (Optional)

User Properties

- AssociateAccountToUser: Account and Contact Intersection BC—Default = Account Contact.
- AssociateAccountToUser: Default BC—Default = Account
- AssociateAccountToUser: Default BD to Re-Execute—Default = Account Id
- AssociateAccountToUser: Intersection BC Contact Field—Default = Contact Id
- AssociateAccountToUser: Intersection BC Date Field—Default =Start Date
- AssociateAccountToUser: Primary Contact Id Field—Default = Primary Contact Id

EstablishMtoM Method

This method is one of the *Account Administration Toolkit Business Service Methods*. It establishes an M:M relationship between two entities.

Arguments

Argument	Description
MtoMIntXBCName	[in] Intersection business component name. (Required)
MtoMIntXDestFldName	[in] Intersection business component destination field value. (Required)
MtoMIntXDestFldValue	[in] Intersection business component destination field value. (Required)
MtoMIntXSrcFldName	[in] Intersection business component source field name. (Required)
MtoMIntXSrcFldValue	[in] Intersection business component source field value. (Required)

Invoke BC Method

This method is one of the *Account Administration Toolkit Business Service Methods*. It is a generic method that allows one to invoke a Business Component-based method from Workflow. A Business Service method is invoked from a workflow by default. This method acts as a bridge to allow one to pass in the Business Component name and the method name, along with the parameters and return value required from Workflow to the Business Component specified.

Arguments

Argument	Description
BC Name	[in] A string to specify the name of Business Component on which you want to invoke its method. (Required)
Method Name	[in] A string to specify the name of the method in the specified Business Component that you want to invoke. (Required)
Param 0	[in] A string to pass in the first argument to the method (Optional)
Param 1	[in] A string to pass in the second argument to the method (Optional)
Param 2	[in] A string to pass in the third argument to the method (Optional)
Param 3	[in] A string to pass in the fourth argument to the method (Optional)
Return Property Name	[out] A string to pass out the output of the method (Optional)

Note: Preconfigured, you can find the usage of this method in the SIS OM Quote to Order Workflow

Complex Product AutoMatch Business Service Method

The Complex Product AutoMatch Business Service includes one method that is used to match components in a quote, order, or asset with components in the current version of the product model, the Auto Match method.

Auto Match

Auto Match compares the input product instance to the customizable product definition. For each product or class that cannot be located in the definition of the relationship specified by the product instance, Auto Match searches for that product or class in another relationship under the same parent in the product definition hierarchy. If the product or class exists in one or more relationships, the product instance is updated so that the product or class is associated with

the first of those relationships. If the product or class cannot be found in any relationship in the customizable product definition, the product or class is removed from the product instance. The details of each change made by Auto Match are added to the AutoMatchReport and returned as the output of the method with the modified product instance.

Arguments

Argument	Description
SiebelMessage	[in] Product instance to be transformed by Auto Match.
AutoMatchReport	[in] Product instance that has been transformed by Auto Match.

Examples

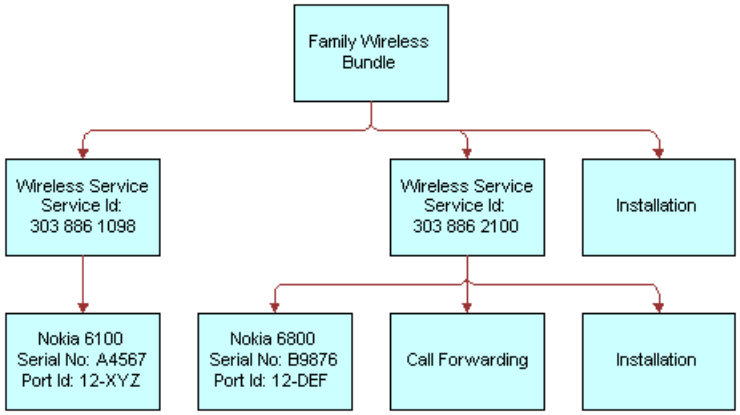
Review the following Auto match method examples.

Service Profile Upgraded from SCE 6.x

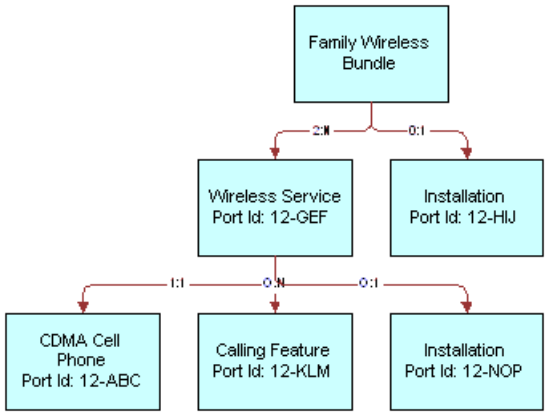
The following example shows how this method is used when a service profile record is upgraded.

Note: In the following examples, a port is an instance of a relationship.

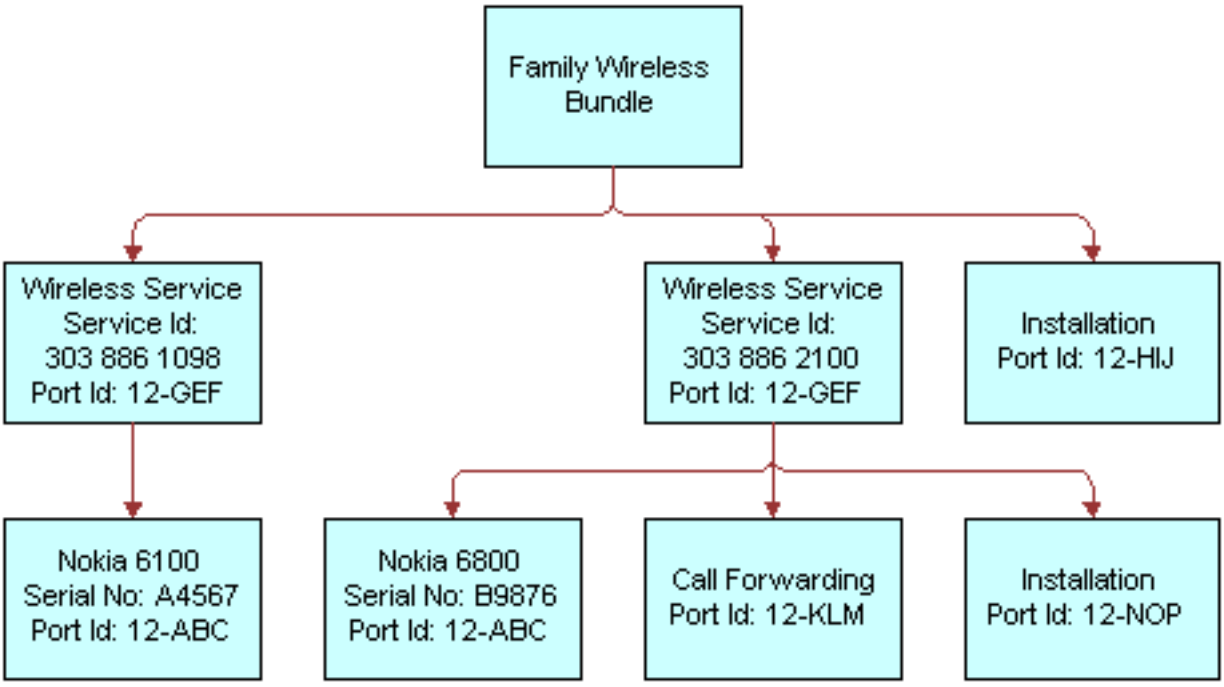
1. The following diagram shows a service profile record that has been upgraded from SCE 6.x:



2. The customizable product definition was reimplemented as shown in the following diagram (rounded boxes represent a class).



3. Auto match assigns the following port IDs without error, as shown in the following diagram.

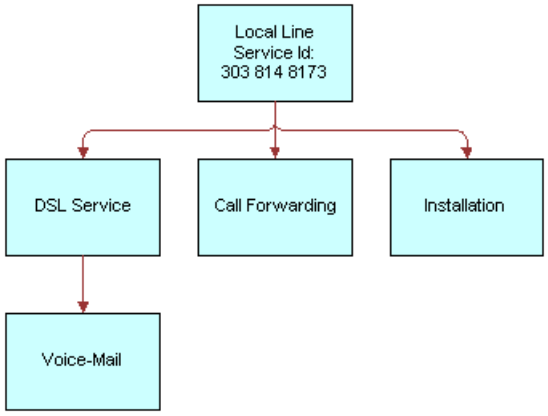


Note: The Installation component port IDs were assigned without ambiguity because each of their parent components only have one port that supports the Installation product. Also, the erroneous port ID originally assigned to the Nokia cell phones was replaced by the correct port ID without generating an error.

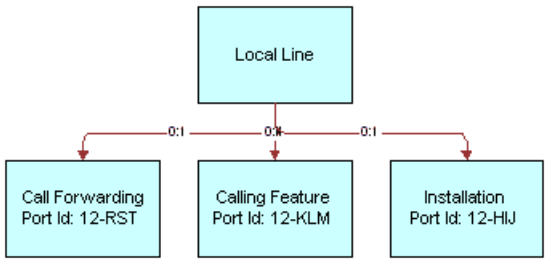
Service Profile Imported from a Legacy System

The following example shows how this method is used when a service profile is imported from a legacy system.

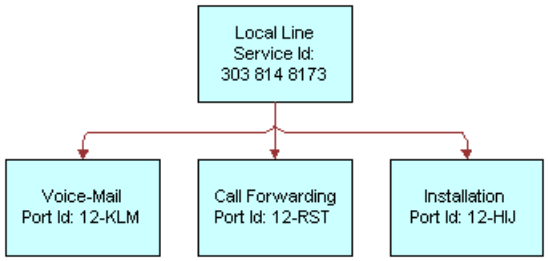
1. The service profile record shown in the following diagram was imported from a legacy system.



2. The customizable product definition is shown in the following diagram.



3. Auto match does its best to assign port IDs, as shown in the following diagram.



Note: The DSL Service component was deleted because it does not exist beneath Local Line. The Voice-Mail component was reparented and associated with the Calling Features class. The Call Forwarding component could have been associated with the product relationship or as a member of the Calling Features class. Hence, it was assigned to the first port found (the product relationship) and a warning message was issued.

11 Projected Asset Cache

Projected Asset Cache

This chapter describes the VORD Projected Asset Cache business service, also referred to as “Projected Asset Cache.” This chapter includes the following topics:

- *About Projected Asset Cache*
- *Projected Asset Cache Business Service Methods*
- *Using the VORD Projected Asset Cache Business Service*

The “VORD” in the name of this business service originally stood for Vertical Order Management, but this business service is now used generally in Siebel Enterprise Applications as well as Siebel Industry Applications.

About Projected Asset Cache

The Projected Asset Cache is a persistent business service that loads all assets, open orders, and quote line items matching a specified search specification into memory using the most efficient SQL queries possible. Projected Asset Cache is used by the Compound Product Validation. It is used by Network Validation to populate the network nodes pick applet. It is also used by Compatibility for cross-product compatibility checking. The Projected Asset Cache can be used by any application that needs a consolidated view of product instances across the quote-to-order-to-asset lifecycle.

Projected Asset Cache automatically converts between the different field names in the Quote, Order, and Asset business components. The cache is always stored and queried using the Asset field names. Open Order and Quote line items are applied to the existing assets to generate a future projected state of the assets for a specified date.

The Projected Asset Cache business service supports multiple concurrent cache instances. When the cache is initialized, a unique identifier is returned. Subsequent queries must specify the identifier of the cache to query.

The following are two key functions to the Projected Asset Cache.

Retrieve Data

To retrieve data, the Projected Asset Cache queries the following business components:

- Quote Item: Quote Item XA
- Order Entry - Line Item: Order Item XA
- Asset Mgmt - Asset: Asset Mgmt - Asset XA

Within these business components, Projected Asset Cache limits the retrieval of data to only those fields and attributes required by the rules checkers. To find this information, the Projected Asset Cache:

- Loads all the Asset records
- Finds all open orders associated with these Asset records
- Loads all quote line items for the current quote associated with the Compound Product.

The Projected Asset Cache then uses a predefined Field Mapping Service business service to translate the field names among business components.

Build the Future Requested State

After retrieving the data, the Projected Asset Cache builds the future requested state of the product instances. It takes into consideration all assets matching the search specification and applies all open orders due to complete prior to the specified date. It then applies the current quote or order to generate the future requested state.

The array of projected assets generated is stored in the business service. It is available for performing validations until it is released or a new initialization of the Projected Asset Cache occurs.

Because the Projected Asset Cache can include any field in the Asset Mgmt - Asset business component, and because it also includes data from the Quote Item and Order Line Item business components, the fields must be mapped across three different business components.

This mapping is done by the Field Mapping Service business service. You add mappings by creating new user properties on the Field Mapping Service business service.

The following table gives an example of one field mapping, which translates the Service Point Serial Number field in the Asset Mgmt - Asset business component to the corresponding field in the Quote Item business component and in the Order Entry - Line Item business component.

Name	Value
Asset Mgmt - Asset:Quote Item Map 2	[Service Point Serial Number]:[Service Point]
Asset Mgmt - Asset:Order Entry - Line Items Map 2	[Service Point Serial Number]:[Service Point]

Note: If you add custom fields to the business components from which the Projected Asset Cache retrieves data, and you want to use these fields either in simple expression rules or custom rules as part of a custom business service, you must add new field mappings.

Projected Asset Cache Business Service Methods

Projected Asset Cache methods are summarized in the following table.

Method	Comment
Initialize Method	Creates a new cache for assets that match the specified search expression.
Query Method	Filters the list of assets in the Projected Asset Cache to those that match the Search Expression.

Method	Comment
Reset Method	Removes a specified cache from the Projected Asset Cache.
Retrieve Method	Supports a combined query of the Initialize and Query methods.

Using the VORD Projected Asset Cache Business Service shows an example of how to structure the input parameters for the VORD Projected Asset Cache in a workflow.

Initialize Method

The Initialize method creates a new cache for assets that match the specified search expression. It applies open orders until the specified future date. It applies line items from the specified quote.

The Initialize method retrieves only those fields and attributes specified by the Field and Attribute input arguments. If a Future Date is provided, open order lines satisfying the search expression that have a due date prior to the Future Date are applied to the associated assets. If a Quote Id is provided, the quote line items of the specified quote that satisfy the search expression are applied to the associated assets.

Arguments

Argument	Description
Search Expression	[in] Search Expression with which to initialize the cache. Only Assets, Quote Line Items, Order Line Items and attributes that satisfy the search expression will be loaded into the cache. (Required)
Field	[in] Property set of fields to be retrieved by the Projected Asset Cache. (Optional)
Attribute	[in] Property set of attributes to be retrieved by the Projected Asset Cache. (Optional)
Future Date	[in] Project assets on this date. Only orders that have a due date less than this future date will be applied. (Optional)
Quote Id	[in] Row ID of the quote for which line items are to be queried and loaded into the Projected Asset Cache. (Optional)
Asset Cache Key	[out] Unique identifier with which to access the Projected Asset Cache. Used by the Query method.

Related Information

See *Query Method* and *Validate Method*.

Query Method

The Query method filters the list of components in the Projected Asset Cache to those that match the Search Expression. It then counts the number of components, sums the values of Aggregate Field, or calculates the minimum, maximum or average for each unique combination of group-by fields, and sorts the result by the Sort Field. This method is analogous to a SQL SELECT statement of the form:

```
SELECT * | COUNT(*) | SUM([Aggregate Field]) | MAX([Aggregate Field]) |
MIN([Aggregate Field]) | AVG([Aggregate Field]), [Calculated Field] WHERE [Search
Expression] GROUP BY [Group By Field] HAVING [Having Expression] SORT BY [Sort By
Field]".
```

The search expression supports a list of AND clauses and OR clauses with the following operators:

```
=, <>, !=, <, >, >=, <=
```

Arguments

Argument	Description
Asset Cache Key	[in] Unique identifier with which to query the cache. The key identifies the projected asset cache to validate, and it is returned by the Initialize method. (Required)
Search Expression	[in] Consider only assets that satisfy this Boolean expression. (Optional) Example: `([Product Type] = "Connection") AND ( ([Service Address] = "") OR ([To Service Address] = ""))`
Aggregate Function	[in] The type of query executed against the projected asset cache. Valid values are "Sum", "Count", "Max", "Min", "Avg" or "". (Optional) **Note:** Aggregate Function does not support the use of Date fields.
Aggregate Field	[in] The asset field considered by an aggregate query. Example: Bandwidth. (Optional) **Note:** Date fields are not supported in aggregate queries.
Calculated Field	[in] Unique identifier with which to query the cache. This key is returned by the Initialize method. The Calculated Field consists of a property set containing definitions for calculated fields that are returned as part of the result set. (Optional) Example: `{'Error Text', '[Count] [Product]s have no Service Address selected'}`

Argument	Description
Group By Field	[in] A property set containing a comma-separated list of fields or attributes used to group by when evaluating an aggregate function. (Optional) Example: `{'Service Address', ''},{'To Service Address', ''}`
Having Expression	[in] Consider only groups that satisfy this Boolean expression. (Optional) Example: `[Count] > 1`
Sort By Field	[in] A property set containing a comma-separated list of fields by which to sort the result set. (Optional) Example: `{'Product', ''},{'Service Id', ''}`

Result	[out] Property set of projected asset cache entries that satisfy the search expression. The property set contains child property sets that represent rows of the result.
	
	Example:
	
	`{{{'Count', '2'}, {'Service Address', '50 Main St., Denver, CO 80207'}, {'To Service Address', '101 California, New York NY 10234'}},`
	
	`{{'Count', '4'}, {'Service Address', '50 Main St., Denver, CO 80207'}, {'To Service Address', '901 Peach Tree, Atlanta, GA 98765'}}}`

Related Information

See *Initialize Method* and *Validate Method*.

Reset Method

The Reset method deletes the specified cache (or all caches, if not specified) from the Projected Asset Cache.

Argument

Argument	Description
Asset Cache Key	[in] Unique identifier with which to query the cache. This key is returned by the Initialize method. (Optional)

Related Information

See *Validate Method*.

Retrieve Method

The Retrieve method supports a combined Initialize or Query, and is essentially a wrapper around Initialize + Query. The input arguments are the same as those for Initialize. The output argument is that of Query.

Arguments

Argument	Description
Search Expression	[in] Search Expression with which to initialize the cache. Only Assets, Quote Line Items, Order Line Items and attributes that satisfy the search expression will be loaded into the cache. (Required)
Field	[in] Property set of fields to be retrieved by the Projected Asset Cache. (Optional)
Attribute	[in] Property set of attributes to be retrieved by the Projected Asset Cache. (Optional)
Future Date	[in] Project assets on this date. Only orders that have a due date less than this future date will be applied. (Optional)
Quote Id	[in] Row ID of the quote for which line items are to be queried and loaded into the Projected Asset Cache. (Optional)

Result	[out] Property set of projected asset cache entries that satisfy the search expression. The property set contains child property sets that represent rows of the result.
	
	Example:
	
	`{{{'Count', '2'}, {'Service Address', '50 Main St., Denver, CO 80207'}, {'To Service Address', '101 California, New York NY 10234'}},`
	
	`{{'Count', '4'}, {'Service Address', '50 Main St., Denver, CO 80207'}, {'To Service Address', '901 Peach Tree, Atlanta, GA 98765'}}}`

Using the VORD Projected Asset Cache Business Service

This topic shows how to structure the input parameters for the VORD Projected Asset Cache business service in a workflow.

To use VORD Projected Asset Cache business service in a workflow

- The structure for all input parameters must be as follows:
  - All values added to the input PropertySet must be added as child PropertySets.
  - All child PropertySets must have their type set by the SetType PropertySet method.

  For example, for each Field argument:
  - Create a new PropertySet, set its type to Field, and then use the SetProperty method to set the field name.
  - Add this new PropertySet as a child to the main property used in the business service.

  For each Search Expression argument:
  - Create a new PropertySet, sets its type to Search Expression, use the SetProperty method to set the property name as Search Expression, and then set Search Expression as the property value.
  - Add this new PropertySet as a child to the main property used in the business service.

  > **Note:** All fields used in the Search Expression field must be present in the Field PropertySet.

  Example:

```
var assetCacheSvc = TheApplication().GetService("VORD Projected Asset Cache");
var svcInputs = TheApplication().NewPropertySet();
var svcOutputs = TheApplication().NewPropertySet();
var field = TheApplication().NewPropertySet();
var search = TheApplication().NewPropertySet();
field.SetType("Field");
field.SetProperty("Network Element Type", "");
var CompoundProduct = "XXXXXXX";
search.SetType("Search Expression");
search.SetProperty("Search Expression", "([Compound Product Number] =
\""+CompoundProduct+"\")");
svcInputs.AddChild(field);
svcInputs.AddChild(search);
assetCacheSvc.InvokeMethod("Initialize", svcInputs, svcOutputs);
var assetCacheKey = svcOutputs.GetProperty("Asset Cache Key");
```

12 Compound Product Validation

Compound Product Validation

This chapter is a reference that explains the methods developed for the Compound Product Validation Engine (CPVE) business service. This chapter includes the following topic:

- *About Compound Product Validation Engine Business Service*
- *Compound Product Validation Engine Business Service Methods*

About Compound Product Validation Engine Business Service

The Compound Product Validation Engine (CPVE) business service, VORD CPVE Validation Service, is a set of methods that allows the Compound Product Validation Engine to be initiated through a Workflow.

The CPVE business service includes methods to validate a network and display rule violations. These methods are summarized in the following table.

Method	Comment
FindFutureDate Method	Gets the due date of the current line item.
Format Violation Method	Formats rules violations in a single string that can be displayed to the user.
Validate Method	Executes the cross-product validation rules associated with a compound product and returns any rule violations.
ValidateComplexProduct Method	Executes product validation expression validation rules associated with a complex product and returns any rule violations.
ValidateComplexProductAll Method	Executes product validation expression validation rules associated with the complex product in the document and returns any rule violations.
ValidateComplexProductFromPropertySet Method	Executes product validation expression validation rules associated with the complex product in the property set and returns any rule violations.

Compound Product Validation Engine Business Service Methods

The CPVE business service includes methods to validate a network and display rule violations. These methods are as follows:

- *FindFutureDate Method*
- *Format Violation Method*
- *Validate Method*
- *ValidateComplexProduct Method*
- *ValidateComplexProductAll Method*
- *ValidateComplexProductFromPropertySet Method*

FindFutureDate Method

This method gets the value of the field specified by FutureDateFieldName for the business component instance identified by BusinessComponentName and RowId.

The FindFutureDate method requires the current UI BC context.

Arguments

Argument	Description
BusinessComponentName	[in] Name of the business component from which to get the future date. (Required)
RowId	[in] Row Id of the business component instance for which to get the future date. (Required)
FutureDateFieldName	[in] Name of the field in the business component that stores the date. (Required)
FutureDate	[out] Value of the return date. (Required)
Returns	The value of the date field to use to validate the network at a date in the future.

Related Information

See the topic about workflows in *Siebel Order Management Guide* .

Format Violation Method

This method takes a property set of rules violations and formats them in a single string that can be displayed to the user.

Arguments

Argument	Description
RuleViolation	[in] Property set of child property sets of type 'RuleViolation'. (Required)
RuleViolationText	[out] Single string containing the formatted rules violations. (Required

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Validate Method*.

Validate Method

This method executes the cross-product validation rules associated with a compound product and returns any rule violations. It queries the Business Component VORD Compound Product Rule for all the rules defined for the top level product (highest-level network product) of the compound product (network). It then instantiates the business service for each of the rules and asks them for the fields and attribute values they need. It then initializes the Projected Asset Cache by asking it to build a future state of all the root line items within this compound product. This is done by querying the Quote Item, Order Item, Asset and their XAs for the fields and attributes required by all the rules, and then applying them to the associated assets. It then invokes the Validate method of each rules checker business service and creates a consolidated list of rules violations.

The Validate method is called within the VORD Validate (Order) and VORD Validate (Quote) workflows, which are invoked from the Quote and Order Network Applets.

Arguments

Argument	Description
CompoundProductNumber	[in] Compound Product Number (Network Id) associated with the compound product to be validated. (Required)
FutureDate	[in] Date at which to validate the compound product. (Optional)
QuoteId	[in] Row Id of the current quote. (Optional)
CompoundProductNumber	[out] Compound Product Number (Network Id) associated with the compound product. (Required)

Argument	Description
RuleViolationEmpty	[out] Y or N flag indicating whether there are any violations. (Required)
RootCompoundProduct	[out] Name of the compound product. (Required)
SiebelMessage	[out] Property set of child property sets of type 'RuleViolation'. (Required)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , and *Format Violation Method*.

ValidateComplexProduct Method

This method executes product validation expression validation rules associated with a complex product and returns any rule violations. It queries the Business Component VORD Compound Product Rule for all the rules defined for the top level product (network product) of the compound product (network). It then instantiates the business service for each of the rules and asks them for the fields and attribute values they need. It then initializes the Projected Asset Cache by asking it to build a future state of all the root line items within this compound product. This is done by querying the Quote Item, Order Item, Asset and their XAs for the fields and attributes required by all the rules, and then applying them to the associated assets. It then invokes the Validate method of each rules checker business service and creates a consolidated list of rules violations.

The ValidateComplexProduct method is called within the VORD Validate Complex Product (Order) or VORD Validate Complex Product (Quote) workflows, which in turn are called within the Verify Item (Order) or Verify Item (Quote) workflows.

Arguments

Argument	Description
FutureDate	[in] Date at which to validate the complex product. (Optional)
IsComplexProduct	[in] Y or N flag indicating whether the product is complex. (Optional)
QuoteId	[in] Row Id of the current quote. (Optional)
RootAssetIntegrationId	[in] Root Id of the integration asset. (Required)
RootProductId	[in] Root Id of the product. (Required)
RootProductName	[in] Root name of the product. (Optional)
RuleViolationEmpty	[out] Y or N flag indicating whether there are any violations. (Required)

Argument	Description
Siebel Message	[out] Property set of child property sets of type 'RuleViolation'. (Required)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , *Validate Method*, and *Format Violation Method*.

ValidateComplexProductAll Method

This method executes product validation expression validation rules associated with the complex product in the document and returns any rule violations. It queries the Business Component VORD Compound Product Rule for all the rules defined for the top level product (network product) of the compound product (network). It then instantiates the business service for each of the rules and asks them for the fields and attribute values they need. It then initializes the Projected Asset Cache by asking it to build a future state of all the root line items within this compound product. This is done by querying the Quote Item, Order Item, Asset and their XAs for the fields and attributes required by all the rules, and then applying them to the associated assets. It then invokes the Validate method of each rules checker business service and creates a consolidated list of rules violations.

The ValidateComplexProductAll method is called within the VORD Validate Complex Product All (Order) or VORD Validate Complex Product All (Quote) workflows, which in turn are called within the Verify Header (Order) or Verify Header (Quote) workflows.

The ValidateComplexProductAll method requires the current UI BC context.

Arguments

Argument	Description
BusinessComponentName	[in] Name of the business component. (Required)
FutureDate	[in] Date at which to validate the complex product. (Optional)
QuoteId	[in] Row Id of the current quote. (Optional)
RuleViolationEmpty	[out] Y or N flag indicating whether there are any violations. (Required)
SiebelMessage	Property set of child property sets of type 'RuleViolation'. (Required)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , *Validate Method*, and *Format Violation Method*.

ValidateComplexProductFromPropertySet Method

This method executes the product validation expression validation rules associated with the complex product in the property set and returns any rule violations.

Arguments

Argument	Description
IdFieldName	[in] Row Id of the field name. (Required)
RecordSet	[in] The record set. (Optional)
RootProductId	[in] Root Id of the product. (Required)
RootProductName	[in] Root name of the product. (Optional)
RuleViolation	[out] The rule violation. (Optional)
RuleViolationEmpty	[out] Y or N flag indicating whether there are any violations. (Required)

Related Information

See the topic about workflows in *Siebel Order Management Guide* , *Validate Method*, and *Format Violation Method*.

13 Copy Service

Copy Service

This chapter explains the use of the ISS Copy business service, or "Copy Service." This chapter is organized as follows:

- *About Copy Service*
- *Configuring Copy Maps*
- *Copy Service Methods*

Note: The Copy method of the Copy Service is retained for backward compatibility, but it is recommended that you use the Data Transfer Utility business service for new applications. For more information, see *Data Transfer Utilities Business Service*.

About Copy Service

Siebel order management copies data from one document type to another as a transaction progresses. Example transactions for which data is copied from one document type to another include the following:

- Quote-to-Agreement
- Opportunity-to-Quote
- Order-to-Agreement
- Agreement-to-Order

In earlier releases, the mapping between objects was either hard-coded (as in Quote-to-Agreement) or inconsistently defined (such as with business component user properties or SIS OM PMT mappings). In this release, all mappings between objects are defined in the Administration - Application screen, then the Data Map Administration views of the run-time client.

Using the Administration - Data views and the ISS Copy business service, you can create new mappings and update existing mappings. A mapping can support one or more business components from a business object (for example, quote, quote line item, and quote payments).

CAUTION: For Quotes and Orders, ISS Copy service is only used to copy Line Items and Attribute records during the base business component copy operation. After making changes to the repository (adding or removing columns), you must also change the data map for CopyQuote and CopyOrder so that the desired set of fields are copied.

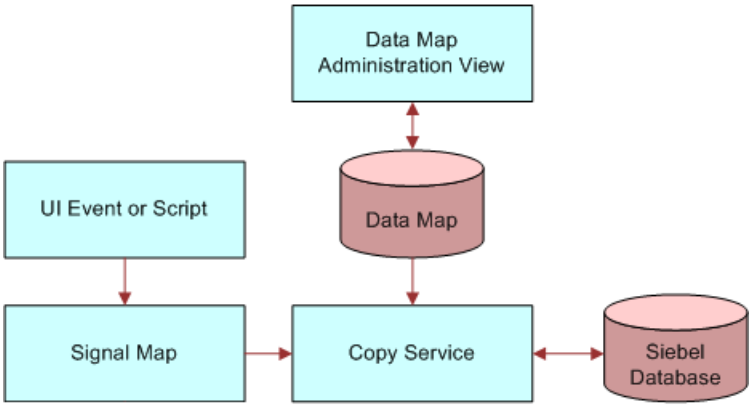
The ISS Copy business service (or "Copy service"), provides a mechanism for copying data from one business object to another. Use the Administration - Application screen, then the Data Map Administration views to define the business components and fields to be copied in a given situation.

Note: With Copy Service, data map object definitions are cached in the object manager. If you make changes to the definitions, you must restart the Siebel server.

Configuring Copy Maps

For information about configuring copy maps, see the topics about creating and validating data maps in *Business Processes and Rules: Siebel Enterprise Application Integration* .

Components of the Copy Service mechanism are shown in the following figure.



Data maps are cached in the Object Manager. Copy Service uses batched SQL updates for optimum throughput.

Copy Service Methods

The Copy Service includes the following methods, which can be called from script or signals:

- *GetFieldValueFromInstance Method*
- *LoadInstanceFromBC Method*
- *SetFieldValueFromInstance Method*
- *PopAndReleaseInstance Method*
- *Copy Method*
- *RefreshBCFromInstance Method*
- *CleanupEAI Method*
- *CleanupInstance Method*
- *LoadEAI Method*
- *SetupLineNumbers Method*
- *SetupSyncUpsert Method*
- *StoreEAI Method*
- *CheckEligibilityHelper Method*
- *CalculatePriceHelper Method*

GetFieldValueFromInstance Method

This method retrieves the business component field value from the instance.

Arguments

Argument	Description
InstanceName	[in] ISS business component instance name.
FieldName	[in] Business component field name.
BusCompName	[in] Used to specify a business component other than the instance business component, if necessary. (Optional)
FieldValue	[out] The field value.
SearchSpec	[out] A search specification of the following format: `[FieldName] = "FieldValue"`

LoadInstanceFromBC Method

The method loads the instance from the business component and pushes it into an instance. If SourceInstance is specified, the new instance is created by cloning the source instance and positioning it on the same row ID. Alternatively, BusObj, BusComp, and a search specification can be used to define the new instance.

The instance business component loaded must not have a parent defined in the BusObj.

Arguments

Argument	Description
SourceInstance	[in] The source ISS instance name. If specified, the new instance is constructed by positioning it on the same row ID as the source instance. (Optional)
InstanceName	[in] Required if SourceInstance is specified. Otherwise, optional default to “TheInstance”. (Optional)
BusObjName	[in] Business object name. (Optional)
BusCompName	[in] Business component name. (Optional)
SearchSpec	[in] Search specification on the business component. (Optional)

Argument	Description
InsertOnly	[in] If set to Y, ???? (Optional)

SetFieldValueFromInstance Method

The method sets the field value to an instance.

Arguments

Argument	Description
InstanceName	[in] ISS business component instance name.
FieldName	[in] business component field name
BusCompName	[in] Used to specify a business component other than the instance business component, if necessary. (Optional)
FieldValue	[in] Field value.
GetLOVValue	[in] LOV Type. If defined, the FieldValue is the LOV language-independent code. The actual field value will be the language-dependent display value as defined by LOV Type or Language independent code. (Optional)

PopAndReleaseInstance Method

This method pops out and releases the instances.

Arguments

Argument	Description
InstanceName	[in] The ISS instance name.

Copy Method

The method copies the source instance to the destination instance based on the defined copy map.

Arguments

Argument	Description
SourceInstance	[in] Source instance name.
DestinationInstance	[in] Destination instance name.
MapName	[in] Data map name.
CachedUpdate	[in] Y or N. Performance option to allow for cached updates. All SQL generated in the operation is issued to the database in one batch. Therefore, a SQL in the block cannot depend on a previous SQL being committed to the database. (Optional)
Release	[in] Source, Destination, or All. If defined, the corresponding instance(s) is popped up and released after the operation. (Optional)

RefreshBCFromInstance Method

The method refreshes the instance business component. If defined in the input argument, the line item business component is also refreshed.

Arguments

Argument	Description
InstanceName	[in] ISS business component instance name.
BusCompName	[in] Used to specify a buscomp other than the instance buscomp, if necessary. (Optional
BusCompLineItemName	[in] Line item buscomp name. (Optional)

CleanupEAI Method

CleanupEAI frees the memory used by the EAI data structure.

Syntax

Cleanup EAI <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
InstanceCollectionName	A string indicating the name of the instance collection

Usage

CleanupEAI frees the memory used by the EAI data structure. The name of collection is passed in as the InstanceCollectionName property.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

CleanupInstance Method

CleanupInstance frees the memory used by the CxObj data structure.

Syntax

Cleanup Instance <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
InstanceName	A string indicating the name of the instance

Usage

CleanupInstance frees the memory used by the CxObj data structure. The name of instance is passed in as the InstanceName property.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

LoadEAI Method

LoadEAI loads the product line item data structure through EAI and creates the CxObj memory structure for it.

Syntax

Load EAI <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
ParentObjectId	The Quote Id or Order Id or Agreement Id or parent Asset Id
IntegrationObjectName	The integration object name to use to load the object through EAI
SearchSpec	The search spec to apply when loading the object through EAI. Format: `[Header.Id] = 'QuoteId' AND [Line Item.Root Id] = 'RootId' (ex. "[Header.Id] = '2-4YOXR' AND [Line Item.Root Id] = '2-4YOXX'")`
InstanceCollectionName [optional]	A string indicating the name of the instance collection. Defaults to "TheInstanceCollection"
InstanceName [optional]	A string indicating the name of the instance. Defaults to "TheInstance"
RootId [optional]	The row ID of the root product. If empty, then a new row is created using the Product Id property. This is used during Validation of a product.
ProductId [optional]	Only used if RootId is empty. A dummy memory structure is created using the Product Id.

Output Arguments

Input Argument	Description
RootId	The row ID for the root product.

Usage

LoadEAI loads the product line item data structure through EAI and creates the CxObj memory structure for it. The CxObj structure is cached for use later by Configurator UI service and Configurator Service and also by Eligibility & Compatibility.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

SetupLineNumbers Method

SetupLineNumbers updates the line numbers for the Line Item being customized corresponding to the CxObj.

Syntax

SetupLineNumbers <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
HeaderIntCompName	Header Integration Component Name
Integration Object Name	The integration object name to use to load the object through EAI
InstanceCollectionName	A string indicating the name of the instance collection
InstanceName	A string indicating the name of the instance
Line Number Field	Line number field
LineItemIntCompName	Line Item Integration Component Name
Row Id	Line Item Id

Usage

SetupLineNumbers updates the line numbers for the Line Item being customized corresponding to the CxObj.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

SetupSyncUpsert Method

SetupSyncUpsert updates EAI operations on CxObj nodes for better performance during the subsequent sync using EAI.

Syntax

SetupSyncUpsert <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
InstanceCollectionName	A string indicating the name of the instance collection

Usage

SetupSyncUpsert updates EAI operations on CxObj nodes for better performance during the subsequent sync using EAI.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

StoreEAI Method

StoreEAI updates the line item by storing the CxObj structure using EAI.

Syntax

Store EAI <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
IntegrationObjectName	The integration object name to use to load the object through EAI
InstanceCollectionName [optional]	A string indicating the name of the instance collection. Defaults to "TheInstanceCollection".
ParentObjId	Quote Id, Order Id, Agreement Id, or Asset Id

Usage

StoreEAI updates the line item by storing the CxObj structure using EAI.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

CheckEligibilityHelper Method

Raises the GetEligibility signal on a configurator instance.

Syntax

CheckEligibilityHelper <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
InstanceName	Name of instance
Mode	Quote, Order, Asset or Agreement

Output Arguments

Output Argument	Description
Status	SUCCESS or ERROR
Error Message	(Optional) Error message, if any

Usage

Raises the GetEligibility signal on a configurator instance. This is invoked from the Batch Validate workflow. The workflow can be modified to skip this step if needed.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

CalculatePriceHelper Method

Raises the CalculatePrice signal on a configurator instance.

Syntax

CalculatePriceHelper <inputArgs>, <outputArgs>

Input Arguments

Input Argument	Description
InstanceName	Name of instance
Mode	Quote, Order, Asset or Agreement

Output Arguments

Output Argument	Description
Status	SUCCESS or ERROR
Error Number	(Optional) Error code, if any
Error Message	(Optional) Error message, if any

Usage

Raises the CalculatePrice signal on a configurator instance. This is invoked from the Batch Validate workflow. The workflow can be modified to skip this step if needed.

Invoked From

Browser Script, COM Data Control, COM Data Server, Server Script, Java Data Bean, Siebel Mobile Web Client Automation Server.

14 Data Transfer Utilities Business Service

Data Transfer Utilities Business Service

This chapter describes the Data Transfer Utilities business service. It includes the following topics:

- *About Data Transfer Utilities*
- *About Data Maps*
- *Example of Defining Data Maps to Use with the DTU*
- *Examples of Invoking the DTU*
- *Data Transfer Utilities Methods*

About Data Transfer Utilities

This business service allows you to transfer data from a source business component to a destination component. For example, a user can enter data in one view, then use a toolbar command to navigate to another view. Data entered in the first view is automatically entered in the second view.

Considerations for Data Transfer Utilities

This topic covers considerations that are important when you use the Data Transfer Utilities.

CAUTION: Spool SQL statements during the development stage to verify that all operations are performed.

Use of Active Business Objects

Data Transfer Utilities execute inside a client's object manager.

The DTU reuses the current active business objects. It does not instantiate an independent source business object unless directed. This leads to both a leaner memory use and better performance. This is even more so if the destination business object is the same as the source business object. In such a case, no new business objects are instantiated for the business service.

Because of the reuse of active objects, you must exercise caution to preserve the current business object context. For example, the business components must not be in the query state when DTU is launched.

Invocation Context

You can pass a reference to the active Buscomp to DTU, if you invoke the business service from Event Manager, Buscomp Named Method, or from a workflow process that is invoked by the event manager. You are not required to have an active Buscomp.

Not all Buscomp events can be used to invoke the DataTransfer method. For example, Query event in general must not be used to trigger DataTransfer, as the buscomps are not in an updateable state.

In general, use PreDeleteRecord event; do not use DeleteRecord event. The Siebel event manager does not pass in a reference to the active Buscomp in the DeleteRecord event.

Use special care when the service is used with other business services in a workflow. Other business services should not interfere the passing of a reference to the active buscomp. Use a spooled SQL statement to confirm the operations carried out are correct.

Well-Positioned Buscomps

A well-positioned Buscomp is a Buscomp that has been positioned correctly, and whose position should not be disturbed. DTU uses the following rule:

- The initiator Buscomp is a well-positioned Buscomp.
- The ascendants of a well-positioned Buscomp are well-positioned.
- For a given data map component, the buscomps involved in all its parent data map components are well positioned.

If the source Buscomp is well-positioned:

- Data transfer is only invoked on the current row of the source BusComp. Otherwise, the operation is carried out on all rows in the source BusComp at the moment of invocation.
- Advanced options such as source search spec, source sort spec should be empty.

If the destination Buscomp is well-positioned:

- You do not need to specify key fields to retrieve the destination record. Even if you do in this case, DTU would ignore them.
- And if the current operation is Insert, it would change to Update by default, unless overridden by Operation Overrides.

Example of Buscomp That is Not Well-Positioned

Calling the DTU Data Transfer method from an applet that is based on buscomp Quote, but which has the data mapped source buscomp Opportunity. In this case:

- The UI context (Quote) does not match the data mapped buscomp (Opportunity), and
- DTU must query all Opportunity buscomp records, unless the RowId parameter is additionally passed, to identify a single opportunity record.

Recursive Invocation

By default, you cannot use the DataTransfer operation to invoke another DataTransfer operation. In other words, at anytime within a client's object manager, there is only one DataTransfer method in the call stack.

Using Named Parameters in DTU

You can use named parameters to pass in a run-time dynamic value into Data Transfer Utilities (DTU). For example, imagine you want to pull a contact's latest information into your Buscomp. At design-time, you cannot foresee what is the contact’s ID. Instead, you use a named parameter, &ContactId, and at run time, you pass in the value &ContactId.

Named parameters are defined implicitly in two ways:

- **Business Service Arguments.** Pass in the named parameters when DTU is invoked. DTU knows an argument is a named parameter if the argument name is prefixed with an ampersand (&).

For example:

```
var psinputs, psoutputs;
var myContactId = '0-45TU890';
psinputs = TheApplication().NewPropertySet();
psoutputs = TheApplication().NewPropertySet();
psinputs.SetProperty ("DataMapObj", "My Test DTU Object");
psinputs.SetProperty ("Operation", "Update");
psinputs.SetProperty ("&ContactId", myContactId);
var obs = TheApplication().GetService("FINS Data Transfer Utilities");
obs.InvokeMethod ("DataTransfer",psinputs, psoutputs);
```

&ContactId serves as a named parameter.

The input value of a named parameter can be a calculation expression. In order to do so, set the value to:

```
Expr: "YourExpression"
```

which is the syntax of Buscomp field predefault. At run time, the expression is evaluated against the initiator Buscomp. For more information about initiator buscomp, see the argument description in *DataTransfer Method*.

- **Assignment by DTU.** At run time, you can transfer into a named parameter if the field type is Parameter. When this happens, if the named parameter is still not defined, it is instantiated.

A named parameter must be implicitly defined first before it can be used. In other words, un-assigned named parameters cannot be used.

Named parameters can be used to define Data Map Component Advanced Options, and Data Map Field Source or Destination that are of type Expression.

All named parameters are output into the DTU service output arguments.

Tip: Whenever a named parameter is used in DTU, it must be prefixed with &.

Calculation Expressions in DTU

When Data Map Field Source or Destination Type is Expression, the Source or Destination Fields are calculation expressions that follow Siebel Query Language syntax. See *Siebel Personalization Administration Guide* for more information about Siebel Query Language.

DTU contains two extensions to Siebel Query Language: curly bracket pair {field}, and named parameter.

Curly Bracket Pair {field}

Use this syntax to refer to a buscomp field at the other business component side. For example, if you define the following expression as the source:

```
{ContactId}
```

[ContactId] would be evaluated at the destination bsucomp. When {} is involved, please note that {} takes the highest precedence over other operator. Thus, if you have an expression like:

```
'{Last Name}' + 'Test'
```

{Last Name} has precedence over quotes "". If the person's last name is Agee, {Last Name} is evaluated to be "Agee". As a result, the final value is:

```
"Agee" Test

instead of

{Last Name} Test
```

Named Parameter

A calculation expression can contain named parameters, using the syntax of [&Parameter]. It is important to pre-fix the ampersand to indicate a named parameter. For example,

```
"Sadmin's opportunity #" + [&OpptyId]
```

For more information about named parameters, see *Using Named Parameters in DTU*.

Using DTU with Order Management Signals

Order Management signals are a powerful infrastructure that allow you to define complicated runtime events. For more information, see *Signals*.

DTU can be directly invoked through a signal. When this happens, the signal infrastructure passes the row ID of the current instance buscomp to the RowId argument of DTU. This tells DTU which source buscomp record to work with.

In a rare case when you invoke DTU from a signal, but you do not want DTU to use RowId passed from the signal as the context, simply set the IgnoreRowId argument to Y.

Since signals pass the RowId argument instead of the buscomp instance into DTU, a new instance of source buscomp is always created.

About Working with Hierarchical Business Components

Hierarchical business components, such as *Quote Item*, are business components that define a recursive hierarchical foreign key field to themselves. In Web Tools, these buscomps have the *Hierarchical Parent Field* attribute defined. Sometimes, their buscomp user properties also contain a definition for *Root ID Field Name*. For more information, see *Siebel Tools Online Help* .

When the data of a hierarchical buscomp is copied to another hierarchical buscomp, care must be taken to re-wire the hierarchical foreign keys: the hierarchical parent field and the root ID field. They must be re-wired to point to the corresponding destination records. DTU automatically re-wires those two foreign keys when a hierarchical buscomp is copied to another hierarchical buscomp.

For performance reasons, many data map objects in Siebel Order Management use a flattened version of a hierarchical buscomp instead. For example, data map object QuoteToSalesOrder, which creates a sales order based on a quote, contains the Line Item component that uses Quote Item (Simple) and Order Entry - Line Items (Simple). Both simple buscomps are not hierarchical. The reason for using simple buscomps is performance, as hierarchical buscomps require more CPU and memory. However, when those simple buscomps are used, you must define the foreign key mapping

yourself. This is generally achieved using the field-level advanced option ID Mapping Component described in the following table.

This option was referred to as MapId in ISS Copy Service (described in *Data Map Fields*), which is not used by DTU in 8.0. For more information, see *ISS Copy Service and the Data Transfer Utility*.

In rare cases, you do not want DTU to automatically set hierarchical parent ID and root ID when a hierarchical buscomp is copied to another hierarchical buscomp. Set the component advanced option *Disable Hierarchy* to Y. You must type this option yourself, as it is not available from the pick list in the data map administration screen.

ISS Copy Service and the Data Transfer Utility

The DataTransfer method of the DTU replaces the Copy method of the ISS Copy Service. DTU offers more functionality and usability.

ISS Copy Service is still supported. Earlier configurations that use the Copy Service will still work with the Copy Service in 8.0. Only the AutoOrderSalesQuote and AutoOrderServiceQuote signals are re-configured to use DTU. It is recommended that new development be based on DTU.

Some differences are between DTU and ISS Copy Service are:

- DTU does not require you to instantiate the source and destination instances first. Instead DTU instantiates and deletes them automatically. This makes the invocation easier and safer.
- Copy service data map field options include:
  - **SequenceField.** For the DTU data map, SequenceField is renamed as Sequence Field.
  - **MapId.** For the DTU data map, MapId is renamed as the ID Mapping Component option. The difference is that the MapId option value is a source buscomp name, and ID Mapping Component is the data map component name, as DTU allows a buscomp to be used in multiple components.

  For backward compatibility, DTU still recognizes the copy service field options.

Configuring Event-Based Commands for DTU

The Data Transfer Utilities (DTU) business service allows you to configure toolbar and menu commands based on Siebel Event Manager.

To configure event-based commands

1. In Web Tools, create or open a workspace and then navigate to Object Explorer.

   To use the workspace dashboard, see *Using Siebel Tools* .
2. Define a command where:

   Business Service is “FINS Data Transfer Utilities” and method is “FAFireEventxxx”.

   The method name can be anything that begins with “FAFireEvent”. When the command is invoked, it, in turn, invokes method EventMethodxxx on the primary buscomp of the active view, where xxx is of the same value as in FAFireEventxxx.
3. Define a toolbar.

4. Define a toolbar item for the command you defined.
5. Save your changes and submit the workspace for delivery.
6. In the Siebel client, define a run-time event that will receive EventMethodxxx.
7. Navigate to the Administration - Runtime Events screen, then the Events view, and create a Buscomp run-time event as listed in the following table. See *Siebel Personalization Administration Guide* for more information about run-time events.

Field	Entry
Sequence	-1
Object Type	BusComp
Object Name	The name of the business component in which the event is invoked. For a toolbar command, this is the primary business component in the view in which the command is invoked.
Event	InvokeMethod
Sub Event	EventMethodxxx. Choose the same value for xxx that you chose for FAFireEvent.
Action Set Name	The action set that invokes Siebel Workflow Manager or a business service.

   Alternatively, you can define a workflow that has a Start step that contains run-time events. When the workflow is activated, both the Action Sets and run-time events are created automatically for you. For more information, see *Siebel Business Process Framework: Workflow Guide* .
8. Siebel run-time events are cached. After you make changes, click the Runtime Events applet menu item Reload Runtime Events.
9. Configure dynamic enabling of the command. For more information, see *Dynamic Enabling of Commands for DTU*.
10. Define command visibility:
    a. In Siebel Tools, navigate to Business Services, then FINS Data Transfer Utilities.
    b. Define a user property in which Name is MethodName GotoView, and Value is set to the name of a view. MethodName is the name of the command method.

       When you define this user property, this method is enabled only for users who have visibility to the view defined in the value. If the method does not contain a GotoView, visibility is not imposed on the method.

Dynamic Enabling of Commands for DTU

When a command is invoked from a toolbar button or menu, the Data Transfer Utilities business service invokes the method EventMethodxxx on the primary business component of the active view. The primary business component should be derived from CSSBCBase to allow the invocation to be captured by Siebel Event Manager.

When the view is changed, Siebel framework polls each command for the application-level toolbar buttons and application menu to determine whether the button or menu items should be made read-only.

There are two mechanisms for the dynamic enabling and disabling of commands in DTU:

- Srf mode
- Mock Event Sink

The System Preference FINS DTU Enable FireEvent Mode is used to determine the mode. The value should be Srf or Runtime Event. The default value is Srf.

Srf Mode

In the Srf mode, a FAFireEventxxx invocation on a buscomp is enabled if there is a user property underneath the FINS Data Transfer Utilities business service as such that the name of the user property is:

Name: FAFireEventxxx Static Enabled BC [n]

Value: Buscomp Name

You can define multiple Buscomps for a FAFireEventxxx method.

Srf mode is introduced primarily for performance reasons. Compared with Runtime Event mode, it allows fast enabling and disabling of a command button without actually invoking a run-time event. Srf mode is the default mode.

Mock Event Sink

When System Preference FINS DTU Enable FireEvent Mode is Runtime Event, Data Transfer Utilities determines at run time whether a FAFireEventxxx method should be disabled or not by initializing Mock Event Mode. It sets up a global mock event flag within the client's object manager. It then invokes EventMethodxxx on the primary business component of the active view. If this EventMethodxxx is finally captured by a Mock Event Sink, a global response flag is set. When the Data Transfer Utilities finds out the response flag is set by a mock event sink, it enables the FAFireEventxxx method for that particular view. Otherwise, the method is disabled.

Mock Event Sinks are specialized business service methods that capture mock events. They check whether the client's OM is in the mock event mode. If not, they do nothing. If so, they reply to the mock event by setting the response flag as well as the output argument.

MockMethodReplied is Y.

TryMockMethod in Data Transfer Utilities is a mock event sink. DataTransfer method has a built-in mock event sink.

Performance Tuning for DTU

You can improve performance of DTU by using the following tips:

- "When the operation is insert, and you are sure no duplicate records would be created in the process, do not check Key flags. This means that DTU does not need to query the database to identify duplicates.
- Use BatchMode in the Option argument whenever possible.
- Use Cached Updates whenever possible.
- For bounded picklist fields, MVG primary ID, and MVG source ID fields, skip field validation whenever possible. Note that skipping the bounded picklist validation would also skip setting fields of the pick map.

- When a reference to the active instance buscomp is passed to DTU, it is a delicate balance to decide whether or not to launch the DTU with a new instance of the source buscomp or re-use the current active instance. By using the active instance, you avoid using CPU and memory for the new business object. On the other hand, when DTU works off the active instance, it has to restore the buscomp context to its original state, causing refreshing and looping of buscomps. When a large number of buscomps or buscomp records are involved , this can be expensive. This is why the NewSrcBusObj option is used in the "Auto Order Web Service" example.

About Data Maps

Data maps are the logic defining the flow of data from one location to another. The DTU business service uses data maps to transfer data from one location in the Siebel application to another.

This topic includes information about the following:

- *Data Map Objects*
- *Data Map Components*
- *Data Map Component Advanced Options*
- *Data Map Fields*
- *Data Map Field Advanced Options*
- *Migrating Data Map Objects Between Environments*

Data Map Objects

Data map objects indicate the data that is being transferred from the source business object to the destination business object. You can use the Administration - Application screen, then the Data Map Administration view to define data map objects. Only Siebel administrators have access to this screen. Data map objects are described in the following table.

Object	Description
Name	Data map object name. Enter a unique name.
Source Business Object	Source business object name.
Destination Business Object	Destination business object name. You can specify the same business object as both source and destination business object.
Inactive	Check this box to make the data map object inactive.

Tip: Data map objects are cached in memory. Whenever you make changes to an existing data map object, click the Clear Cache button to refresh the cache so that your changes appear.

You can import or export data map objects as XML files through the Data Map Object applet menu items: XML Import, XML Export.

Data Map Components

Data map components define the mapping at the child business component level. Each data map object can contain multiple data map components. You can arrange data map components in a parent-child hierarchy, but you must specify a parent for all except one data map component. The parentless data map component is called the root data map component. Data map components are described in the following table.

Component	Description
Name	Data map component name. Enter a unique name for each data map component in a data map object.
Source or Destination Business Component	Source or destination business component name. If you specify a parent for this data map component, you must define this business component as a child of the source or destination business object to which the parent data map component is mapped.
Parent	Parent data map component name. If you: • Specify a parent, the parent is mapped to particular source and destination business components. Generally, you map the child data map component to a child of those source and destination business components. • Do not want to specify a parent, leave it empty to indicate that this is the root data map component. Each data map object can have one or more root data map components.
Inactive	Check this box to make the data map component inactive.

Data Map Component Advanced Options

Fine-tune data transfer at the component level by using Advanced Options multi-value fields. Data map components advanced options are described in the following table.

All advanced option values can contain named parameters. At run time, the named parameter is substituted by its run-time value.

If the source Buscomp has been well positioned, the source search spec, the source sort spec, and source record row number must be evaluated to be empty at run time, otherwise a wrong advanced option error is encountered. See *Well-Positioned Buscomps* for more information.

Advanced Options does not apply to multi-value group subcomponents.

Component	Description
Source Search Specification Source Sort Specification	Defines the source Buscomp search spec and sort spec. The value can be a literal search spec or sort spec string. It can also contain a named parameter. See *Using Named Parameters in DTU*. For example: `[Id] = [&ContactId]`

Component	Description
	where ContactId is a named parameter. At run time, only named parameters are replaced by their string values.
Source Record Row Number	One can selectively transfer only a subset of source Buscomp records. This can be defined in three formats: • Start-End • Start- • Number For example, 0-5, 4-, 0. **Note:** The row number starts at 0.
Operation Override	This option allows one to override the operation at the component level. For example, if the current operation is Insert, you can use this option to set some component to operate Update instead.
No Association	Y or N. Applicable to buscomps that have association list. By default, it is Y. Data Transfer Utilities first try to locate the desired destination record in the associate list. If successful, the located record is associated. Otherwise, a new record is created. If N, association of existing records is not attempted. A new record is created instead.
Cached Updates	The valid values include: • Source Component • Destination Component • Source or Destination Component This is a performance enhancement option that defers sending SQL statements to the database until they can be sent together. Since none of the SQL statements is sent until the end, subsequent steps in the block cannot be dependant on a previous step having been committed to the database. When you turn on this option, make sure to confirm all SQL statements are generated correctly and there is no inter-dependency.
Field Validation	Y or N. By default it is Y. By default, setting field values in a Siebel BusComp triggers field value validation. If you have a bounded picklist, the new value is validated against the picklist and the fields in the pick maps are set. Disabling field validation also turns off picklists. It is a performance enhancement option.
Source Filter Specification	An expression in Siebel search specification syntax that is used to filter out records at the runtime. Only source records whose filter specification evaluates to be true are transferred. Filter specification differs from search specification in that search specification is imposed at the database query, and filter specification is imposed while looping through the records.
Disable Order Management Signals	The valid values include • Source Component • Destination Component

Component	Description
	• Source or Destination Component When the Disable Order Management Signals option is imposed on the source or destination components, buscomp operations on those buscomps do not trigger order management signals during the data transfer process.

Data Map Fields

Data map fields define the field-to-field mapping. Data map fields are described in the following table.

Field	Description
Source Type or Destination Type	Type of the source or destination field. Can be: Field, Expression, or Parameter: • **Field.** A Siebel Buscomp field. • **Parameter.** A named parameter. The parameter must be prefixed with an ampersand (&). See *Using Named Parameters in DTU*. For example: `&ContactId` • **Expression.** A Siebel calculation expression. See *Calculation Expressions in DTU*.
Source or Destination	The contents of these fields depends on the source and destination type. If the type is: • Field, use Buscomp field name. • Expression, use a Siebel calculation expression. • Parameter, use a named parameter. If the destination field is a calculated expression, then the record is not used to update the destination Buscomp. Instead, the result of the expression, evaluated at run time, is written back into the source field at the end of the data transfer operation of the component. If the Source is: • A Buscomp field, then source Buscomp is updated. • A Parameter, the corresponding named parameter value is updated. • An Expression, nothing happens.
Key	Matches the destination records with source records. For example, the Update operation updates the record in the destination business component whose key destination fields all match those of the corresponding source fields. Each data map component in general contains at least one key field. When there is no key defined, if the operation is:

Field	Description
	• Insert, DTU would proceed without checking if a duplicate record with the same key fields already exists. • Update, it would update the current destination record. If the destination business component is populated with an associated list business component, at least one key field is required.
Source or Destination Multi-Value Link	This link indicates that the source and destination fields are multi-value fields. Data is transferred from one multi-value field to another by dividing data map fields into several subcomponents. All entries with the same source and destination multi-value link constitute a subcomponent. Specify a key for each subcomponent. **Note:** Data transfer from a multi-value field to a single-value field is not allowed.

Data Map Field Advanced Options

Data map field advanced options allow you to fine tune data transfer operations at the field level. These options are described in the following table.

Option	Description
Field Validation	Possible values are Y or N. The default is Y. For more information, see the Field Value component in *Data Map Component Advanced Options*. If Field Validation option is defined at both component and field levels, the field level definition wins.
Sequence Field	Possible values are Y or N. The default value is N. If the value is Y, a sequence number starting from 1 and increased by 1 is assigned to the destination field in each record.
Id Mapping Component	The option is used to re-wire foreign keys. The valid values are the names of data map components in the data map object. At run time, the source ID to destination ID mapping of the named data map component defines the foreign key mapping used by the data map field. For example, when you copy from quote item to order item in a data map component *Items*, the hierarchical parent ID of a quote item cannot be literally copied to the order item parent ID. It needs to be re-wired to the corresponding order item ID. The ID mapping component must be *Items*, as the quote to order ID mapping is used to look up the order parent ID field using the value of the quote parent ID field. For related information, see *Configuring Event-Based Commands for DTU*.

Migrating Data Map Objects Between Environments

Data map definitions are enabled for export and import by the Application Deployment Manager (ADM). For information about using ADM, see *Siebel Application Deployment Manager Guide* .

> **Note:** Data Maps can be Workspace enabled in your development environment. If they have been Workspace enabled in your development environment and you are working in that environment, then you can migrate Data Maps like all other Repository Data using the Migration Application. ADM Migration will not be applicable if Data Maps have been Workspace enabled.

Example of Defining Data Maps to Use with the DTU

This topic shows how data mapping is defined to convert a quote into a sales order using the DTU. This is in essence Auto Quote function of the DTU. This example guides you though the mapping in the application.

The data copied over mainly consists of a three-level hierarchy:

- Header
- Line items
- Extended attributes (XA)

There are also other auxiliary entities such as payments, requested, and promised scheduled lines. This example shows how header (*Mapping Headers*), line items (*Mapping Line Items*) and XA (*Mapping the Extended Attributes*) are mapped. Other entities are mapped like either line items or XA. First, you must find the data map object where the header, line item, or XA are mapped as described in *Finding the Data Map Object*.

Finding the Data Map Object

You must find the data map object where the header, line items, and XA are mapped.

To find the data map object

1. Navigate to the Administration - Application screen, then the Data Map Administration view.
2. In the Data Map Object list, select the record named QuoteToSalesOrder.

   The components of this data map object, including Header, Line Item, and XZ, appear in the Data Map Component list, after the Data Map Object list.

Mapping Headers

First, look at the mapping of the header component, to see how the quote header buscomp is mapped to the order header buscomp.

Notice that a calculation expression is used to look up the order type LOV to default the order type to sales order.

Two advanced options at the component level are of interest, because they are both important to improve performance:

- **Cached Updates.** Enables cached updates at the destination buscomp level. All updates at the order buscomp, and its descendant buscomps, are cached and issued in one batch to the database.
- **Skip Order Management Signals.** Because of this option, setting field values at the order buscomp does not trigger order management signals that usually would invoke pricing and eligibility workflows.

To view the header mapping

1. In the Data Map Component list, select the record named Header.
2. Look at the mapping in the Data Map field list.

   Notice that the record with Expression in the Source Type field has the expression LookupValue("FS_ORDER_TYPE","Sales Order"), which is used to default the order type to sales order.
3. Click the select button in the Advanced Options field of the Header record to see that Cached Updates and Skip Order Management Signals are selected as options.

Mapping Line Items

Next, look at the Line Item component.

The record defines its parent as Header, so it is invoked as a child of the Header component.

To improve performance, use simple buscomps instead of the line item buscomps used in the Siebel Call Center user interface. These improve performance because they are light-weight and, most important, because they are not defined as hierarchical buscomps.

Because they are not hierarchical, you must define their hierarchical parent fields in the mapping, using the Data Map Field records with Root Quote Item ID and Parent Quote Item ID in their Source field.

Both these records use the advanced option with the name ID Mapping Component to define the foreign keys to define hierarchical parent fields.

> **Note:** For backward compatibility, the application retains the advanced option MapId, which functions like ID Mapping Component.

For more information, see *Configuring Event-Based Commands for DTU*.

To view the line item mapping

1. In the Data Map Component list, select the record named Line Item.
2. Look at Advanced Option and the mapping in the Data Map field list.

Mapping the Extended Attributes

Next, look at the XA component.

Note that deep XA buscomps are used. In Web Tools, these buscomps are defined as children of the header buscomps, not as children of line items. This improves performance, because one query can retrieve all grand-children XA

attributes that belong to a quote. To mirror the Tools configuration, XA is defined as a child of Header in the Data Map Component.

As deep XA buscomps are defined as children of the header, you cannot rely on Siebel buscomp links to set the parent line item IDs, which are stored in the Object Id field. You also cannot literally take object ID from quote to order, as the quote XA object ID points to a quote item, and the order XA object ID points to an order line item. You need to use ID mapping to look up the foreign keys in the order side. This is realized by the advanced options:

- Name: ID Mapping Component
- Value: Line Item

Note: For backward compatibility, the application retains the advanced option MapId, which functions like ID Mapping Component.

For more information, see *Configuring Event-Based Commands for DTU*.

To view the line item mapping

1. In the Data Map Component list, select the record named XA.
2. Look at Advanced Option and the mapping in the Data Map field list.

Examples of Invoking the DTU

The following topics illustrate the two ways of invoking the DTU, as they are used in the Siebel application, and how to use DTU services:

- *Example of Invoking the DTU from a Signal: Auto Sales Order*
- *Example of Invoking DTU from a Workflow: Auto Order Web Service*
- *Example of Using DTU Services* shows how to use a DTU business service to copy data from one business component to another business component.

Example of Invoking the DTU from a Signal: Auto Sales Order

Auto sales order is a function that creates a sales order from the current quote. It can be invoked by clicking the Sales Order button in the Quote screen, then the Order view. This button invokes the signal AutoOrderSalesQuote. This example shows you how this signal invokes the DTU.

To see how the signal Auto SalesOrderQuote invokes the DTU

1. Log into Siebel application as an administrator.
2. Navigate to the Administration - Order Management screen, then the Signals view.
3. Query for AutoOrderSalesQuote.
4. In the last version list applet, drill into version 3.
5. You can see the three actions taken by the signal. Be sure that the you have selected the action that uses the method DataTransfer of the DTU Business Service, which copies the data into a new order.
6. In the Properties list, you can view the arguments of the DataTransfer method in the Parameter list, as shown in the following table:

Name of Parameter	Value
DataMapObj	`QuoteToSalesOrder`
Operation	`Insert`
Option	`/BatchMode`
SharedGlobalDestID	`Y`

At runtime, the signal infrastructure also passes the RowId of the current quote ID to DTU.

7. The SharedGlobalDestId parameter allows the destination record ID to be output to a shared global variable called SharedGlobalDestId. This variable is picked up in by the RefreshBCFromInstance method, whose arguments are shown in the following table:

Name of Parameter	Value
BusCompName	`Order Entry - Orders`
InstanceName	`ISS Instance`
TargetRowIdShared Global	`DTUSharedGlobalDestId`

The TargetRowIdSharedGlobal argument repositions the order buscomp to this row ID after refreshing, which is the row ID of the order just created.

Example of Invoking DTU from a Workflow: Auto Order Web Service

The Auto Order web service exposes Siebel's auto order function as a web service. It takes a quote row ID as the input argument, creates an order based on the quote, and returns a quote integration object. The web service is implemented by a workflow.

To see how the workflow SISOMAuto OrderWebService invokes the DTU

1. In Oracle's Siebel Tools, locate the workflow SISOMAutoOrderWebService.
2. In this workflow, select the step DTU Auto Order, which uses DTU to create a new order from a quote using the DataTransfer method.

3. Display input arguments, which have the values shown in the following table:

Input Argument	Sequence	Type	Value	Property Name
DataMapObj	1	Literal	`QuoteToSalesOrder`	None
Operation	2	Literal	`Insert`	None
Option	3	Literal	`/BatchMode/NewSrcBusObj`	None
RowID	4	Process Property	`None`	Object Id

Notice that there are some subtle differences from the input arguments of the previous example:

- Here, RowId is explicitly passed in as an argument, but when you use a signal, the order management infrastructure implicitly passes in a RowId argument.
- The Option argument contains "NewSrcBusObj". In the previous example, since signals pass the "RowId" argument instead of the buscomp instance into DTU, a new instance of the source buscomp is always created. This workflow specifies its Business Object as Quote. As a result, DTU receives a reference to the quote business component as part of the invocation context. Because you use NewSrcBusObj in this argument, DTU does not work off the instance of the quote buscomp associated with the workflow. Instead it creates a new instance of the quote business object. For information about why NewSrcBusObj is used, see *Performance Tuning for DTU*.

4. Display the output argument for this step, which has the values shown in the following table:

Property Name	Sequence	Type	Output Argument
ActiveOrderId	5	Output Argument	&OrderId

5. The newly created order ID is output through the DTU named parameter &OrderId. The parameter is defined in the data map Header component data map field view, which has the values shown in the following table:

Order	Source Type	Source	Destination Type	Destination
52	Parameter	&OrderId	Expression	[Id]

Example of Using DTU Services

The following example shows how to use the FINS Data Transfer Utility business service to copy data from one business component to another business component.

To copy a business component and specify a search specification

1. Create an input property set and search specification as follows:

```
Operation = "Insert"
Option = "/NewSrcBusObj /NewDstBusObj"
DataMapObj = "CopyContact"
&ContactId = "<the contact row_id you want to copy>"
```

2. Configure DataMapObj ("CopyContact") to include a named parameter as follows:
   a. Go to the root level data map component in the Data Map Component Applet.
   b. Open the Advance Options Picker.
   c. Add following New Advanced Option:
      - Name: Source Search Specification
      - Value: [Contact Id]=[&ContactId]
   d. Clear cache.
3. In the business service simulator, invoke the FINS Data Transfer Utility business service using the DataTransfer method, passing in the input property set specified in *Example of Using DTU Services*.

   The Siebel CRM client starts copying all the contacts.
4. From the documentation, it may seem like the Init parameters are needed, but they are not. The InitBO/InitBC input arguments cannot be used to construct a buscomp to start DTU. They are used to indicate from which buscomp of the active busobj you want to use to launch DTU.

Data Transfer Utilities Methods

This topic describes the following Data Transfer Utilities business service methods.

DataTransfer Method

The DataTransfer method transfers data from the source business component to the destination business component. Its arguments are described in the following table.

Argument	Description
Data Map Object (Required)	The name of the data map object that defines the mapping.
Operation (Required)	Valid entries include Insert, Update, Delete, and Upsert.
GotoView (Optional)	The name of a view that appears to users after the data transfer operation.
Option (Optional)	A text field that allows you to specify additional options for the operation. Supported options include: • NewSrcBusObj. Force to instantiate a new Source business object. Use instead of NewBusObj. • NewDstBusObj. Force to instantiate a new destination business object.

Argument	Description
	• RootNotCommitted. Suggest DTU not to commit the root component, if possible. • SrcRootAdminMode. Set the source Buscomp of the root data map component to Admin mode. This is valid only if the root source Buscomp has not been executed. • BatchMode. This is a performance enhancement option that suppresses runtime events, disables undo, and defers field pre-defaults until committing the record. Batch mode is only enabled for source or destination business objects that are not the active (initiator) business object.

	The following syntax is recommended for defining Option:
	`/option1 /option2 ...`
	For example,
	`/NewSrcBusObj /NewDstBusObj`

Initiator Business Object (Optional)	Used as a sanity check. If the BusObject that invokes DTU is different from what is specified by the InitBO argument, DTU exits as an external error. Initiator Business Object is part of the invocation context. DTU receives a reference of the initiator business object only when invoked from Runtime Evens, Buscomp Named Methods, or workflow processes with its business object defined. DTU can be invoked without an initiator business object.
Initiator Business Component (Optional)	By default, the Buscomp that invokes DTU serves as the InitBC. Initiator Buscomp plays an important role in determining how records are transferred. Use the InitBC argument to set other Buscomp in the Initiator BusObject as the Initiator Buscomp.
Initiator Search Specification or Initiator Sort Specification (Optional)	Initiator Buscomp search spec and sort spec.
Initiator Buscomp Enumerate Flag (Optional)	Y or N. By default, Initiator Buscomp Enumerate Flag is N. When it is true, DataTransfer is applied to each record in the initiator Buscomp. When InitSearchSpec or InitSortSpec is specified, InitEnumFlag is implicitly true, even if InitEnumFlg is set to N.
MockMethodReplied	Y or N.
RowId (optional)	The ID of the root source buscomp record. For more information, see *Using DTU with Order Management Signals*.
IgnoreRowId (optional)	Y or N. If Y, RowId argument is then ignored. For more information, see *Using DTU with Order Management Signals*.
SharedGlobalDestId (optional)	Y or N. If Y, the destination record ID is output to a shared global called DTUSharedGlobalDestId.

Note: It is recommended that you specify both InitBO and InitBC specifically when invoking DTU. DTU requeries the initiator buscomp when InitSearchSpec, InitSortSpec, InitEnumFlg are used.

FAFireEventxxx Method

FAFireEventxxx is a hidden method that you can use to create a toolbar command. It invokes the method “EventMethodxxx” on the primary business component of the active view. “EventMethodxxx” triggers the event manager, which invokes either a workflow or a business service.

GetActiveViewProp Method

This is an auxiliary function to retrieve the active view’s properties. It does not take any input arguments.

Arguments

Argument	Description
Business Object	Business object name.
View	Active view name.
Screen	Active screen name.
Thread Applet	Thread applet name.
Is Administration Mode	Y or N.
View Mode Code	An integer representing Siebel view mode: • 0: SalesRep View • 1: Manager View • 2: Personal View • 3: AllView

TryMockMethod Method

This is an advanced auxiliary function for administration of the tool bar button workflow. It does not take any input arguments.

Arguments

Argument	Description
MockMethodReplied	Y or N.

QueueMethod Method

Launch a queue method on an applet in another view. When invoked, the UI navigates to the view specified, and then the Queue method is invoked on the specified applet.

Other input arguments of this method will be cached into the application Shared Global, which can be retried back.

DTU DataTransfer method has built-in integration with the QueueMethod. When the input argument DataMapObject has the format:

```
SharedGlobal: NameofSharedGlobal
```

The data map object name can be retrieved from Shared Global with the name NameofSharedGlobal.

Arguments

Argument	Description
GotoView	Name of the view to go to.
Applet	Applet name.
Method	Queued method to be invoked on the applet.

15 Other Component Business Services for C/OM

Other Component Business Services for C/OM

This chapter describes the Context Service business service, as well as other important C/OM business services. Information is provided as follows:

- *InMemory Upgrade Data Service*
- *SIS OM PMT Service*
- *Context Service Business Service*
- *ISS ATP Service*
- *ISS Credit Card Transaction Service*
- *ISS Credit Check Service*
- *ISS Disable Service*
- *ISS Package Product Service*
- *ISS Payment Profile Service*
- *ISS Promotion Agreement Manager*
- *ISS Promotion CP Admin Service*
- *ISS Promotion Edit UI Service*
- *ISS Promotion Management Service*
- *ISS Revenue Synchronization Service*
- *ISS Sequence Service*
- *ISS Service Product Service*
- *ISS Shipping Calculation Service*
- *ISS Shipping Cost Service*
- *ISS Smart Part Number Generation Service*
- *ISS Spread Discount Service*
- *ISS Tax Calculation Service*
- *ISS Template Service*

Note: This chapter does not address three component business services that are addressed elsewhere in this guide. For information about Data Validation Manager business service, see *Data Validation Manager* For information about ISS Approval Business Service, see *Approvals Manager* For information about ISS Copy Service, see *Copy Service*.

InMemory Upgrade Data Service

InMemory Upgrade Data Service stores document in cache and operates on stored document through provided methods. This business service is used during in-memory upgrade process and parallel configurator view.

It includes the following methods.

AddMissedItems Method

This method adds the missed line items to the target document by taking the Missed Lists obtained in ApplyRulesExternal.

Arguments

Argument	Description
Missed List	[in] Hierarchy

CallPricer Method

This method calculates the price of the stored document (quote, sales order or asset). Pricing Signal Name is configurable, by default the CalculatePrice signal is invoked.

Arguments

Argument	Description
IntObjectName	[in] String
Mode	[in] String
PricingSignal	[in] String
SiebelMessage	[in]/[out] Hierarchy

GetConditionalCharge Method

This method retrieves the cumulative penalty charge for disconnecting stored document.

Argument	Description
Charged Amount	[out] String
Item Id	[in] String

Retrieve Method

This method retrieves the document from the cache for processing.

Argument	Description
Asset Integration Id	[in] String
Due Date	[in] String
HasCustomizableProduct	[in] String
Header Revision	[in]/[out] Hierarchy
Product Id	[out] String
Root Action Code	[out] String
Root Asset Integration Id	[out] String
Root Line Item Id	[out] String

SetConditionalCharge Method

This method updates the cumulative penalty charge for disconnecting stored document.

Argument	Description
Charged Amount	[in] String
Item Id	[in] String

SetSelectedItem Method

This method generates internal map to refer to selected items in the stored document. Not necessarily the entire document is updated. In some cases, only a portion of the document is updated, but the entire document needs to be repriced.

Argument	Description
Field Name	[in] String
ItemIds	[in] String
Type	[in] String

UpdateMessage Method

This method updates the stored document.

Argument	Description
Dirty Flag	[in] String

SIS OM PMT Service

SIS OM PMT Service provides many of the functions to support out-of-the-box order management workflows, and as part of these workflows various OOTB Profile Attributes are set.

It includes the following methods.

Construct Search Spec Method

This methods builds consolidation search specification.

Argument	Description
BC Search Specification	[out] String

Argument	Description
Field Name	[in] String
Promotion Field Name	[in] String
PromotionInstanceIds	[in] String
Type	[in] String

Get Selected Row Count Method

This methods returns number of rows selected in the active buscomp.

Arguments

Argument	Description
RootItemId	[out] String

Context Service Business Service

The Context Service business service has two main functions, as follows:

- The Context Service business service provides the infrastructure for the C/OM-specific invocation mechanism called “Raise Signal” Through use of a signal, you invoke multiple actions (either of a business service or of a workflow) in a certain order. All associated actions are fully configurable by an integrator. A signal can be triggered through UI buttons—with the standard Siebel Invoke Method—or it can be triggered specifically by business services or business components.
- The Context Service business service acts as the data broker for other C/OM modules, such as Pricing, Eligibility, Product Recommendation, Promotion, and so on. Through the variable map APIs (GetRowSetData and SyncRowSetData), Context Service retrieves the current context data, and then constructs input property sets for other business services. After the property sets are processed by other business services, they can be synchronized back to the database through the SyncRowSetData API.

The Context Service business service provides APIs for constructing a property set from the current ordering context and synchronizing changes to that property set back to the source.

It includes the following methods.

GetRowSetData Method

See *Variable Map Methods of the Context Service Business Service*.

SyncRowSetData Method

See *Variable Map Methods of the Context Service Business Service*.

ISS ATP Service

The ISS ATP business service contains methods for the ISS Fulfillment Service.

It includes the following methods.

CSSISSFulfillmentService::SetATPInputArgument Method

This method is called by the signal ATPInquire. It prepares input arguments before calling the ATP business service.

Arguments

Argument	Description
ATPAction	[in] The action to be specified at the line level: Inquire, Reserve, UnReserve.
RowId	[in] The RowId to process
Mode	[in] "Order", "Quote"

Example Arguments

```
[ATP Action ]= "Inquire";
[RowId]= "42-4ZBY1";
[Mode] = "Order";
```

CSSISSFulfillmentService::ATPRunCheck Method

This method provides a business service wrapper function for ATP ASIs.

Arguments

Argument	Description
Id	[in] The header ID for the quote or order.
Line Item Id	[in] If “inquire at line level” is called, this contains the line ID. If “inquire all” is called, then this is empty.
Inbound Integration Object	[in] The inbound internal integration object must be quote-specific or order-specific. This is used for database write.
Outbound Integration Object	[in] The outbound internal integration object must be quote-specific or order-specific. This is used for database query.
ATPAction	[in] The action to be specified at the line level: such as Inquire, Reserve, and so on.

Example Arguments

```
["ATP Action"] = "Inquire";
["Outbound Integration Object"] = "ATP Check Interface Request - Orders";
["Inbound Integration Object"] = "ATP Check Interface Response - Orders";
["Id"] = "99-2AICU";
["Line Item Id"] = "42-4ZBY1";
["Outbound Integration Object1"] = "ATP Check Interface Request - Orders";
["Inbound Integration Object1"] = "ATP Check Interface Response - Orders";
```

ISS Credit Card Transaction Service

The ISS Credit Card Transaction business service performs credit card authorization tasks.

It includes the following methods.

AuthCharge Method

This method authorizes and settles payment for the current payment line item.

Authorization Method

This method authorizes payment for the current payment line item.

Charge Method

This method settles payment for the current payment line item.

Refund Method

This method refunds payment for the current payment line item.

Reverse Method

This method reverses authorization of payment for the current payment line item.

ISS Credit Check Service

The ISS Credit Check business service performs credit status checks and writes status information to the database.

It includes the following methods.

CreditCheckRunCheck Method

This method runs a credit check for the selected order or quote from an external ASI source, then writes the credit status to the database.

Arguments

Argument	Description
Id	[in] The ID of the quote or order.
Inbound Integration Object	[in] The internal inbound integration object for Quote or Order.
Outbound Integration Object	[in] The internal outbound integration object for Quote or Order.
Return Error	[in] A flag for which, when set to “N”, the function will return an OK regardless. (This is used for situations in which even if the credit check errors out, you want to create the order.)

SetCreditCheckResults Method

This method writes the credit status data to the database. If the credit status is empty, it is set to indeterminate.

Arguments

Argument	Description
Credit Status As Of	[in] The date that the status is valid.
Credit Status Code	[in] The status code. A number from 1 to 6 or a value pre-LOV lookup (such as Okay or Indeterminate).
Credit Status Message	[in] A string describing credit status.
Id	[in] The ID of the quote or order.
Return Error	[in] A flag that, when set to “N”, the function will return an OK regardless.

ISS Disable Service

ISS Disable business service is used to disable certain activities when using ISS Copy Service. This service is used in the ReviseCopyQuote signal.

Note: For information about ISS Copy Service, see *Copy Service*.

It includes the following methods.

DisableCopyXAService Method

This method disables the generation of XA attribute copy on the business component.

Arguments

Argument	Description
InstanceName	[in] ISS business component instance name.
BusCompName	[in] Name of the business component inside the ISS instance business object. The BusComp class must be derived from CSSBCOrderMgmtBase.

DisableCheckCanInsert Method

This method skips the CanInsert check on the business component when new records are inserted.

Arguments

Argument	Description
EnableCanInsert	[in] Y or N. When Y, the business component will skip the CanInsert check.
InstanceName	[in] ISS business component instance name.
BusCompName	[in] Name of the business component inside the ISS instance business object. The BusComp class must be derived from CSSBCOrderMgmtBase.

RestoreServiceState Method

This method restores the business component state modified by DisableCopyXAService or SkipCheckCanInsert methods.

Arguments

Argument	Description
InstanceName	[in] ISS business component instance name.
BusCompName	[in] Name of the business component inside the ISS instance business object. The BusComp class must be derived from CSSBCOrderMgmtBase.

ISS Package Product Service

The ISS Package Product business service allows you to collect any number of simple products into a single package or to remove simple products from a package when in the Quote Line Items or Order Line Items views. After selecting one or more related simple products, you can then collect them into one package and treat the package as one line item. Later, you can also separate a packaged collection into its separate pieces and treat the separate pieces as separate line items.

Note: You cannot package customizable products. When you package simple products, they get collected into one product called a Package. You cannot package Packages.

It includes the following methods.

MergeIntoOnePackage Method

This method collects simple products in one package.

RemoveFromPackage Method

This method takes one or more simple products out of a package.

ISS Payment Profile Service

The ISS Payment Profile business service provides the functions to update the existing profile from a Quote or Order, or to create a new payment profile from the current Quote or Order for the current account.

It includes the following methods.

SaveAsPaymentProfile Method

In the Quote or Order screen, Payment, and then the Payment Detail view, click Create Profile to bring up a pop-up applet in which the user can specify a name for the payment profile.

UpdatePaymentProfile Method

In the Quote or Order screen, Payment, and then the Payment Detail view, click Update Profile to update the existing associated profile.

ISS Promotion Agreement Manager

The ISS Promotion Agreement Manager business service provides a set of methods that deal with commitments associated with the promotion process to be implemented in workflows.

This business service is used to check commitment compliance, and to generate agreements, agreement items, and covered assets for promotions that require a commitment from the customer.

It includes the following methods.

CalculateDates Method

This method is used to calculate the Start and End dates for a new agreement.

Arguments

Argument	Description
Root Product Id	[in] The row ID of the promotion record.
Promotion Source Instance Id	[in] Promotion instance upgrade integration ID. This is relevant only in the case of a promotion upgrade. (Optional)
Start Date	[out] The start date of the new agreement.
End Date	[out] The end date of the new agreement.
Effective Date	[out] The effective date of the new agreement.
Old Agreement Id	[out] Row ID of the agreement for the original promotion. This is relevant only for promotion upgrades, when the input argument Promotion Source Instance Id is passed in.
Old Agreement Item Id	[out] Row ID of the agreement line item of the agreement for the original promotion. This is relevant only for promotion upgrades, when the input argument Promotion Source Instance Id is passed in.
Old Promotion Id	[out] Row ID of the original promotion. This is relevant only for promotion upgrades, when the input argument Promotion Source Instance Id is passed in.

CheckCommitmentCompliance Method

The CheckCommitmentCompliance method allows the user to verify commitment compliance on all records in the current document. This method is invoked in the Verify Promotion workflow process. This method returns a property set of promotions that have violated an active agreement.

Arguments

Argument	Description
Active Document Type	[in] The type of document that is currently active, for example, Quote or Order.
Advance To	[in] Date for which the penalty amount is calculated.
Buscomp Name	[in] Name of the business component.
Buscomp Additional SearchSpec	[in] Additional search specification that may be applied to the business component.
Sort Specification	[in] Sort specification for the business component.

Argument	Description
Promotions Violated Flag	[out] A flag (Y or N) to indicate whether there are any promotions that violate an active agreement. "Y" indicates existence of promotions violating active agreements after filtering on the current document is done. "N" indicates absence of promotions violating any active agreements.
Violated Promotions	[out] List of promotion violations, if Promotions Violated flag is 'Y'.

FilterCurrentDocument Method

This method takes as input the property set of violated promotions returned by the CommitmentComplianceCheck method and removes from the property set all promotions that exist in the current document with an action code set to Delete.

Arguments

Argument	Description
Buscomp Name	[in] Name of the business component.
Buscomp Additional SearchSpec	[in] Additional search specification that may be applied to the business component.
Active Document Id	[in] Row ID of the active document, for example, Quote or Order.
Promotions Violated Flag	[out] A flag (Y or N) to indicate whether there are any promotions that violate an active agreement. "Y" indicates existence of promotions violating active agreements after filtering on the current document is done. "N" indicates absence of promotions violating any active agreements.
Violated Promotions	[in] List of promotion violations.
Violated Promotions	[out] List of promotion violations, if Promotions Violated flag is 'Y'

FilterPAC Method

This method assumes that Projected Assets Cache has been initialized. This method queries the Projected Assets Cache based on the search specification passed in as an input argument. If the promotion in the violated promotions list does not exist in the Projected Assets Cache, then assume it is already deleted and remove it from the violated promotions list.

Arguments

Argument	Description
Asset Cache Key	[in] Cache key assuming that Projected Assets Cache has been initialized.
Search Expression	[in] Search specification to be used for querying Projected Assets Cache.
Violated Promotions	[in] List of promotion violations.
Violated Promotions	[out] List of promotion violations, if Promotions Violated flag is 'Y'.

GetPromotionDetails Method

This method is used in workflows to load promotion-related fields from the input SiebelMessage. This method returns a promotion status that is used in the workflows for branching.

Arguments

Argument	Description
SiebelMessage	[in] Contains a single complex Open Order line item.
Account Id	[out] Row ID of the Account associated to the Order.
Asset Integration Id	[out] Asset integration ID that is used to open order items for an asset.
Contact Id	[out] Row ID of the Contact associated with the Order.
Old Promotion Id	[out] Row ID of the existing Promotion that is already an Asset. This is set in the SetOldAssetDetails method. This argument is used in the case of a Promotion Upgrade.
Old Promotion Instance Id	[out] Promotion Instance Integration ID of the old promotion that is already an asset. This is set in the SetOldAssetDetails method. This argument is used in the case of a Promotion Upgrade.
Product Type	[out] Type of the Root Product on the line item. For example, Product or Promotion.
Promotion Id	[out] Row ID of the Promotion associated with the line item.
Promotion Instance Id	[out] Promotion Instance Integration ID of the line item. This indicates the promotion instance with which the line item is associated.

Argument	Description
Promotion Rule Id	[out] Row ID of the promotion component rule that this line item references.
Promotion Source Instance Id	[out] Promotion Instance Upgrade Integration ID on the line item. This is relevant only in the case of a promotion upgrade.
Root Product Id	[out] Row ID of the root product or promotion.
Root Product Name	[out] Name of the root product or promotion.
Status	[out] Promotion Status returned by this method. This argument is used in workflows for branching.

InvokeCopyService Method

The InvokeCopyService method invokes the ISS Copy Service business service to copy the promotion details set up in Promotion Administration—such as Charge Plans, Terms and Conditions, Conditional Charges, Related Assets—to the corresponding Charge Plans, Terms and Conditions, Conditional Charges, Covered Assets in the Agreement that has been created for the corresponding Account for this promotion.

Arguments

Argument	Description
DestBusCompName	[in] Name of the destination business component.
DestBusObjName	[in] Name of the destination business object.
DestinationSearchSpec	[in] Search specification used for the destination business component.
Map Object Name	[in] Name of the data map object that has been set up to do the field-level copy from the source business component to the destination business component.
SourceBusCompName	[in] Name of the source business component.
SourceBusObjName	[in] Name of the source business object.
SourceSearchSpec	[in] Search specification used for the source business component.

SetProfileAttributes Method

This method saves the Start, End, and Effective dates for the new agreement to be created for the promotion in the user profile.

Arguments

Argument	Description
Promotion Start Date	[in] Start date of the promotion agreement.
Promotion End Date	[in] End date of the promotion agreement.
Promotion Effective Date	[in] Effective date of the promotion agreement.

RemoveProfileAttributes Method

This method clears the Start Date, End Date, and Effective Date from the user profile.

SetOldAssetDetails Method

This method is invoked in workflows to maintain the Row Id of the current asset, the Integration Id of the current asset, the Row Id of the promotion associated with the current asset, the Promotion Instance Id of the current asset, and so on, in internal storage, so these IDs can be used for further evaluation by the GetPromotionDetails method.

Arguments

Argument	Description
SiebelMessage	[in] Contains a single complex Open Order line item.

ISS Promotion CP Admin Service

The ISS Promotion CP Admin business service allows you to add more constraints for a customizable product (CP) when the CP is covered by a promotion. In the process of defining a promotion, you can change cardinality or add more domain and attribute constraints for a CP when the CP is covered by a promotion rule. At run time, the constraints will

be checked against the CP when it is covered by the promotion. The CP constraints are cached along with the promotion definition.

Note: When you make changes to the CP constraints under a promotion, you must clear the cached promotion information.

It includes the following methods.

ClearCache Method

This method clears cached promotion information.

Arguments

Argument	Description
Prod Prom Id	[in] String
Prod Prom Rule Id	[in] String
CfgRequest	[out] Hierarchy

GetPromotionConstraints Method

This method retrieves CP constraints for a promotion rule.

ISS Promotion Edit UI Service

The Promotion Edit UI business service provides a specialized user interface that displays promotions or products grouped by each promotion rule. This service helps to generate required data structures and to render this Promotion Edit UI.

This is the entry point function to invoke the Promotion Edit UI session from the product catalog.

Note: The business component that invokes this service must support the AddtoCart operation.

It includes the following methods.

ApplyEditPromotion Method

The ApplyEditPromotion function triggers the AddtoCart operation on the Promotion Selection Catalog business component, then retrieves the LastItemId value from the profile attributes, and continues the rest of the Edit UI rendering operations.

Note: The product type for the selected record must be set to Promotion.

Arguments

Argument	Description
Promotion Def	[in] The Promotion Def Id of the selected promotion record.
Return View	[in] Returns the view name when the user completes an Edit UI session.

EditPromotion Method

This method is an entry function to enter a Promotion Edit UI session.

Note: The product type for the selected record must be set to Promotion.

Arguments

Argument	Description
Promotion Def	[in] The Promotion Def Id of the selected promotion record.
Promotion Instance	[in] The Promotion Instance Id of the selected promotion record.
Return View	[in] Returns the view name when the user completes an Edit UI session.

ISS Promotion Management Service

The ISS Promotion Management business service is used to handle product promotions at run time, such as Apply Promotion, Integrity Check, Recommend Promotion, and so on. This service also provides functions to integrate with Unified Messaging and to support Asset-Based Ordering, such as Load Message and Load Promotion-Related Assets.

It includes the following methods.

ApplyPromotion Method

This method applies the promotion in the current document.

Arguments

Argument	Description
Active Document Id	[in] String
Prod Prom Id	[in] String
Prod Prom Instance Id	[in] String
Qty	[in] String
Target Document	[in] String

ClearAssociation Method

This method dissociates items with a promotion.

Arguments

Argument	Description
Active Document Id	[in] String
Index	[in] String
Prod Prom Instance Id	[in] String
Target Document	[in] String
List	[in] Hierarchy

ClearMessages Method

This method clears previous UMS messages related to promotion.

Argument

Argument	Description
Promotion Messages	[in] String

CollectAssetList Method

This method collects assets selected by the user and assets not selected but covered by selected promotions.

Arguments

Argument	Description
Active Document Id	[in] String
Target Document	[in] String
All Asset List	[out] Hierarchy
Promotion Instance List	[out] Hierarchy
Promotion Num	[out] String
SIS Delete Num	[out] String
SIS Select Num	SIS Delete Num
SIS Select Num	[out] Hierarchy
Unselected Prom Related List	[out] Hierarchy
Unused Selected List	[out] Hierarchy
ItemIds	[in]/[out] String

Argument	Description
PromotionInstanceIds	[out] String

GetContext Method

This method retrieves the current active document type and document ID.

Arguments

Argument	Description
Active Document Id	[out] String
Target Document	[out] String

GetResponseType Method

This method retrieves the user's response type (for example: Accept or Reject) for a UMS message.

Argument

Argument	Description
Message Response	[out] String

InitializePAC Method

This method loads the projected asset for a contact or account based on ABO Type.

Arguments

Argument	Description
ABO Type	[in] String
Account Id	[in] String

Argument	Description
Active Document Id	[in] String
Asset Cache Key	[in] String
Asset Cache Key	[out] String
Contact Id	[in] String
Target Document	[in] String

IntegrityCheck Method

This method executes an integrity check for promotions in the current document. It returns a flag indicating whether there are violations, and if so, a list of all violations.

Arguments

Argument	Description
ABO Type	[in] String
Account Id	[in] String
Active Document Id	[in] String
Contact Id	[in] String
Target Document	[in] String
Integrity Violation Flag	[out] String
Violation List	[out] Hierarchy

LoadMessage Method

This method invokes the UMS business service to display promotion-related UMS messages.

Arguments

Argument	Description
Account Id	[in] String
Active Document Id	[in] String
Charge Amount	[in] String
Commitment End Date	[in] String
Commitment Start Date	[in] String
Contact Id	[in] String
Message Type	[in] String
Prod Prom Id	[in] String
Prod Prom Name	[in] String
Recommendation List	[in] Hierarchy
Target Document	[in] String
Violated Promotions	[in] Hierarchy
Violation List	[in] Hierarchy

LoadPromRelatedAssets Method

This method loads assets covered by a promotion, but not selected by the user.

Arguments

Argument	Description
Account Id	[in] String
List Type	[in] String

Argument	Description
Prod Prom Instance Id	[in] String
Unselected Prom Related List	[out] Hierarchy

MsgResponse Method

This method executes the response actions defined for a UMS message

RecommendPromotion Method

This method recommends promotions to the user based on items in the current document. It returns a flag indicating whether there is any promotion to recommend, and if so, a list of recommended promotions.

Arguments

Argument	Description
Account Id	[in] String
Active Document Id	[in] String
Any Recommendation	[out] String
Asset Cache Key	[in] String
Asset Cache Key	[out] String
Match Percentage	[in] String
Recommendation List	[out] Hierarchy
Target Document	[in] String
Top Number	[in] String

ISS Revenue Synchronization Service

The ISS Revenue Synchronization business service is used to synchronize opportunity products with quote items. The Quote method creates a new quote based on the current opportunity. The UpdateOppty method updates the source opportunity after the quote is modified.

It includes the following methods.

Quote Method

This method is used to implement Auto Quote functionality that generates a quote based on the active opportunity. The quote line items will be created according to the opportunity products for which the Auto Quote flag is checked. The Quote method is invoked by a C/OM signal at Opportunity, Quote buscomp.

UpdateOppty Method

This method updates the opportunity with the current data in the line items of the quote or order. The method is invoked by a C/OM signal.

ISS Sequence Service

The ISS Sequence business service is used to re-sequence all line items with sequential line numbers.

It includes the following method.

Sequence Method

This method re-sequences all line items with sequential line numbers.

ISS Service Product Service

The ISS Service Product business service adds a service product to the Quote, Order, or Agreement header and associates it to a regular product. This means that this service product pertains only to the product to which it is associated.

It includes the following method.

Service Method

This method creates a service (covered) product to cover the selected product.

ISS Shipping Calculation Service

The ISS Shipping Calculation business service calculates the shipping charges for a quote or order based on a combination of factors including source location, destination, shipping carrier, shipping method, and weight.

It includes the following method.

CalculateShippingCost Method

The sole method in this business service performs a look-up of the shipping zone that corresponds to each line item of the quote or order based on the source location, destination, shipping carrier, and shipping method. This result is, in turn, used to perform a look-up of the shipping rate that corresponds to the shipping zone and weight.

ISS Shipping Cost Service

The ISS Shipping Cost business service calculates the shipping charges for an eSales quote or order based on factors including shipping carrier and shipping method.

It includes the following method.

CalculateShippingCost Method

The sole method in this business service uses a customer-defined eScript to look up and calculate the shipping charges.

ISS Smart Part Number Generation Service

The ISS Smart Part Number Generation business service generates the Smart Part Number (SPN) for a product based on attribute values of its product class.

You can define SPNs for a product class using the Administration - Product screen, Product Class, and then the Part Number Definitions view of the run-time client. A product class can have two types of part number definitions: Dynamic and Predefined.

When the user picks a product for a Quote item or for an Order item, the SPN of the chosen product is generated by the ISS Smart Part Number Generation Service. The business service gets pointers for the item business component and for the attributes business component, then it traverses its attributes and saves all attribute name-value pairs into a property set that includes the ID and Integration ID of the product. The service also calls other business services to generate the SPN for the product with the property set, and then it saves the SPN value to the Quote or Order item business component.

It includes the following method.

GeneratePartNumber Method

This method generates the Smart Part Number (SPN) for a product based on the attribute values of its product class.

ISS Spread Discount Service

The ISS Spread Discount business service spreads the discount among selected Quote, Order, or Agreement line items, or among all Quote, Order, or Agreement line items. The upper limit for currency code precision is 6 decimals.

The ISS Spread Discount Service includes the following method.

SpreadDiscount Method

This method specifies the input and output hierarchical property sets. The Spread Discount Driver Workflow Process (Spread Discount step) provides an example of this method's usage.

ISS Tax Calculation Service

The ISS Tax Calculation business service is used to calculate tax for a quote or an order.

It includes the following methods.

TaxCalculation Method

This method prepares the appropriate parameters and invokes the Tax Calculator business service to call third-party TaxWare software.

InternalTaxCalculation Method

This method calculates the tax amount based on the tax rate and total defined from Quote or Order.

ISS Template Service

A favorite is an object that has a structure similar to a quote or an order. The ISS Template business service allows the user to store the current quote or order as a favorite. It can also retrieve all the items or selected items from a favorite to add to the quote or order.

It includes the following methods.

SaveAsTemplate Method

This method allows the user to click on the Save as Favorite menu item in Quote or Order to bring up a pop-up applet that prompts the user to specify a name for the template.

OrderTemplate Method

This method copies the saved favorite items into the current quote or order.

OrderTemplateSelectItems Method

This method copies selected favorite items into the current quote or order.