Siebel CRM Desktop for Microsoft Outlook Administration Guide

Version 3.10, Rev. A

January 2018

ORACLE

Contents

Chapter 4: How Siebel CRM Desktop Handles Siebel CRM Data

Chapter 5: How Siebel CRM Desktop Synchronizes Data

Chapter 6: Installing Siebel CRM Desktop

Chapter 7: Administering Siebel CRM Desktop

Chapter 8: Controlling Synchronization

Chapter 9: Customizing Siebel CRM Desktop

Chapter 10: Customizing Picklists

Chapter 12: Customizing Authentication

Appendix A: Reference Information for Siebel CRM Desktop

Appendix B: How Siebel CRM Desktop Maps Fields Between Siebel CRM Data and Microsoft Outlook Data

Appendix C: XML Files Reference

1 What's New in This Release

What's New in Siebel CRM Desktop for Microsoft Outlook Administration Guide, Version 3.10, Rev. A

This guide has been updated to correct or remove obsolete product and component terms.

2 Overview of Siebel CRM Desktop for Microsoft Outlook

This chapter describes an overview of Oracle's Siebel CRM Desktop for Microsoft Outlook. It includes the following topics:

Benefits of Using Siebel CRM Desktop

Oracle's Siebel CRM Desktop for Microsoft Outlook (Siebel CRM Desktop) is an application that integrates Siebel CRM processes and data in Microsoft Outlook®. It allows the user to do typical CRM work directly in a native Microsoft Outlook environment. It centralizes essential business information in the familiar Microsoft Outlook environment. This centralization complements the existing capabilities that a Siebel Business Application provides. The user can use Siebel CRM Desktop to do the following work:

- **Manage Siebel CRM data.** Manage Siebel CRM data and link this data to CRM records directly in Microsoft Outlook. The user can manage a calendar appointment, emails, contacts, accounts, activities, opportunities, and so on directly in Microsoft Outlook.
- **Synchronize data.** Perform bidirectional, incremental synchronization between Siebel CRM Desktop in the Microsoft Outlook client and the Siebel Server. This synchronization helps to keep data up-to-date and consistent.
- **Work while disconnected.** Perform work even if disconnected from the corporate network.

Your organization can realize the following benefits:

- Increased user adoption of your business processes and tools. Siebel CRM Desktop does not require the user to use an application that is new to the user.
- Increased accessibility to data because the user is not required to log in to a Siebel Business Application to view and maintain CRM data.
- Decreased training costs. Most users are already familiar with Microsoft Outlook. You can focus your training on CRM business processes instead of spending time and resources on learning how to perform software interactions.

Scenarios for Using Siebel CRM Desktop

This topic describes several scenarios of how you can use Siebel CRM Desktop with Microsoft Outlook. It includes the following topics:

The scenarios in this topic give examples of how you might use Siebel CRM Desktop. You might use Siebel CRM Desktop differently, depending on your business model.

Scenario for Working with an Opportunity

On Friday afternoon, a sales manager reviews the Big Deal opportunity and realizes that it has been inactive for some time. The manager does the following work in native Outlook to assign the task to a sales representative:

- Creates a new Outlook task
- Creates a relationship between an opportunity and the task and completes other task details
- Sends the task to the assigned owner

On Monday, the representative uses Outlook to view opportunities. This representative examines the opportunity and notices the new activity and the assignor, and then realizes that the demonstration must be revised. The representative can drill down on the activity record to view more information. On Thursday, the representative finishes revising the demonstration and changes the activity status to Done.

Scenario for Managing Contact Information

A sales representative works at High-Tech Office Expo and manages many customers, including a customer named Company Y. While at High-Tech Office Expo, the sales representative meets a new contact who is the CEO of Company X. This company is a competitor of Company Y. The sales representative also encounters an old college friend and they trade contact information.

The sales representative creates a new contact in Outlook and enters information about the CEO in this new contact record. While creating this contact, the representative links the contact to the existing Company X account. Next, the representative uses a scanner to scan the CEO's business card and then attaches the scanned image to the contact. The representative must share this contact with colleagues, so the representative clicks the Share Bar. Siebel CRM Desktop then marks the contact as shared and toggles the Share Bar to indicate that the contact is shared.

In the same Contacts folder, the sales representative creates a new contact for the old college friend. The default option does not share the contact with the Siebel Server and the contact remains private.

The sales representative must call the VP of Sales at Company Y, who is represented in the Siebel CRM data as another contact. The representative finds the contact record and then finds the cell phone number for the contact. The contact record for the VP displays in Outlook along with all the other contacts that the representative can normally view in the Siebel Web Client. Assume another user at High-Tech Office Expo uses the Siebel Web Client to update the contact information for the VP. When the sales representative synchronizes, Siebel CRM Desktop displays this updated information directly in Outlook.

Scenario for Managing Account Information

Company Z is one of the smaller accounts that a sales representative manages. This company recently relocated and the representative must update the address details that are related to the account. The representative drills down on the account in the accounts view and then enters the new address in the form.

To make sure that everyone on the account team is aware of the new location, the representative sends an email to the account team. The representative uses the Email to Account Team button in Outlook to include all account team members. Siebel CRM Desktop enters email addresses for the entire account team in an email message and then displays the message. The representative types in a brief note about the new location and then sends the email.

While the representative is in the account record for Company Z, the representative decides to add a new contact to the account. To do this, the representative types the contact name in the text box under the Contacts section and then CRM Desktop displays the names of contacts who already exist. The representative chooses one of these contacts and then clicks Add to create a relationship. As an alternative, the representative can click the SalesBook icon and then choose an existing contact or create a new contact.

For more information, see "Controlling Buttons That Send Email Messages and Set Up Meetings" on page 196.

Scenario for Creating a Relationship Between an Email an Opportunity

A sales representative opens an email from an external contact. This email explains that the purchasing director at the external contact changed. The representative knows that this information is important for the Big Deal opportunity, so the representative shares this email with the Siebel Server and then relates it with the opportunity. This work makes sure that the entire sales team who is working on this opportunity is aware that there is a new purchasing director. Siebel CRM Desktop synchronizes this information with the Siebel Server as an activity record that includes a relationship with the opportunity, along with the original email as an activity attachment.

When the representative shares the email with the Siebel Server, the representative can choose the sales data that the email describes. To create a relationship between the Big Deal opportunity and the email, the representative types the name in the Opportunity control. If the external contact is related to at least one opportunity, then CRM Desktop presents a filtered list of values while the representative types the value. The representative chooses the record for the Big Deal opportunity and then CRM Desktop links the email to the opportunity.

Scenario for Managing an Opportunity

After meeting with a contact at a key strategic account, a sales representative learns of an upcoming request for proposal that might help to improve the sales pipeline for the next quarter. To complete this work, the representative uses Siebel CRM Desktop to create a new opportunity in Outlook.

To begin, the representative clicks Opportunity on the toolbar and CRM Desktop opens a new opportunity form. The representative enters details for the new opportunity, including the name, related account, lead quality, sales method, sales stage, close date, and so on. The representative knows two contacts at the account, and that these contacts decide whether to place an order. The representative relates these contacts with the opportunity. The representative can choose one or more products and relate them with the opportunity. The representative can assign expected revenue values for the opportunity to indicate the projected opportunity value and to describe how that value is distributed across related products for the opportunity.

If the representative saves these details in Outlook and then synchronizes with the Siebel Server, then Siebel CRM makes the details available to other users who can access the account and contacts.

Overview for Using This Book

This book uses the following terms, unless noted otherwise:

- The *client* is Microsoft Outlook with the Siebel CRM Desktop plug-in installed.
- A *user* is a person who uses the client.
- The *server* is the Siebel Server.
- An *administrator* is anyone who uses an administrative screen in the client to configure CRM Desktop. The Administration - Server Configuration screen is an example of an administrative screen.

Computer font indicates a value you enter or text that Siebel CRM displays. For example:

```
This is computer font
```

Italic text indicates a variable value. For example, the *n* and the *method_name* in the following format description are variables:

```
Named Method n: method_name
```

The following is an example of this code:

```
Named Method 2: WriteRecord
```

A *predefined object* is an object that comes already defined with Siebel CRM. The objects that Siebel Webtools displays in the Object List Editor immediately after you install the product, but before you make any customizations are predefined objects.

Depending on the software configuration you purchase, your Siebel Business Application might not include all the features that this book describes.

Getting Help From Oracle

If you require help from Oracle for using object types, you can create a service request (SR) on My Oracle Support. Alternatively, you can phone Global Customer Support directly to create a service request or get a status update on your current SR. Support phone numbers are listed on My Oracle Support. You can also contact your Oracle sales representative for Oracle Advanced Customer Services to request assistance from Oracle's Application Expert Services.

3 How Siebel CRM Desktop Works

This chapter describes how Siebel CRM Desktop works. It includes the following topics:

Overview of How Siebel CRM Desktop Works

This topic describes an overview of how Siebel CRM Desktop works. It includes the following topics:

Extensions to the Microsoft Outlook User Interface

Siebel CRM Desktop is a composite application that displays Siebel CRM sales data in Microsoft Outlook. To store and display Siebel CRM data, the Outlook add-in framework deploys CRM Desktop to Outlook and extends the Outlook data model and user interface. Extensions to the Outlook user interface allow the user to display Siebel CRM data. The following are some examples of these extensions:

- Custom toolbar buttons.
- Custom menu items.
- Custom forms that display Siebel CRM data.
- Custom controls that are embedded in Outlook forms that display Siebel CRM data.
- Personalization options dialog box.
- Predefined Siebel CRM views in Outlook folders. For example, the Opportunities by Account view.

Outlook uses these extensions to allow the user to do a variety of work. The following are some examples of work that the user can do:

- Create new Siebel CRM data in Outlook.
- Mark the Outlook item to share with the Siebel Server data and related sales data. As an option, it can share or unshare a calendar appointment, task, contact, or email message.

- View and edit sales data.
- Start a standard Outlook action, such as sending an email or scheduling a meeting in the context of a sales item.

The following are some examples of validation that CRM Desktop can do when the user enters data:

- Make sure the data type is valid for a given field.
- Make sure each required field includes information.
- Make sure some fields are disallowed, depending on the access rules for conditional data.

Infrastructure That Siebel CRM Desktop Uses

Figure 1 illustrates the infrastructure that Siebel CRM Desktop uses.

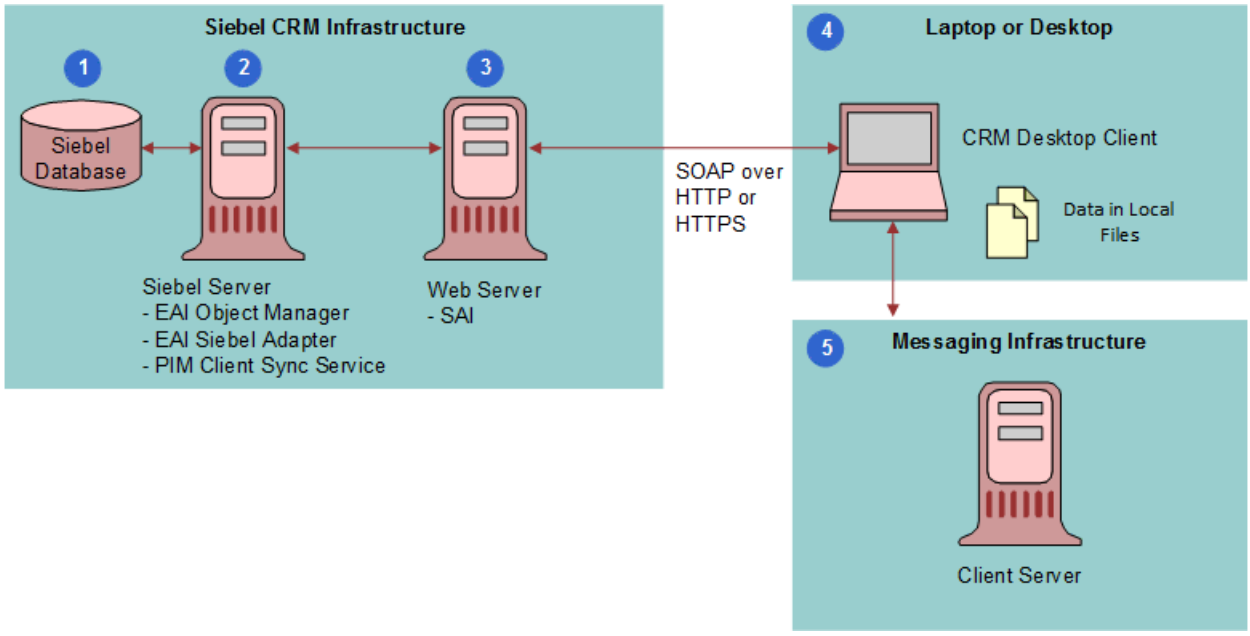


Figure 1. Infrastructure That Siebel CRM Desktop Uses

Explanation of Callouts

The infrastructure that CRM Desktop uses includes the following items:

1 **Siebel Database.** Stores information about opportunities, contacts, service requests, accounts, and so on.

2 **Siebel Server**. Runs the server components for CRM Desktop and manages synchronization sessions with the Siebel Application Interface. Hosts the processes that CRM Desktop requires to support synchronization and handles incoming synchronization requests from the client through the Siebel Application Interface. For more information, see "How Siebel CRM Desktop Uses the Siebel Enterprise" on page 26.

3 **Web Server**. Acts as the Web server that establishes a user session with the Siebel Server. Handles incoming requests to Siebel Web services and helps route the client synchronization request to the proper component in the Siebel CRM infrastructure that supports the Web service. For more information, see "About the Web Service API" on page 28.

4 **Laptop or Desktop.** The computer where CRM Desktop is installed. Oracle implements CRM Desktop as the Outlook add-in. CRM Desktop deploys a binary add-in during installation that supports integration with Outlook, custom user interface capabilities, and the Synchronization Engine. CRM Desktop metadata is available in Outlook after you download and install the customization package from the Siebel Server. Siebel CRM data is available on the client computer after the first synchronization finishes. The customization package includes features that allow synchronization and customization. For more information, see "Customizing the First Run Assistant" on page 89.

5 **Messaging Infrastructure.** Handles local email, calendar items, contacts, and tasks. The messaging infrastructure provides support for mobile access and Web access to information and for storing data. For more information, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25.

Architecture That Siebel CRM Desktop Uses

Figure 2 illustrates the architectural components that Siebel CRM Desktop uses.

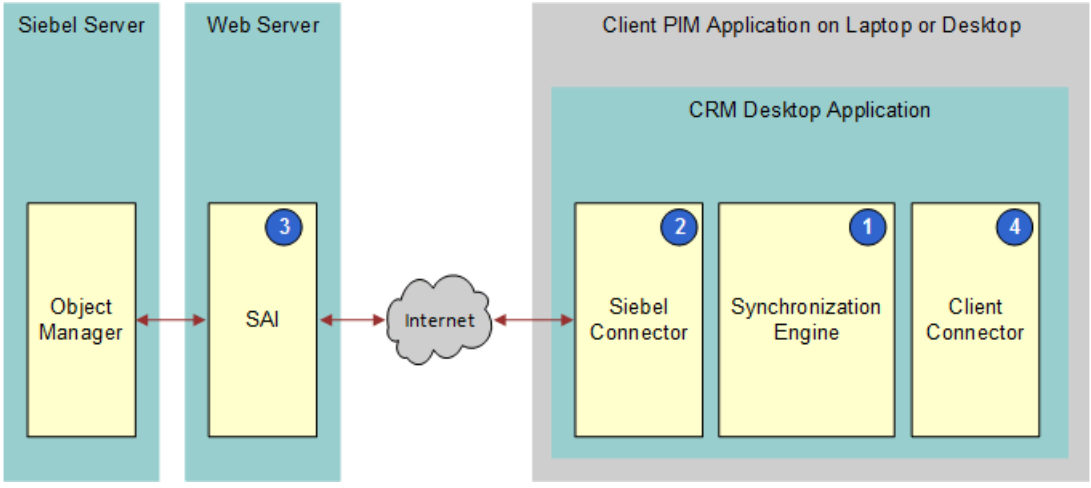


Figure 2. Architecture That Siebel CRM Desktop Uses

Explanation of Callouts

CRM Desktop uses the following architecture components:

1. **Synchronization Engine.** Starts the synchronization process. Determines the changes that CRM Desktop requires to synchronize between the client and the Siebel Server, as determined by the differences between the data sets that are available in each system. To get information from the Siebel Server, it submits requests to the connector and then, to determine the required data changes, it processes the replies. It works with the Outlook connector to make the necessary data changes in the data storage in Outlook.
2. **Siebel Connector.** Connects the personal information manager (PIM) client to the Siebel Server. Submits requests and receives replies and works with the Synchronization Engine. The connector interfaces with the Siebel Server through the Web service infrastructure.
3. **Siebel Application Interface (SAI).** Brokers requests from the Siebel Connector to the Siebel Server.
4. **Client Connector**. Allows the Synchronization Engine to access the data storage in Outlook. Supports queries, inserts, updates, and deletes of data in this data storage.

Overview of How Siebel CRM Desktop Synchronizes Data

Siebel CRM Desktop uses a process that the client controls to synchronize data between Outlook and the Siebel Server. Once installed, the client initializes the Siebel CRM data that is available in Outlook through the first synchronization. An incremental synchronization synchronizes subsequent changes that occur in Outlook or on the Siebel Server.

The user must do the first synchronization with the Siebel Server to make Siebel CRM data available in Outlook. *First Run Assistant* is a wizard that guides the user through the setup of the CRM Desktop add-in in Outlook. It displays when the user starts Outlook for the first time after the CRM Desktop add-in is installed. It starts the first synchronization.

The user can choose among several preferences and then start the first synchronization while using the First Run Assistant. CRM Desktop does the following work:

1. Connects Outlook to the Siebel Server and then authenticates the user.
2. Determines the configuration that the user can access. A relationship with a responsibility that is related to a customization package determines this access. For more information, see "Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files" on page 32.
3. Downloads and applies the configuration.
4. Synchronizes the appropriate data. The connector configuration, synchronization mappings, visibility rules, default internal filters, and default user filters determine this synchronization.

For more information, see "How Siebel CRM Desktop Synchronizes Data Between the Client and the Siebel Server" on page 63.

About Web Service Usage During Synchronization

A component of the Synchronization Engine that you deploy to the client supports synchronization. This component connects to the Siebel Server through the Web service infrastructure. Web services provide access to synchronization for metadata and synchronization for Siebel CRM data. Siebel CRM Desktop provides access to individual objects through the standard Siebel EAI (Enterprise Application Integration) runtime repository objects, such as integration objects and integration components. These objects acquire data through their relations with business objects and business components.

About Siebel CRM Desktop and Microsoft Outlook Data

Microsoft Outlook *data* is data that the user creates in the native Outlook application. Examples include a calendar appointment or task. *Siebel CRM data* is data that can include the following items:

- Business data that the user creates in the CRM Desktop add-in
- Data that a user creates in the client of a Siebel Business Application, such as Siebel Call Center
- Data that resides in the Siebel database on the Siebel Server

Examples of Siebel CRM data include an opportunity, account, or activity. CRM Desktop uses native Outlook data files, so Outlook displays Siebel CRM data through native Outlook user interface elements, such as lists and forms. Outlook can display this data simultaneously with other Outlook data while using the same user interface concept, such as a mailbox folder. The user can choose a folder that displays Siebel CRM data and can also view Outlook data in the Outlook list view.

Siebel CRM Desktop displays Siebel CRM data in the following contexts:

- Outlook **forms.** These forms are extensions to Outlook calendar, contacts, email, and tasks.
- Outlook **items.** For example, the details of an account or opportunity that is related to the Outlook calendar appointment that is shared with CRM Desktop.

When disconnected from the Siebel Server, the user interacts with data that the user can access locally in Outlook.

How Siebel CRM Desktop Stores Siebel CRM Data

Figure 3 illustrates how Siebel CRM Desktop stores Siebel CRM data.

Figure 3. How Siebel CRM Desktop Stores Siebel CRM Data

Explanation of Callouts

Siebel CRM Desktop stores Siebel CRM data in the following way:

1. You install CRM Desktop as a COM add-in with Outlook on the client computer.
2. CRM Desktop stores lookup data, such as lists of values and currencies, and relational data, such as contact and account intersection records, in the local CRM Desktop database, by default. The database element in the siebel_basic_mapping.xml file includes the complete list of object types that the local CRM Desktop database contains. For more information, see "Where Siebel CRM Desktop Stores Data in the File System" on page 83.

3 CRM Desktop stores Siebel CRM data depending on how the user sets the following default email delivery location in the Outlook profile where you install CRM Desktop:

- **POP3 email account.** CRM Desktop stores data in the Outlook Personal Folders (.pst) file.
- **Client database email account in cached mode.** Microsoft Exchange is the client database. It synchronizes data that resides in the .ost file in Outlook with the mailbox that resides on the Microsoft Exchange Server.

This data includes all parent items, such as accounts and opportunities, and native Outlook items, such as contacts, tasks, calendar, email items.

4 Siebel CRM Desktop does not interfere with communication between Outlook and the Microsoft Exchange Server. Outlook synchronizes with the Microsoft Exchange Server just as it does if you do not install Siebel CRM Desktop.

To install and use CRM Desktop, it is not necessary for you to customize the Microsoft Exchange Server or acquire permissions to access it. CRM Desktop does not support a Microsoft Exchange email account that is not cached.

How Siebel CRM Desktop Uses the Siebel Enterprise

This topic describes some of the major components in the Siebel Enterprise that Siebel CRM Desktop uses. It includes the following topics:

For an illustration of how parts of the Siebel Enterprise fit in Siebel CRM Desktop, see “Siebel Enterprise Components That Siebel CRM Desktop Uses” on page 27.

Siebel Enterprise Components That Siebel CRM Desktop Uses

Figure 4 illustrates Siebel Enterprise components that Siebel CRM Desktop uses.

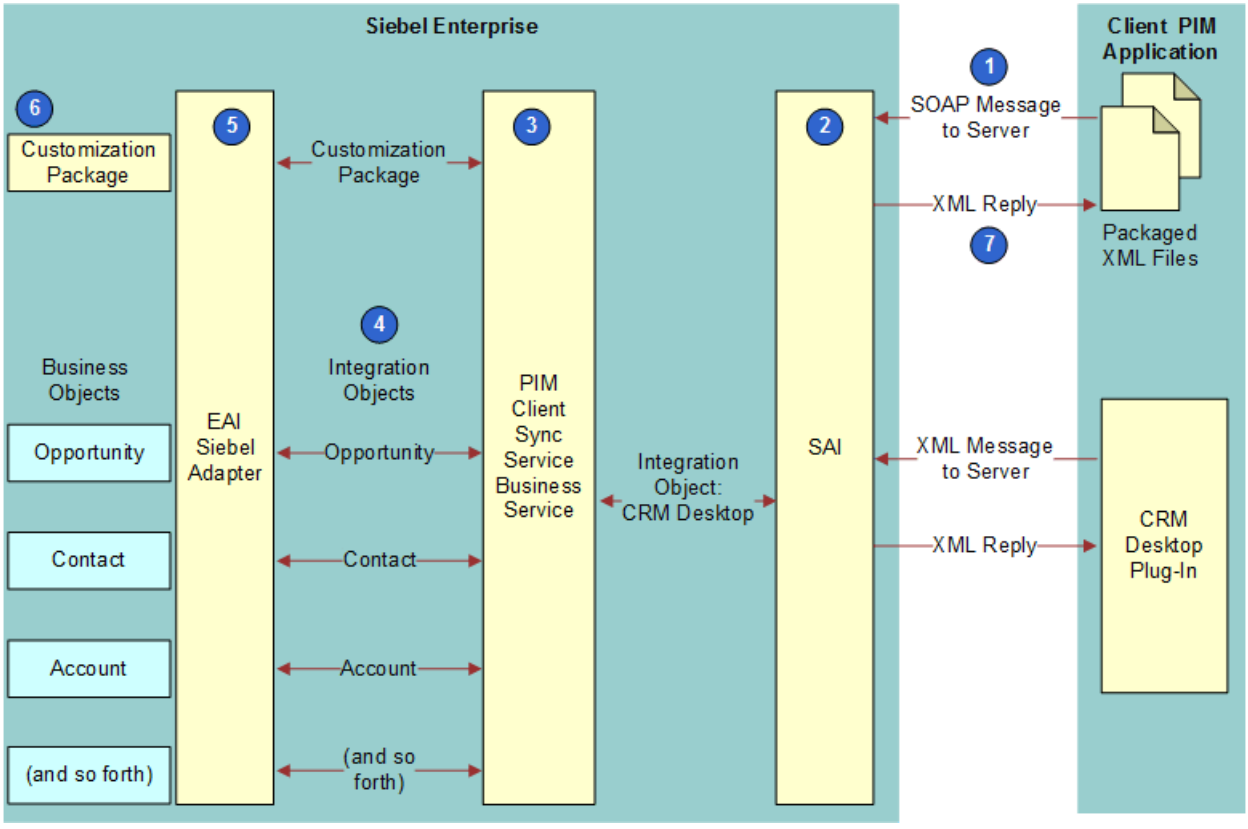


Figure 4. Siebel Enterprise Components That Siebel CRM Desktop Uses

Explanation of Callouts

CRM Desktop uses the following Siebel Enterprise components:

1 **SOAP Message to Server.** To request metadata, sends a SOAP (Simple Object Access Protocol) message to the Siebel Server over HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol Secure). The payload is a Siebel message that it sends in a SOAP envelope.

2 **Siebel Application Interface (SAI).** To handle requests from Outlook for data and metadata synchronization, CRM Desktop does the following:

- Uses the Web Service API as the endpoint of the Web service.
- Accepts SOAP over HTTP or HTTPS from the Synchronization Engine in Outlook.

- Uses the configured endpoint to connect to the EAI Object Manager server component or another server component that provides access to the Web services.

For more information, see "About the Web Service API" on page 28, and *Configuring Siebel Business Applications*.

3 **PIM Client Sync Service Business Service**. Handles the data synchronization request from the Synchronization Engine and then passes this request to the EAI Siebel Adapter business service for processing. This business service supports batching, error handling, and a few custom functions, such as providing record counts and resolving calendar appointment attendees that reside on the Siebel Server. For more information, see "About the PIM Client Sync Service Business Service" on page 29.

4 **Integration Objects**. Requests processes through the EAI Siebel Adapter business service. It uses integration objects to describe the underlying Siebel data structure. It specifies objects and fields that are available for synchronization. For more information, see "About Integration Objects" on page 30.

5 **EAI Siebel Adapter.** Sends requests to the data and metadata synchronization Web services. These services use the Execution method and various operations to process the requests through the EAI Siebel Adapter business service. Example operations include querypage, insert, update, and delete. For more information, see "About the EAI Siebel Adapter Business Service" on page 29.

6 **Customization package.** A collection of XML, JavaScript, DXL, and scoped_helpers.vbs files that describe the CRM Desktop add-in that runs in Outlook. It verifies that the latest application metadata for the user is available on the Siebel Server. For more information, see "About the Customization Package" on page 33.

7 **XML Reply to** Microsoft Outlook. Sends an XML message that contains the metadata that CRM Desktop requests in Step 1. It sends this message to Outlook over HTTP or HTTPS.

For more information, see "How Siebel CRM Desktop Synchronizes Data Between the Client and the Siebel Server" on page 63.

About the Web Service API

The client calls operations in the web service and then the web service calls the related business service. The web service does not itself implement integration objects and business services. Instead, it references them. Outlook communicates with the Web Service API on the Siebel Server. This Web Service API references the PIM Client Sync Service business service. Communication between Outlook and the Siebel Server occurs through SOAP packages that include embedded Siebel messages. The client merely views the web service as an address. The Web Service API includes the following objects:

- Custom integration object for communication between the Siebel Server and Outlook. For more information, see "About Integration Objects" on page 30.
- PIM Client Sync Service business service that supports synchronization with Outlook.
- Integration objects that support Siebel objects.

The business service that the Web Service API references provides the following functionality:

- Performs bulk insert, update, and delete operations in the Siebel database

- Queries and retrieves objects as determined by a given criterion
- Queries and retrieves objects as determined by a set of object IDs
- Returns a count of the number of object records that match a given criterion
- Retrieves list data for object attributes in the Siebel database

About the PIM Client Sync Service Business Service

The PIM Client Sync Service business service delegates calls that Siebel CRM Desktop receives from Outlook and contains methods that handle calls from CRM Desktop. Each method is generic and is not tightly coupled to a specific Siebel object. A Web service provides access to the business service methods. CRM Desktop accesses the Web service through SOAP messages. The business service receives and sends the custom integration object as method parameters of the business service.

The PIM Client Sync Service business service does the following work:

1. Parses the input hierarchy of the object instance.
2. Retrieves commands from the Siebel message. These commands are embedded in the incoming instance of the integration object.
3. Processes the Siebel messages with the help of methods on the EAI Siebel Adapter business service.
4. Embeds the output of the Siebel message into the integration object instance.

   To send the information back to Outlook through the web service interface, the PIM Client Sync Service business service does this step.

For more information, see "About Integration Objects" on page 30.

About the EAI Siebel Adapter Business Service

The *EAI Siebel Adapter* business service is a predefined data interface that interacts with the Siebel Object Manager. It does this to access and modify data in the Siebel database. The following work occurs:

1. The EAI Siebel Adapter business service does the following work:
   a. Takes, as input, an XML document or a property set that conforms to the definition of an integration object in Siebel CRM.
   b. Queries, inserts, updates, deletes, or synchronizes data with the Siebel business object layer.
2. The PIM Client Sync Service web service calls the PIM Client Sync Service business service.
3. The PIM Client Sync Service business service submits requests to the EAI Siebel Adapter to query, insert, update, or delete data in the Siebel database.

For more information, see "About Integration Objects" on page 30.

About Integration Objects

An *integration object* is an object that includes the contents of the messages that Siebel CRM Desktop exchanges between the Siebel Server and Outlook.

Siebel Message Usage with the EAI Siebel Adapter

A *Siebel message* is an instance of an integration object that provides the input to the EAI Siebel Adapter business service. This integration object can carry multiple commands in a single call to the business service. The commands can be grouped together so that the business service can process the commands in a batch. If CRM Desktop sends an integration object instance from Outlook, then Siebel CRM encapsulates this object in an element of the Siebel message. Each of these messages includes a Siebel message header.

Integration Objects That Siebel CRM Desktop Uses

Siebel CRM Desktop uses integration objects that include the following prefix:

```
CRMDesktop
```

You must not create a new integration object for an existing object. Only create a new integration object for a new, custom object that does not already have an integration object. Consider the following examples:

- You must not create a new integration object for accounts. Instead, you can extend the CRMDesktopAccountIO integration object.
- You must create a new integration object for Channel Partner. For more information, see "Creating an Integration Component for the Channel Partner MVG" on page 348.

The following list includes some of the integration objects that allow CRM Desktop to access Siebel CRM data, such as accounts, opportunities, and contacts. Note that this list includes only some integration objects. It does not include all the integration objects that CRM Desktop uses:

- CRMDesktopAccountIO
- CRMDesktopActionIO
- CRMDesktopAssignmentGroupIO
- CRMDesktopBusinessAddressIO
- CRMDesktopContactIO
- CRMDesktopCurrencyIO
- CRMDesktopEmployeeIO
- CRMDesktopIndustryIO
- CRMDesktopInternalDivisionIO
- CRMDesktopInternalProductIO
- CRMDesktopListOfValuesIO
- CRMDesktopOpportunityIO
- CRMDesktopPickListGenericIO
- CRMDesktopPickListHierarchicalIO
- CRMDesktopPositionIO
- CRMDesktopSalesCycleDefIO
- CRMDesktopSystemPreferencesIO
- CRMDesktopUserDetailsIO

CRM Desktop uses the following integration objects to meet other integration requirements:

- **CRMDesktopLocaleIO.** PIM locale setting integration objects that provide access to the locale settings.

- **CRMDesktopSystemPreferencesIO.** An integration object for a PIM system preference that provides access to the system preferences for the Siebel Server.
- **CRMDesktopUserDetailsIO.** For login user data.
- **PIMClientMetaData and PIMClientMetadataFile.** For metadata file download.
- **PIMClientSync.** A wrapper that includes other integration objects.

User Details Business Component

The User Details business component is a clone of the Employee business component except for the Currency, Login Id, and Language fields. Siebel CRM Desktop uses it to retrieve the default user values. Calculated fields in this business component provide access to the Currency, Login Id, and Language fields. The User Details business object references the User Details business component. The CRMDesktopUserDetailsIO integration object allows CRM Desktop to access the following fields in Outlook:

- LoginId
- LoginName
- Currency
- Position
- PositionId
- OrganizationId
- OrganizationName
- Language

About Authentication and Session Management

The Siebel Server provides a lightweight context management facility for Web service authentication. To manage authentication with this facility, CRM Desktop uses a combination of user credentials and a SessionID token. When user credentials are presented in the SOAP header of a Web service request, CRM Desktop performs formal authentication before it runs the Web service operation. If the authentication succeeds, then the operation proceeds and CRM Desktop places a special SessionID token in the SOAP header of the Web service reply.

When Outlook includes the SessionID in subsequent Web service requests, CRM Desktop uses this SessionID to restore cached session information. This configuration bypasses the substantially more expensive process of running the authentication again. If presented with the SessionID and a valid set of user credentials, then CRM Desktop attempts to use the SessionID before it resorts to the user credentials and reauthentication. The session that the SessionID tracks is subject to expiration and other security checks.

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

Metadata That Siebel CRM Desktop Uses

This topic describes the structure of the metadata that Siebel CRM Desktop uses and how you can modify it to support a customization. It includes the following topics:

For more information, see "Overview of Customizing Siebel CRM Desktop" on page 159.

Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files

The responsibility that a customization package references creates a relationship between a Siebel CRM Desktop user and a CRM Desktop configuration. If the package is activated and published, then a user that the responsibility references can download the configuration that the package defines. This configuration is a collection of metadata files that CRM Desktop stores on the Siebel Server and downloads to Outlook during synchronization.

Figure 5 includes an example of how several users, U1, U2, and U3, are related to several responsibilities, R1, R2, R3, R4, and R5, and how these responsibilities are related to several customization packages, P1, P2, and P3.

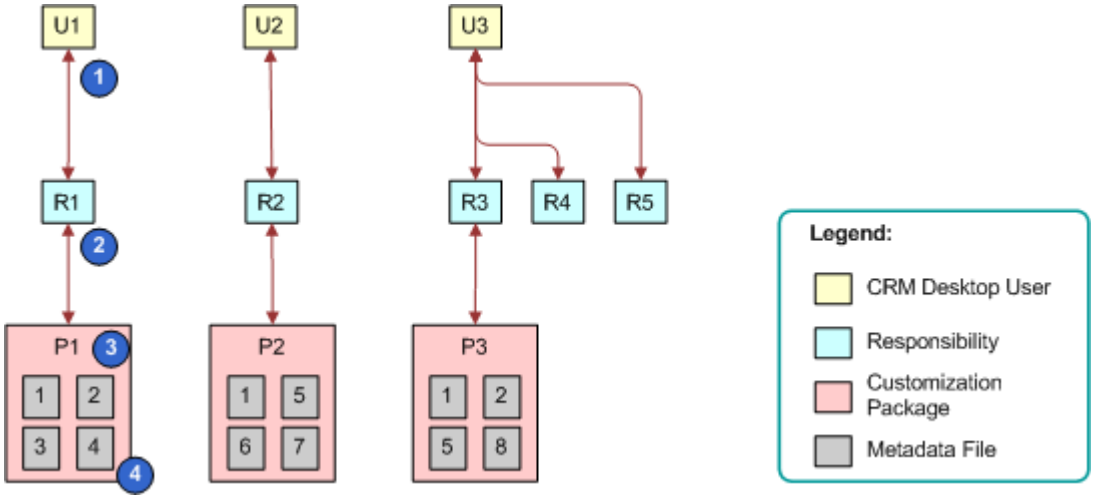


Figure 5. Example of Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files

Explanation of Callouts

The following relationships exist between users, responsibilities, customization packages, and metadata files:

1. **CRM Desktop user**. The user of an implementation of Siebel CRM Desktop.
2. **Responsibility.** A Siebel responsibility, such as Sales Representative. It corresponds to the job role that the user performs.
3. **Customization package.** A collection of metadata files. CRM Desktop creates a relationship between these files and a responsibility. For more information, see "About the Customization Package" on page 33.
4. **Metadata File**. A description of CRM Desktop that CRM Desktop deploys to Outlook as XML code or JavaScript files. For more information, see "About Metadata Files" on page 34.

How Siebel CRM Desktop Allows Users to Access Siebel CRM Data

Siebel CRM Desktop allows the user to access Siebel CRM data in Outlook in the following ways:

- **Through responsibility.** Similar to how a view allows the user to access data in the Siebel Web Client, CRM Desktop uses a responsibility to create a relationship between the user and a customization package. This relationship identifies the application metadata that CRM Desktop sends to the user through metadata synchronization. The metadata defines the objects that CRM Desktop synchronizes with Outlook. For more information, see "Guidelines for Assigning Responsibilities to Customization Packages" on page 79.
- **Through synchronization filters**. The customization package includes metadata files that specify the data access control and the filters to apply when CRM Desktop synchronizes data with the Siebel Server. For example, the default configuration specifies that the user can synchronize accounts, contacts, and opportunities that are related to the sales team that Siebel CRM assigns to the user.

Depending on your business requirements, you can create different customization packages and assign them to different users through responsibilities. You can create different access control and synchronization filters for each customization package to meet individual user requirements.

About the Customization Package

A *customization package* is a collection of XML metadata files and JavaScript files that Siebel CRM Desktop uses with a responsibility. CRM Desktop deploys a customization package when the user synchronizes the metadata. This synchronization identifies the customization package that is available to the user. You can modify the metadata files to customize your deployment. The following items are some examples of the customizations that you can make:

- Add or remove fields that CRM Desktop synchronizes.
- Change the layoutthat CRM Desktop uses to display a custom form in the client.
- Change a control that CRM Desktop deploys to Outlook.
- Change a security rule.

The customization package describes the following information:

- The extensions to the CRM Desktop user interface. This interface includes Outlook views, forms, lookup controls, and toolbars.
- Translated text strings that CRM Desktop uses to create prompts and labels in Outlook.
- Data validation and security logic.
- Descriptions of synchronization preset filters and view modes that CRM Desktop uses during synchronization.
- Criteria that CRM Desktop uses to detect duplicate objects.
- Business logic that JavaScript provides.
- Data mapping between Siebel fields and Outlook fields.

If you change the data model, then the customization package does a complete resynchronization.

For more information, see Chapter 9, "Customizing Siebel CRM Desktop."

About Metadata Files

A *metadata file* is an XMLor JavaScript file that Siebel CRM Desktop uses to display Siebel CRM data and user interface behavior. CRM Desktop uses these files in the following ways:

- **XML files.** Describes the default synchronization objects, synchronization mapping, custom views and forms in Outlook, and so on.
- **JavaScript files.** Describes business logic that CRM Desktop uses for data validation, custom actions that it provides access to in toolbars, and other custom processing that it does in Outlook.

The following items describe how CRM Desktop uses metadata files with a customization package:

- A customization package includes a collection of metadata files. These files describe the entire CRM Desktop add-in that you deploy to Outlook.
- A customization package consists of a set of metadata files.
- CRM Desktop requires that a customization package include a minimum number of files. These files are described in "Files That the Customization Package Contains" on page 434.
- You can use a single metadata file with more than one customization package. These metadata files can be part of another customization package. Figure 5 on page 32 illustrates this relationship where the same metadata file occurs in different packages. For example, packages 1 and 3 include metadata file 2.
- CRM Desktop creates a relationship between a customization package and a single Siebel responsibility. It creates a relationship between the user and this responsibility so that the user can access the customization package and the CRM Desktop configuration that you deploy to Outlook.

For more information, see "Files That the Customization Package Contains" on page 434.

How Siebel CRM Desktop Reuses, Modifies, and Updates Metadata Files

Siebel CRM Desktop creates a relationship between each customization package and a collection of metadata files, and it can use each metadata file with more than one customization package. You cannot modify the metadata file after CRM Desktop creates a relationship between this metadata file and an active deployment package. You can export, modify, and then reimport the metadata file to create a new metadata file. Although multiple customization packages can reference the same metadata files, one or more metadata files typically use different customization packages that include different information.

For example, the Microsoft Outlook Sales Representative responsibility is different and separate from the Microsoft Outlook Sales Manager responsibility. Although you can create a relationship between an existing responsibility and a customization package, it is recommended that you create a new responsibility. This configuration provides you with more control in determining the users that CRM Desktop uses with a customization package.

For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

About Metadata Administration

You use metadata administration to create and manage a customization package. You can do the following work in metadata administration:

- Upload a metadata file to the Siebel Server.
- Create, update, or delete a customization package.
- Display customization packages that are available, including information about a customization package, such as information about child metadata files.
- Create, update, or delete a metadata file.
- Display metadata files and the details about each metadata file.
- Create a relationship between a metadata file and a customization package.
- Download the necessary metadata files through web services for the user.
- Track the expiration of a customization package and files.
- Control user privileges.
- Control user access to a customization package.

For more information about:

- Work you do to administer metadata during installation, see "Administering Metadata Files" on page 78.
- A description of metadata files that you can administer, see "About Metadata Files" on page 34.
- How CRM Desktop synchronizes metadata, see "How Siebel CRM Desktop Synchronizes Data Between the Client and the Siebel Server" on page 63.

4 How Siebel CRM Desktop Handles Siebel CRM Data

This chapter describes how Siebel CRM Desktop handles some types of Siebel CRM data. It includes the following topics:

How Siebel CRM Desktop Handles Activities

This topic describes how Siebel CRM Desktop handles an activity. It includes the following topics:

Overview of How Siebel CRM Desktop Handles Activities

In Siebel CRM, an *activity* is a work item that the user must track or display as an interaction. The following items are examples of activities:

- A To do item
- An email sent to a contact
- A calendar appointment that includes a contact

Siebel CRM can display an activity in the Activities screen or in the Calendar in the client of a Siebel Business Application, such as the Mobile Web Client. The Display In field of the Activities list determines where CRM Desktop displays an activity in Outlook. This field includes the following values:

- Calendar and Activities
- To Do and Activities
- Activities only
- Communication and Activities

The Type field specifies the type of activity. It can contain a wide range of possible values. For example:

- Calendar Appointment
- Field Repair
- Email-Outbound
- Research

Siebel CRM Desktop uses one of the following custom Siebel CRM activity objects to support an activity in Outlook:

- **Calendar item.** This item is a meeting or **calendar appointment**.
- task**.** This item is a To Do. For example:
  - Book a flight
  - Review new proposal
- **Other item.** This item is a record of communication. For example:
  - Correspondence was sent
  - Demonstration was given

Siebel CRM Desktop does not map a Siebel CRM activity to a single native object in Outlook. Instead, it synchronizes an activity from the Siebel Server to the client as a custom activity record rather than as the Outlook task or calendar item. After synchronization, CRM Desktop does the following:

- Creates the Outlook calendar item that matches the calendar item from Siebel CRM
- Creates the Outlook task that matches the task from Siebel CRM

If the activity is a Siebel CRM activity, then CRM Desktop creates a shared calendar appointment in Outlook.

Outlook does not synchronize directly between native Outlook items and records on the Siebel Server, so CRM Desktop uses the Siebel CRM activity as an intermediary between a native Outlook item that resides in the user mailbox and a Siebel CRM activity that resides on the Siebel Server. If the user creates a shared Outlook calendar appointment, email, or task, then CRM Desktop creates another item in Outlook that represents the Siebel CRM activity record in addition to the shared native Outlook item.

To support this configuration, CRM Desktop uses an activity object as a proxy to synchronize all activities, regardless of type. It does the following:

- Parses each activity when it downloads this activity into calendar, email, orother Outlook objects. For example, it parses a calendar item into the Outlook Calendar.
- If the user modifies a native Outlook item, then CRM Desktop modifies a hidden activity object that contains the information that the object requires, such as a description, start time, relations to other objects, and so on. CRM Desktop synchronizes this hidden object. It does not synchronize the native Outlook item. CRM Desktop uses this same configuration for a native Outlook item that the user creates.

How Activities Are Created or Modified

The user can do one of the following to read, update, create, or delete records:

- Use the Add Activity button.
- Use a form for an item in Outlook that includes a relationship with an activity, such as all the activities for an account. This form allows the user to link the activity with a Siebel CRM record in Outlook, such as an account, opportunity, or contact, and to display the link to the corresponding activity.

Siebel CRM Desktop can create an activity for an item in Outlook, such as a calendar appointment, a task, or an email.

How Siebel CRM Desktop Processes Activities

Siebel CRM Desktop uses the following types of objects to process an activity:

- A native Outlook item, such as a calendar appointment, an email, or a task
- A Siebel CRM activity record in Outlook that CRM Desktop synchronizes from the Siebel Server
- A Siebel CRM activity record on the Siebel Server

Figure 6 illustrates the relationships between Outlook items in Outlook, Siebel CRM records in Outlook, and Siebel CRM records on the Siebel Server. Multiple activity types map to the Display In value of the Activities Only list. For example, demos, and so on. For brevity, Figure 6 does not include these types.

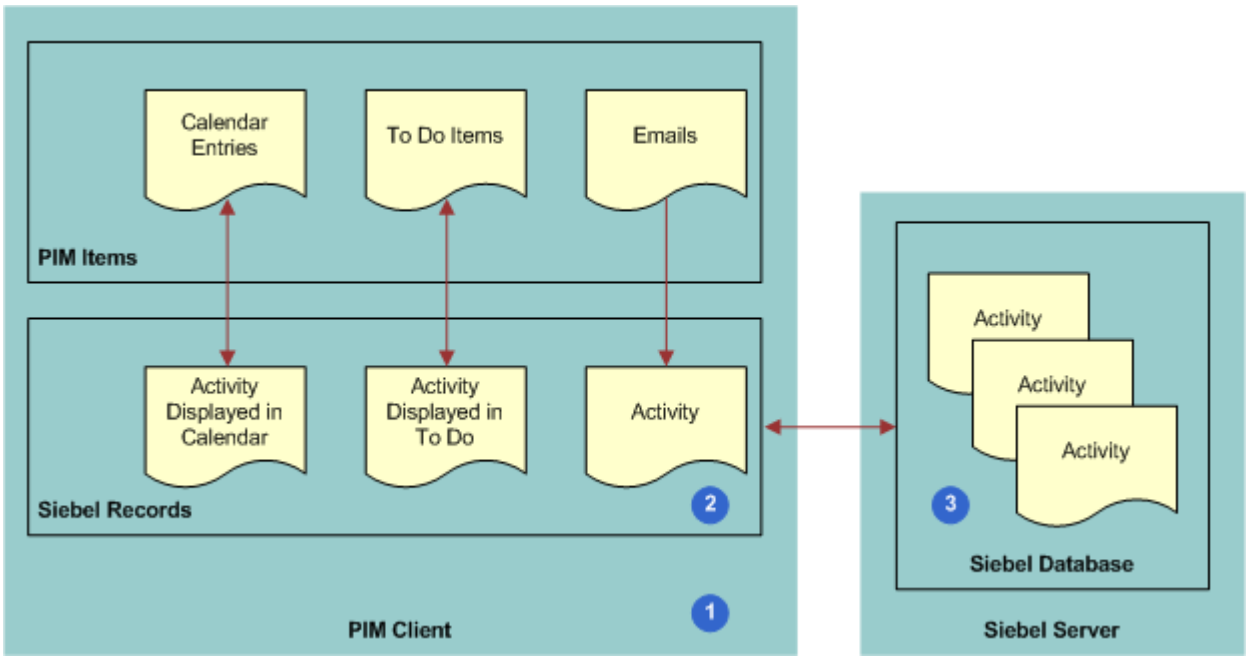


Figure 6. Relationship Between Outlook Items, Siebel Activities in Outlook, and Siebel Activities on the Siebel Server

Explanation of Callouts

Siebel CRM Desktop does the following work to process an activity:

1. An activity is created in Outlook. For more information, see "How Activities Are Created or Modified" on page 39.
2. Siebel CRM Desktop adds a record as a Siebel CRM activity in Outlook.

   If the user marks the Outlook calendar appointment, email, or task as shared, and then saves and closes this item, then CRM Desktop immediately creates the Siebel CRM activity in Outlook.
3. During synchronization, CRM Desktop maps the Siebel CRM activity in Outlook one-to-one with the corresponding activity in the Siebel database on the Siebel Server. If the user shares a new Outlook item or creates an activity in Outlook, then CRM Desktop uploads this activity during synchronization to the Siebel Server and inserts it in the Siebel database. For more information, see "How Siebel CRM Desktop Creates Corresponding Native Microsoft Outlook Items" on page 41.

How Siebel CRM Desktop Creates Corresponding Native Microsoft Outlook Items

When synchronizing data, if the user possesses the rights to view the activity, and if the activity meets the requirements that the filter settings for the user defines, then Siebel CRM Desktop downloads to Outlook any new activities that reside on the Siebel Server.

Table 1 describes how CRM Desktop creates a corresponding native Outlook item depending on the settings of the Display In field.

Table 1. How Siebel CRM Desktop Creates a Corresponding Native Outlook item

Display In Value for the Siebel CRM Activity	Description
Calendar and Activities	CRM Desktop creates a calendar item as a calendar appointment only. If the Siebel Start date is not set for the activity, then CRM Desktop does not create a calendar item in Outlook Calendar. The calendar item is linked to this activity.
To Do and Activities	CRM Desktop creates a native Outlook task that is linked with this activity.
Not Calendar and Activities or To Do and Activities	CRM Desktop synchronizes the activity as visible under the appropriate parent object, such as an Account Activity, Contact Activity, and so on.
Activities Only	If an activity is Activities Only, and if this activity is not related to another item that the user can view, such as an account or contact, then CRM Desktop synchronizes the activity but does not display it in the client. Instead, it stores it in a hidden Activities folder that is not visible to the user.

How Siebel CRM Desktop Resolves Participants and Email Recipients of Activities

This topic describes how Siebel CRM Desktop resolves participant lists and email recipients in the Outlook calendar.

How Siebel CRM Desktop Resolves Meeting Attendees

Siebel CRM Desktop does the following work:

- If the meeting organizer adds an email in the To line, then it creates a relationship with an employee or contact.
- If the meeting organizer uses an MVG (multi-value group) in the meeting form to add a relationship, then it adds the email address to the To line.

- If the meeting organizer uses an MVG in the activity form to add a relationship, then it does not update the To line. This configuration allows the user to create a relationship between the Siebel CRM activity and a contact but not invite the contact to the meeting.

How Siebel CRM Desktop Resolves Owners and Assignees

Siebel CRM Desktop does the following work to resolve task owners and task assignees:

- Resolves assignees in the To field into employees and contacts.
- Does not add the email addresses to the To field if relationships with employees or contacts are made through the Employee or the Contact MVG dialog box for the activity that is linked to a shared task. Creating a relationship with an employee or a contact does not assign the task to this employee or contact.

How Siebel CRM Desktop Resolves Email Recipients

Siebel CRM Desktop does the following work:

- If a user manually shares an email sent to or received from a contact, then it does the following:
  - Resolves the recipients and sender of the email into contacts.
  - Suggests relationships for resolved contacts for the email activity.
  - Suggests a list of accounts and opportunities that are related to the resolved contacts.
  - If the user chooses an account or opportunity, then it creates a relationship between the account or opportunity and the activity that the user creates from the email.
  - If the user manually creates a relationship between the shared email and a Siebel CRM record before the user sends the email, then the automail processing feature does the following work:
    - Preserves the relationships that the user makes.
    - Updates the email activity with contact relationships that CRM Desktop resolves from the email addresses of the recipients.
- If an email activity is created automatically, then it does the following:
  - Resolves the email recipients and sender into contacts.
  - Creates a relationship between these contacts and the email activity.
  - Of these contacts, it creates the following relationships for the first contact it encounters that contains a check mark in the Save Correspondence check box:
    - Sets this contact as the primary for the activity
    - Creates a relationship between the primary account for this contact and the activity
  - If the user sends a shared email, then CRM Desktop does the same processing but it does not resolve the sender as a contact.

- If, to create a relationship for a contact, the user uses the Contact MVG dialog box for an activity that is linked to a shared email, then CRM Desktop does not add email addresses to the To field. A relationship that the user creates with a contact does not update the recipients list for the email message.

How Siebel CRM Desktop Displays Activities in Microsoft Outlook

Siebel CRM Desktop displays data for a Siebel CRM activity in Outlook in the following ways:

- For a shared calendar appointment, email, or task, it displays details of the related activity fields in the native Outlook form. For example, the native Outlook calendar appointment form displays the following information:
  - The description of the Siebel CRM activity in the Subject field of the native Outlook calendar
  - The account that is related to this Siebel CRM activity in the Account field in the extended area of the form

  For example, the user can use the native Outlook form to review and change the account, opportunity, contacts, and employees for the Siebel CRM activity that is related to the shared item.
- As a list of Siebel CRM activity records that are related to a parent sales record. For example, a list of activities that are related to an account or opportunity.

CRM Desktop does not display a folder in the user mailbox for an activity, by default. You can configure the metadata to make this folder visible. For more information, see "Type Tag of the Siebel Basic Mapping File" on page 458.

How Siebel CRM Desktop Sets the Primary Employee of Activities

This topic describes how Siebel CRM Desktop sets the primary employee of an activity for a calendar appointment or a task.

How Siebel CRM Desktop Sets the Primary Employee of a Calendar Appointment

Siebel CRM Desktop sets the Primary Owned By field of a calendar appointment according to the following priority:

1. Resolves the email address of the native Outlook calendar appointment to a Siebel CRM employee. If CRM Desktop finds an employee record that contains this address, then it sets the meeting organizer of the Outlook Calendar event as the primary.

2 If CRM Desktop does not find an employee that contains this address, then it compares this address with addresses from email accounts in the Outlook profile. If it finds a match, then it returns the employee from the employee object. This situation can occur if the email address that is set for the current employee is not the same as the account address in the native Outlook record for this employee.

3 If CRM Desktop does not find a match among the email accounts in the Outlook profile, then no employee is found. In this situation, it sets the primary employee to the value in the Generic Siebel Owner system preference. For more information, see "How Siebel CRM Assigns Meeting Organizers" on page 44.

For more information, see "Controlling How Siebel CRM Desktop Assigns Calendar Appointment Owners" on page 110.

How Siebel CRM Assigns Meeting Organizers

A *Siebel user* is a user who is registered to use Siebel CRM Desktop or a Siebel Business Application, such as Siebel Call Center. The *meeting organizer* is the user who creates the meeting. If a user creates a meeting, then Siebel CRM does the following:

- If the meeting organizer is a Siebel user, then it sets the value in the Owner field of the activity to the following value:

  Meeting Organizer

- If the meeting organizer is not a Siebel user, then it sets the value in the Owner field of the activity to the value that you specify in the Generic Siebel Owner system preference. For more information, see "Controlling How Siebel CRM Desktop Assigns Calendar Appointment Owners" on page 110.

How Siebel CRM Assigns a Meeting Organizer If This Organizer Is Not a Siebel User

Siebel CRM requires the following:

- Every activity must include an owner.
- A Siebel employee record must exist for this owner.

Assume a Siebel user creates a calendar appointment in Outlook. In this situation, an employee record exists for this user, so Siebel CRM Desktop sets this user as the owner and then synchronizes this calendar appointment to the Siebel Server.

An employee record does not exist in the following situations:

- Assume employee A in your organization is not a Siebel user. This employee creates a meeting and then invites another employee in your organization who is a Siebel user to this meeting. A Siebel employee record does not exist for Employee A, and this employee cannot own a Siebel CRM record.
- A contact who is external to your company creates a meeting. A Siebel contact cannot own a meeting.

To create the meeting in this situation, Siebel CRM must first determine the owner for this activity. To avoid duplication errors and access conflicts between users for this meeting, Siebel CRM does the following:

1. Creates a meeting.
2. Assigns the value that you specify in the Generic Siebel Owner system preference as the owner of this meeting.

   For more information, see Resolving Synchronization Conflicts on page 154.

How Siebel CRM Desktop Sets the Primary Employee

Siebel CRM Desktop does following work to set the primary employee for a task:

- If a user creates a shared task that is shared only with the user, then it resolves the user as the owner of the Siebel CRM activity.
- If a user creates a task that is shared and delegated, and if the user keeps a copy of the task in the user mailbox, then it creates an activity and sets the owner according to the following rules:
  - If the user delegates the task only to another employee, then CRM Desktop does the following work:
    - Creates an activity in Outlook for the user
    - Creates a relationship between this employee and the employees team
    - Sets the first employee in the To line as the owner
  - If the user delegates the task of a shared contact to a mixture of shared, unshared, or native Outlook contacts, then CRM Desktop does the following:
    - Creates the activity
    - Sets the user as the owner
    - Creates relationships between all shared contacts that it resolved from email addresses in the task To line. It makes these relationships in the Contacts list.
  - If the user delegates the task to shared contacts and employees, then CRM Desktop does the following work:
    - Sets the first employee in the To line as the owner
    - Creates a relationship between the creator and the Employee team
    - Creates a relationship between contacts and the Contacts list
    - Does not create relationships with other employees

    This configuration helps to avoid having two similar activities for the same employee:
    - For each assigned employee who accepts the task, CRM Desktop creates an activity with this employee, sets the owner, and creates a relationship between the task creator andthe Employees team. It does not create any other relationships.
    - The activity that CRM Desktop creates in Outlook for the first employee in the task To line is the same as the activity that it creates in Outlook for the task creator.
  - If the user delegates the task to an external contact, then CRM Desktop creates an activity and sets the creator as the owner.

- If a user receives and shares a task, then CRM Desktop creates the activity, sets the employee who received the task as the owner, and adds the employee who sent the task to the Employees team as a nonprimary member.

How Siebel CRM Desktop Handles Attachments

Siebel CRM Desktop uses the EAI Siebel Adapter business service to do upload, download, and delete operations on attachments. It does this work in a way that is similar to that of a normal query, update, add, or delete operation.

How Siebel CRM Desktop Handles Shared Activities

This topic describes how CRM Desktop handles a shared activity. It includes the following topics:

Microsoft Outlook supports the concept where relationships can exist between more than one user and the same meeting or task. In this situation, Siebel CRM Desktop prevents the creation of duplicate Siebel activities. It makes sure that it creates a relationship between only one Siebel CRM activity and a single Outlook item that more than one user synchronizes. The first user who synchronizes the Outlook item creates the Siebel CRM activity on the Siebel Server. Siebel CRM Desktop links Outlook items with the Siebel CRM activity for any subsequent user who synchronizes.

To prevent duplicate records, CRM Desktop does the following:

1. Uses the unique identifier for the meeting and task that Outlook provides.
2. Enters information in the CRMD Integration Id field on the Siebel CRM activity with the unique identifier from Step 1.
3. During synchronization, it validates that no other Siebel CRM activity includes this same unique identifier.
4. If it finds that there is no other activity, then it creates a new activity.
5. If it finds that there is another activity, then it downloads the existing activity with the same unique identifier, and then links it with the Outlook item.

For more information, see "How Siebel CRM Desktop Avoids Duplicate Data" on page 73.

How the Origin of an Activity Affects Handling

This topic describes how the origin of an activity affects handling.

How the Origin of an Activity Affects Handling if the Item Originates in Siebel CRM

If an item originates in Siebel CRM, then Siebel CRM creates the activity on the Siebel Server. When Siebel CRM Desktop downloads this activity from the Siebel Server, it creates a native Outlook item if the current user is the owner of the task, or if the current user is in the meeting participant list. CRM Desktop creates the activity as a simple calendar appointment. No additional handling occurs in Outlook.

How the Origin of an Activity Affects Handling if the Item Originates in Microsoft Outlook

If an item originates in Outlook, then Siebel CRM Desktop creates an activity in Microsoft Outlook, uploads it to the Siebel Server during synchronization, and then downloads it to another user during an incremental synchronization. When CRM Desktop downloads this activity from the server, it does not add an item in the Outlook calendar. Instead, it expects Outlook to create the necessary item in the user mailbox. To create this item, Outlook runs the native process that it uses to send a meeting invitation or to assign a task.

If CRM Desktop does this work before it synchronizes the Siebel CRM activity with the Siebel Server, then it links the Siebel CRM activity with the meeting or task and then displays the item in shared mode. In shared mode, the share bar is active and any related details of the Siebel CRM activity display in the extended area of the native Outlook form. If the user shares the item, then the item might not include all the details that the meeting organizer or task owner specified. These details only arrive after CRM Desktop synchronizes the Siebel CRM activity from the Siebel Server.

How Siebel CRM Desktop Handles a Microsoft Outlook Meeting That Includes Multiple Attendees

The example in this topic describes how Siebel CRM Desktop handles a Outlook meeting that includes multiple attendees.

Figure 7 illustrates how user 1, who is a meeting organizer, creates a native calendar item in Microsoft Outlook and shares it. User 2, the invitee, accepts the invitation and also shares it.

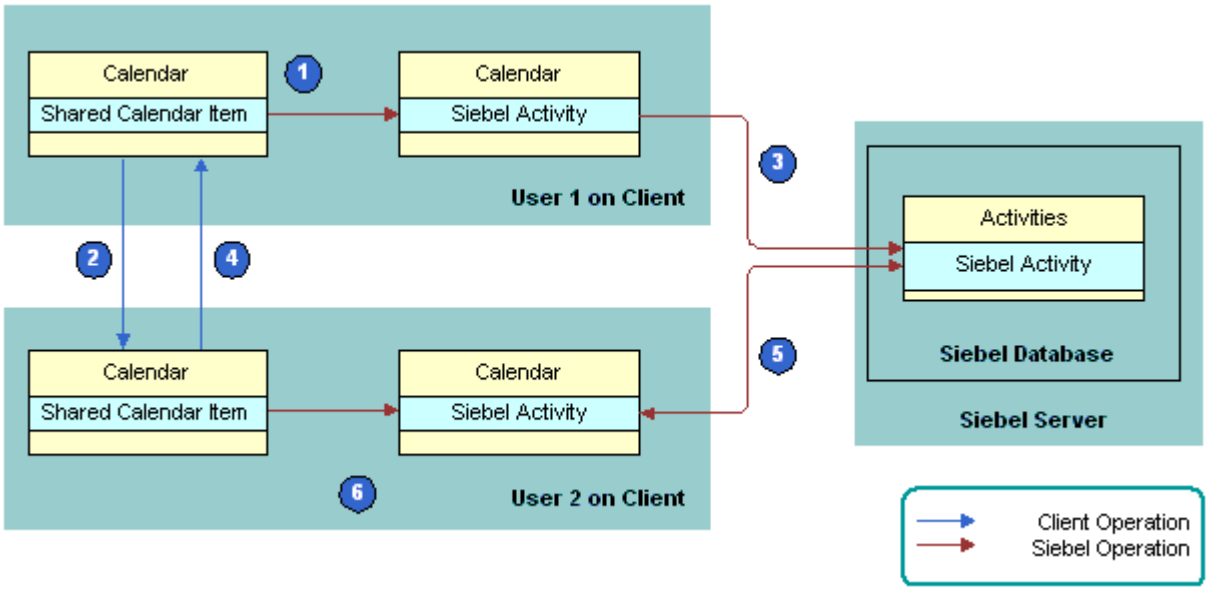


Figure 7. How Siebel CRM Desktop Handles the Outlook Meeting That Contains Multiple Attendees

Explanation of Callouts

The following work occurs:

1. User 1, the meeting organizer, creates a shared meeting with user 2, a meeting attendee who is an employee.
2. User 1 saves the meeting and sends an invitation to user 2. The GlobalObjectIdis the same for the organizer and other meeting attendees.
3. User 1 synchronizes and then Siebel CRM Desktop creates the activity on the Siebel Server.
4. User 2 receives and accepts the invitation.
5. User 2 shares the meeting, saves it, and then synchronizes. Siebel CRM Desktop determines user 1 already synchronized this Outlook meeting with the Siebel Server because these items use the same GlobalObjectIdand include the same meeting organizer. In this situation, Siebel CRM Desktop identifies a duplicate during synchronization.

   Siebel CRM Desktop detects that the activities are equivalent and then does deduplication without displaying the collision dialog box. For more information, see Resolving Synchronization Conflicts on page 154.

How Siebel CRM Desktop Handles a Shared Microsoft Outlook Calendar Appointment That Is Declined

The example in this topic describes how Siebel CRM Desktop handles a Microsoft Outlook calendar appointment that the meeting organizer shares and that the meeting attendee declines. In this situation, the user receives an invitation from another user through Outlook and then declines the invitation:

1. User 1, the meeting organizer, creates a calendar appointment and then shares it and invites user 2, a meeting attendee.
2. User 1 sends an invitation to user 2.
3. User 1 synchronizes. Siebel CRM Desktop uploads the activity to the Siebel Server.
4. User 2 receives the meeting invitation. CRM Desktop creates a shared meeting in the client of User 2. It shares the meeting because the preference for User 2 is to create new native Outlook items as shared. It creates the Siebel CRM activity with User 2 in the employee team.
5. User 2 declines the meeting, with the decline set to send notification to the organizer.
6. CRM Desktop deletes the calendar appointment from the calendar for user 2.
7. If user 2 declines the shared meeting, and if the activity is not synchronized with the Siebel Server, then CRM Desktop deletes the activity in Outlook for user 2. The same situation applies for any meeting participant who unshares or deletes the shared meeting request.
8. User 1 receives the decline notification, CRM Desktop updates Outlook, and removes user 2 from the employee team.
9. User 1 synchronizes. CRM Desktop synchronizes changes with the Siebel Server.
10. User 2 synchronizes.

How Siebel CRM Desktop Handles Microsoft Outlook Calendar Data

This topic describes how Siebel CRM Desktop handles data in the Microsoft Outlook calendar. It includes the following topics:

How Siebel CRM Desktop Handles Microsoft Outlook Calendar Items That Users Save, Change, or Delete

The following behavior applies if the user saves, changes, or deletes an item in the Microsoft Outlook calendar:

- If the user saves a new Outlook calendar item that is shared, then Siebel CRM Desktop creates a new Siebel CRM activity in Outlook.
- If the user changes a calendar appointment in Outlook, and if the user is the owner of the activity, then Siebel CRM changes the activity.
- If the user changes a calendar appointment in Outlook, and if the user is not the owner of the activity, then Siebel CRM does not change the organizer calendar appointment in Outlook. It is not necessary to synchronize the calendar appointment. Siebel CRM Desktop does not update the Siebel CRM activity.
- If the user deletes a calendar appointment from Outlook, and if the user is not the owner of the activity, then CRM Desktop removes the user from the participant list. If the activity is not synchronized with the Siebel Server, then CRM Desktop deletes the activity in Outlook for each participant.
- If the user deletes a calendar appointment from Outlook, and if the user is the owner of the activity, then CRM Desktop removes the activity.
- If the user synchronizes an All Day Calendar event from Outlook with the Siebel Calendar, then the synchronization does the following:
  - Saves the Calendar event with a start time of 12:00 A.M and an end time of 12:00 A.M.
  - Uses start and end date values that the user specifies in Outlook.

  A Calendar event that is set to a single All Day results in a Siebel Calendar entry that includes a start time of 12:00 A.M and an end time of 12:00 A.M.

For more information, see "How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar" on page 441.

How Siebel CRM Desktop Handles Siebel CRM Activities That Users Save, Modify, or Delete

Siebel CRM Desktop internally creates a relationship between a Siebel CRM activity and a calendar appointment or task. CRM Desktop applies the following logic if a user saves, changes, or deletes a Siebel CRM activity:

- If the activity does not include a relationship with an item, then CRM Desktop attempts to find the related Outlook item, and then creates a relationship with the activity.
- If the activity exists in Outlook, then CRM Desktop links it to the corresponding Outlook item.
- If the activity originates as Siebel CRM data, and if CRM Desktop cannot find a correlation, then it creates a new Outlook item and creates a relationship between it and the activity. The type of Outlook item that it creates depends on the following value in the Display In field of the activity:

- If the value in the Display In field is Calendar and Activities, then it creates a calendar appointment.
- If the value in the Display In field is To Do and Activities, then it creates a task.

The mapping that CRM Desktop creates between the Outlook calendar item and the first Siebel CRM activity is the same as that described in "How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar" on page 441, with CRM Desktop doing the following additions:

- Sets the value in the Show Time As field of the native Outlook calendar appointment to Busy
- Sets the calendar appointment label to None

If the user modifies a Siebel CRM activity, then CRM Desktop does the following:

- If the user makes a simple modification, such as modifying the description, Start Date, and so on, then CRM Desktop synchronizes this modification to the CRM Desktop client the same way it synchronizes any other modification.
- If the user modifies a value in the Display In field, then CRM Desktop does a Delete operation and then a Create operation. For example, assume the Display In value is Calendar and Activities for a shared calendar item that the user created in Siebel. If the user modifies this value, then CRM Desktop synchronizes it to the Outlook calendar. If the user uses a Siebel client to modify this value to To Do and Activities, then the user must delete the Outlook Calendar item, and then create an Outlook Task item.

If the user deletes a Siebel CRM activity, the CRM Desktop does one of the following:

- If the record originated in Siebel, then CRM Desktop deletes it from Outlook.
- If the record originated in Outlook, then CRM Desktop unshares it.

How Siebel CRM Desktop Handles a Calendar Appointment

This topic describes how Siebel CRM Desktop handles a calendar appointment.

How Siebel CRM Desktop Correlates Siebel CRM Activities with PIM Data in Microsoft Outlook

When Siebel CRM Desktop synchronizes a Siebel CRM activity to Microsoft Outlook, it attempts to find the PIM data that resides in Outlook that corresponds to the activity. PIM data is a calendar appointment, task, or email. If it finds this item, then it shares it and correlates it with the Siebel CRM activity. CRM Desktop does this correlation for the following items:

- The Outlook calendar item for a Siebel CRM record where the Display In value of the Siebel CRM activity is Calendar and Activities.
- The Outlook task where the Display In value of the Siebel CRM activity is To Do and Activities.
- Each Outlook activity where the Display In value of the Siebel CRM activity is Activities Only.

For example, to do a correlation for the Outlook Calendar event, it uses the following keys:

1. Key 1:
   a. The CRMD Integration Id equals the GlobalObjectIdof the Calendar event.
   b. The owner is the meeting organizer of the Calendar event.
2. Key 2:
   a. The owner is the meeting organizer of the Calendar event.
   b. The description is the subject of the Calendar event.
   c. The value in the Planned field in the Siebel database equals the start time of the Calendar event. This Planned value maps to the value that the Start field in Outlook contains.

If the user creates an activity in CRM Desktop from the Outlook Calendar event, and if the user shares this activity with Siebel CRM, then the CRMD Integration Id field in the activity record in the Siebel database contains a value.

How Siebel CRM Desktop Correlates Data if Siebel CRM Desktop Is Installed

The following sequence describes how Siebel CRM Desktop uses key 1:

1. User 1 does the following work:
   a. Creates a meeting in Microsoft Outlook
   b. Shares this meeting with Siebel CRM
   c. Sends the meeting request to User 2

   In this situation, Siebel CRM Desktop creates a Siebel CRM activity. It uses the value from the GlobalObjectIdfield of the meeting to populate the value in the CRMD Integration Id field in this activity record. This value is a unique value for this meeting. To identify the meeting organizer and the meeting participants, CRM Desktop uses the corresponding values in the Outlook meeting.
2. User 1 synchronizes this activity and Siebel CRM adds it to the Siebel database.
3. Assume user 2 sets the user preference to not automatically create the PIM item that CRM Desktop shares with Siebel CRM. If user 2 receives the meeting invitation from user 1, then CRM Desktop does not share the meeting for this user in the calendar and it does not create a Siebel CRM activity.
4. When User 2 synchronizes, CRM Desktop synchronizes the activity that it added in Step 2 to Outlook. It uses the find_ol_item function to find the Outlook item that corresponds to this activity. It finds the unshared meeting because the following situations are true:
   - This meeting contains the same GlobalObjectId field that the CRMD Integration Id field of the Siebel CRM activity contains.
   - This meeting contains the same meeting organizer that the Activity Owner field of the Siebel CRM activity contains.

If the meeting attendee synchronizes the activity from the Siebel Server before this attendee receives an invitation, and if this attendee sets the preference in the Options dialog box to not share new PIM items, then CRM Desktop uses the find_proxy_item function to find the Siebel CRM activity. If it finds this activity, then it shares the meeting with Siebel CRM.

How Siebel CRM Desktop Correlates Data if Siebel CRM Desktop Is Not Installed

Assume the following situation is true:

- To track activities, a user uses Outlook and a Siebel CRM application, such as Siebel Call Center.
- This user has not installed Siebel CRM Desktop.
- This user enters activities in Outlook and Siebel Call Center.
- The user has an activity in Siebel CRM. The user also has a calendar appointment in Outlook that matches this activity. This activity and this calendar appointment each include the same subject, start date, and activity owner.
- Assume this user installs Siebel CRM Desktop and then synchronizes.

In this situation, CRM Desktop cannot use Key 1 because the Siebel CRM activity does not include a value in the CRMD Integration Id field. If this field does not contain a value, then CRM Desktop uses key 2. The following sequence describes how it uses key 2:

1. The CRMD Integration Id field is empty, so it skips key 1.
2. Correlates the activity:
   a. If the activity is a task, then it uses Key 2 with the following differences:
      - The owner is the meeting organizer of the task.
      - The description is the subject of the task subject.

      For more information, see Step 2 on page 52.

   b. If the activity is an email message, then it uses the following keys:
   - Key 2.2:
     - CRMD Integration Id is the first twenty-two bytes of the PR_CONVERSATION_INDEX email message
     - Owner is the current user.
   - Key 2.3
     - Uses the same information as Key 2.2, plus includes Planned, which is the date that CRM Desktop sends or receives the email. The value in the Planned field in the Siebel database equals the start time of the Calendar event. This Planned value maps to the value the Start field in Outlook.

Note the following:

- If the Display In value of the Siebel CRM activity is To Do and Activities, then the activity is a task.
- If the Display In value of the Siebel CRM activity is Activities Only, then the activity is an email message.

How Siebel CRM Desktop Uses Natural Keys to Identify Duplicate Activities

Siebel CRM Desktop uses natural keys to detect a duplicate between Microsoft Outlook data and Siebel CRM data. It uses the following natural keys for an activity:

- **Activity Owner and CRMD Integration Id.** These items match the GlobalObjectId of a calendar appointment.
- **Activity Owner and Description.** These items match the subject of the calendar appointment and the Start Date.

CRM Desktop uses these keys to query the Siebel database. This query determines if a duplicate exists for this activity in the Siebel database. The following code is an example of the natural keys that Siebel CRM Desktop might use in the first twenty-two bytes of the PR_CONVERSATION_INDEX email messageconnector_configuration.xml file:

```
<natural_keys>
 <natural_key>
  <field>CRMD Integration Id</field>
  <field>Primary Owner Id</field>
 </natural_key>
 <natural_key>
  <field>Description</field>
  <field>Planned</field>
  <field>Primary Owner Id</field>
 </natural_key>
</natural_keys>
```

For more information, see "Files That the Customization Package Contains" on page 434.

How Siebel CRM Desktop Handles a Repeating Calendar Appointment

Microsoft Outlook uses more repeating patterns than the Siebel calendar uses. Siebel CRM Desktop establishes a correlation between Outlook and a Siebel CRM repeating pattern in the following way:

- If the Outlook pattern matches an existing Siebel CRM pattern, then CRM Desktop uses the corresponding repeating pattern to create a Siebel CRM activity.
- If the Outlook pattern does not match an existing Siebel CRM pattern, then CRM Desktop uses a Siebel CRM pattern that occurs more frequently. It also uses more exceptions for an excess occurrence.

For example, assume the user creates a meeting in Outlook that occurs every two weeks for two months for a total of four meeting instances. If CRM Desktop attempts to synchronize this meeting with the Siebel Server, then it cannot directly support the repeating patterns that are available in Siebel CRM. Instead, it does the following work:

- Creates a weekly meeting that lasts for two months for a total of eight meeting instances.
- Creates four exceptions that cancel the intervening weeks.

To remain compatible with Siebel CRM data, CRM Desktop represents each occurrence that changed as a separate calendar appointment in Outlook data.

CRM Desktop does the following work to handle a repeating calendar appointment:

- Maps a repeating Outlook calendar appointment to a repeating Siebel calendar appointment
- Maps a repeating Siebel calendar appointment to a repeating Outlook calendar appointment

Table 2 describes the Siebel fields that CRM Desktop uses to create a repeating calendar appointment.

Table 2. Siebel Fields That Siebel CRM Desktop Uses to Create a Repeating Calendar Appointment

Siebel Field	Description
ExceptionsList	Stores information about exceptions to instances in the repeating series. It is part of the Activity object.
RepeatingType	The frequency of the calendar appointment.
RepeatingExpires	The date of occurrence of the last instance in the series.
Repeating	The flag that indicates a calendar appointment is repeating.

How Siebel CRM Desktop Handles a Single Instance of a Repeating Calendar Appointment

Siebel CRM Desktop handles an exception to a repeating calendar appointment as separate records in Siebel CRM data. For example, to change the time of an instance of a repeating meeting, it creates a separate calendar appointment. It follows standard handling practices for the Siebel calendar so that it handles the calendar appointment series and exceptions in Siebel CRM appropriately.

How Siebel CRM Desktop Handles a Repeating Outlook Calendar Event That Does Not Include an End Date

If a repeating Outlook Calendar event that Siebel CRM Desktop shares with Siebel CRM does not include an end date, and if this repeating pattern:

- Matches a Siebel pattern, then it clears the value in the Repeat Until field of the Siebel CRM Calendar activity.
- Does not match a Siebel repeating pattern, then CRM Desktop limits this repeating pattern to a maximum duration. Table 3 describes the duration that CRM Desktop sets. It does this when it saves the Siebel CRM Calendar activity in Outlook.

Table 3. How Siebel CRM Desktop Handles a Repeating Calendar Event That Does Not Include an End Date

Repeating Pattern of the Outlook Calendar Event	Maximum Duration of Occurrences That CRM Desktop Uses
Daily meetings	1 year
Weekly meetings	1 year
Monthly meetings	2 years
Yearly meetings	5 years

How CRM Desktop Handles Invitee Lists for a Calendar Appointment

The list of invitees in a calendar appointment can contain contacts and employees. To process the email addresses that are specified in the list, Siebel CRM Desktop intercepts the call to the EAI Siebel Adapter business service. If the user creates a shared calendar appointment in Outlook, then the client attempts to resolve the meeting attendees and categorize the attendees as related contacts or employees when the user saves the calendar appointment.

The user might not possess all the contact or employee data that CRM Desktop requires to parse all attendees, so CRM Desktop repeats this process during synchronization. If it makes an insert or update request for a calendar appointment record, then the Siebel Server validates the meeting attendees. If the server detects any changes in the attendee list, then the server returns the updated list to the client and the server updates the contact and employee lists that are related to the calendar appointment.

How Siebel CRM Desktop Handles Invitee Lists for the Update Operation

To handle an invitee list for the update operation, Siebel CRM Desktop does the following work:

1. Updates the input to the EAI Siebel Adapter business service to reflect changes in the invitee list. This update occurs when Siebel CRM Desktop passes the List of Invitees in the Siebel message in the Email To Line field.
2. Marks the contacts and employees that Siebel CRM Desktop removes from the updated calendar appointment in Outlook, and then updates these records in the message.
3. Marks and updates the contacts and employees that Siebel CRM Desktop newly added as insert records, and then updates these records in the message.

How Siebel CRM Desktop Handles Invitee Lists for the Query Operation

To handle an invitee list for the query operation, Siebel CRM Desktop does the following work:

1. It queries the calendar appointment to return the invitee list in the To line of the email. To handle this query, Siebel CRM Desktop processes the output Siebel messages that the EAI Siebel Adapter business service returns.
2. Processes the output from a call to the Query method or the QueryByTemplate method of the EAI Siebel Adapter business service.
3. If the returned record is an activity record with the type as a calendar appointment, then CRM Desktop modifies it in order to add the email addresses of the employees and contacts in the activity to the email To line. It uses a semicolon to separate each email address. It retains the employee list and the contact list.

How Siebel CRM Desktop Handles a Repeating Calendar Appointment

A *repeating* calendar appointment is a calendar appointment that repeats at a specified interval. Siebel CRM Desktop supports synchronizing a repeating calendar appointment between Microsoft Outlook and the Siebel Server, but it handles a repeating pattern differently for the Outlook calendar appointment than it does for a Siebel calendar activity.

Figure 8 illustrates how CRM Desktop handles a repeating calendar appointment.

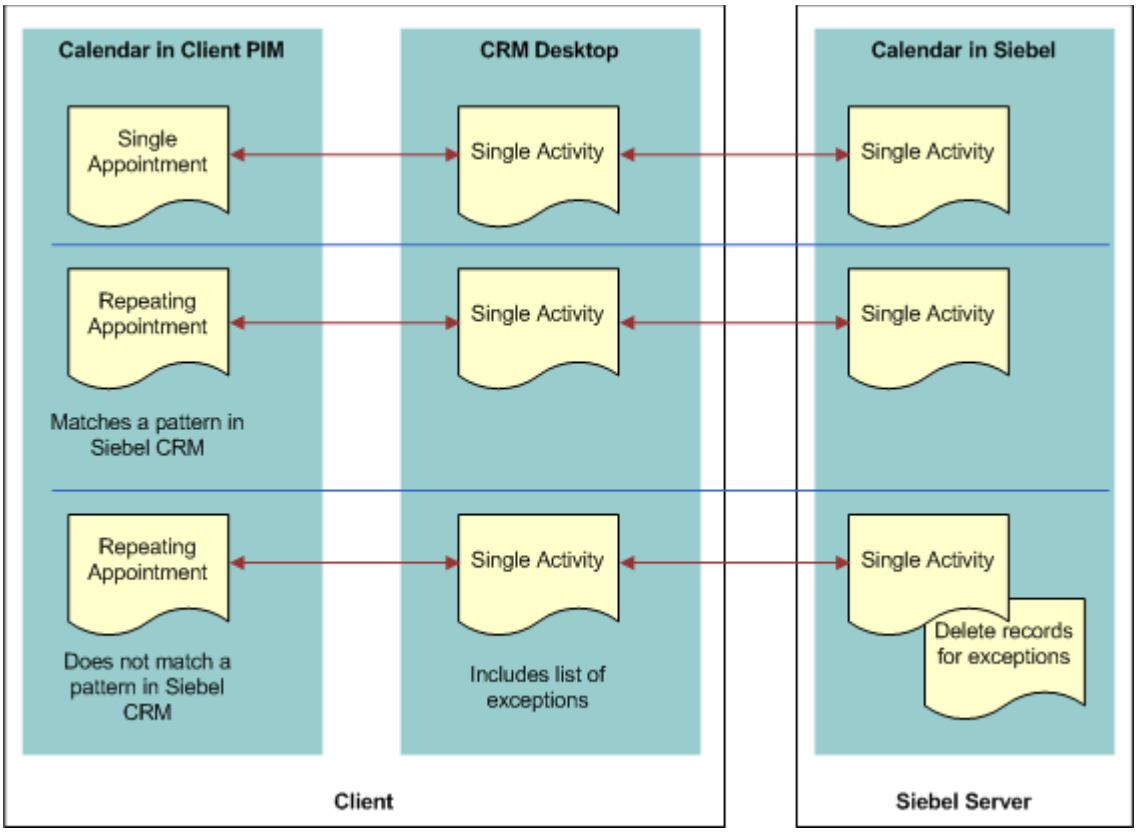


Figure 8. How Siebel CRM Desktop Handles a Repeating Calendar Appointment

For more information, see "How Siebel CRM Desktop Maps Fields Between Siebel CRM Data and Microsoft Outlook Data" on page 441.

How Siebel CRM Desktop Handles Microsoft Outlook tasks

This topic describes how Siebel CRM Desktop handles data for a native Microsoft Outlook task. For more information, see "How Siebel CRM Desktop Maps Fields Between Siebel CRM Data and Microsoft Outlook Data" on page 441.

If the user:

- Saves a new Outlook task that is shared, then CRM Desktop creates a new Siebel CRM activity.
- Changes a native Outlook task that is shared, then CRM Desktop changes the corresponding Siebel CRM activity.
- Deletes a native Outlook task, and if this user:
  - Is the owner of the activity, then CRM Desktop deletes the corresponding Siebel CRM activity.
  - Is not the owner of the activity, then CRM Desktop removes the user from the employee team. It does not delete the corresponding Siebel CRM activity.

How Siebel CRM Desktop Handles Microsoft Outlook Email Messages

Siebel CRM Desktop handles a Microsoft Outlook email message in the following ways:

- Saves the email message as a Siebel CRM activity and sets the activity type to Email - Inbound or Email - Outbound. The customization package specifies the type of activity
- Sets the Display In value to Communications and Activities.
- Creates one Siebel CRM activity for eachOutlook email message that is created.
- Allows the user to link the Siebel CRM activity to a Siebel CRM record. For more information, see "How a User Can Link Siebel CRM Records to Microsoft Outlook Records" on page 60.
- Depending on how the Saving email option is set on the Advanced tab of the Options dialog box, it does one of the following:
  - Adds only the email message or the email message attachments to the Siebel CRM activity.
  - Adds nothing to the Siebel CRM activity. For more information, see "Controlling How Siebel CRM Desktop Handles Email Attachments" on page 111.

If the user deletes the source email message or moves it to a new folder, then CRM Desktop does not change the activity. Deleting or modifying the activity does not affect the source email.

NOTE: If an email is created externally to Outlook (for example, by right-clicking on a file in the Windows Explorer and selecting the Send to option and then Mail recipient option) the Share Bar is not available. Oracle recommends sending the email, then navigating to the Sent Items folder and sharing the email.

For more information, see "How Siebel CRM Desktop Maps Fields Between Siebel CRM Data and Microsoft Outlook Data" on page 441.

How Siebel CRM Desktop Handles Siebel CRM Data with Automatic Email Processing

The user can choose the Save Correspondence option for a shared contact with Siebel CRM data. Siebel CRM Desktop examines the recipients list when it receives an email message. If Siebel CRM Desktop finds an email address that matches one or more contacts with the Save correspondence option chosen, then CRM Desktop creates the corresponding Siebel CRM activity that is related with all contacts that CRM Desktop resolves from the email recipients. It resolves the primary contact first among the contacts that include a check mark in the Save Correspondence check box.

How Siebel CRM Desktop Handles Siebel CRM Data with Manual Email Processing

The user starts manual email processing from the email form. For more information, see "How a User Can Link Siebel CRM Records to Microsoft Outlook Records" on page 60.

How CRM Desktop Displays Data That Is Not Directly Visible

Siebel CRM Desktop synchronizes Siebel CRM account, activity, contact, and opportunity records to the client as complete or incomplete records. It displays incomplete records in the client as read only records. The user cannot edit the information of an incomplete record in a form that displays an incomplete record. The account, contact, and opportunity lists that CRM Desktop displays in an Explorer view include complete Siebel records and incomplete records that Online Lookup pins according to the synchronized records. For more information, see "Controlling How Siebel CRM Desktop Handles Data That Is Not Directly Visible" on page 204and "Controlling the Search in Siebel Button That Does Online Lookup" on page 198.

Complete Records

A *complete record* is a record that matches synchronization filters. Siebel CRM Desktop synchronizes the entire record. The user can view all record information on the record form in the client and an administrator can view it on the Siebel Server. The lower corner of the Filter Records tab displays the number of complete Siebel records that CRM Desktop will synchronize. It does this when you create synchronization filters on the Filter Records tab of the Synchronization Control Panel.

Incomplete Records

An *incomplete record* is a record that does not match the synchronization filter but that Siebel CRM Desktop synchronizes anyway because it is associated with a complete record that does match the filter. An incomplete record is read-only in CRM Desktop regardless of the Siebel visibility that the user possesses for this record. The user cannot edit the record information in the client.

The user can view incomplete records in the association view or in lookup fields on record forms. For example, CRM Desktop displays incomplete:

- Contact records in the Contacts section of the Opportunity form
- Account records in the account lookup in the Account field on the Contact form

To avoid synchronizing the entire Siebel database to the client, CRM Desktop does not synchronize the associations for incomplete records to the client.

How Users Associate Complete and Incomplete Records

A user can associate complete or incomplete records in the client. Siebel CRM Desktop displays these records in SalesBook dialog boxes, such as in the Accounts SalesBook dialog box for the Account field on the Contact form. If the user enters text in a lookup field, then CRM Desktop searches complete and incomplete records and then displays the closest match.

Example of Using Complete and Incomplete Records

Assume the user sets up synchronization filters so that they synchronize the following items:

- All Siebel CRM contacts where the user is on the contact team
- All Opportunities that include a revenue of more $10,000

Assume that Siebel CRM Desktop synchronizes Jackie Driver, a Siebel CRM contact record, to the client as a complete record because it matches the synchronization filter criteria. The user can modify all fields of this record on the contact form in the client. CRM Desktop also synchronizes all opportunities that are related to the contact, including the On-Road Assistance Package opportunity that includes $5,000 in the Revenue field.

If the user opens the On-Road Assistance Package opportunity, then CRM Desktop displays this opportunity in a read-only form that displays the Revenue field. The user can view the revenue but not edit it. CRM Desktop synchronizes this opportunity as an incomplete record because it does not match the synchronization filters.

How a User Can Link Siebel CRM Records to Microsoft Outlook Records

Siebel CRM Desktop allows the user to change linked values. For example, to choose Siebel CRM records to link with the email, the user can use the email form, and then do the following work:

- Use an autocomplete list when the user types characters in a field. For more information, see “Registering Autocomplete Controls” on page 173.
- Use an autocomplete list when the user clicks Contact, Account, or Opportunity on the Extension Bar of the email form.
- Choose an item from a Siebel control on any shared Outlook item:
  - The Siebel control calls the appropriate dialog box that allows the user to choose one or more records.
  - The dialog box supports creating a new record so long as the permissions on the source dialog box allow that operation.

In another example, CRM Desktop allows the user to link a Siebel CRM activity to one of the following Siebel CRM records:

- One account
- One opportunity
- Multiple contacts

How Siebel CRM Desktop Handles Items If the User Removes the CRM Desktop Add-In

If the user removes the CRM Desktop add-in, then Siebel CRM Desktop completely removes all Siebel CRM data. How CRM Desktop handles a shared Outlook item if the user removes the CRM Desktop add-in depends on if the item is Outlook data or Siebel CRM data, and on the type of object. CRM Desktop handles objects in the following ways:

- **Shared** calendar appointment**.** Removes each calendar appointment that originates in Siebel CRM from the Outlook calendar. For the Outlook calendar appointment, it removes any Siebel activities that are related to the Outlook calendar appointment. The calendar appointment no longer displays as shared and no contextual Siebel CRM data is related to the calendar appointment.
- **Shared contact.** CRM Desktop cannot determine if a contact is Outlook data or Siebel CRM data, so it removes all shared contacts from the Outlook .pst or .ost storage file. It is recommended that you back orunshare every contact that the user must preserve before you remove the CRM Desktop add-in.
- **Shared email.** Does not remove an email message that CRM Desktop shares with Siebel CRM. It does remove Siebel activities that are related to a shared email so it no longer displays as shared in Outlook, and so that Outlook does not display any contextual data.
- **Shared** task**.** Handles a task in the same way that it handles a calendar appointment. It removes each task that originates in Siebel CRM from Outlook. It does not remove a native Outlook task. Outlook does not display the task as a shared task and it does not display any Siebel CRM data that is related to the task.

How CRM Desktop Handles Unshared Items if the User Removes Siebel CRM Desktop

If the user removes Siebel CRM Desktop, then an unshared item is not affected. If the user shares an item in Outlook, unshares it, and then synchronizes with the Siebel Server before the user removes Siebel CRM Desktop, then the item is not shared. This item is not affected if the user subsequently removes CRM Desktop. This situation occurs because CRM Desktop only deletes Siebel CRM data and extensions to Outlook that you deploy through CRM Desktop.

5 How Siebel CRM Desktop Synchronizes Data

This chapter describes how Siebel CRM Desktop synchronizes data. It includes the following topics:

For more information, see:

- Overview of How Siebel CRM Desktop Synchronizes Data on page 24
- Chapter 8, “Controlling Synchronization”

How Siebel CRM Desktop Synchronizes Data Between the Client and the Siebel Server

This topic describes how Siebel CRM Desktop synchronizes data between the client and the Siebel Server. It includes the following topics:

How Siebel CRM Desktop Synchronizes Data During the Initial Synchronization

An *initial synchronization* is a type of synchronization that occurs in the following situations:

- Immediately after the user installs Siebel CRM Desktop.
- If you deploy a metadata change to the user that includes a change to the data schema.
- If the options in the login dialog box change. For example, the user name changes or the URL of the Siebel Server changes.

The purpose of the initial synchronization is to initialize the Outlook data storage with the Siebel CRM data that is available to the user. CRM Desktop downloads files in the customization package to Outlook the first time the user synchronizes metadata with the Siebel Server. This metadata includes the following information:

- Definition data for CRM Desktop, such as synchronization rules, object definitions, and so on
- Siebel CRM data for CRM Desktop, such as accounts, opportunities, and so on

CRM Desktop does the following work during the initial synchronization:

1. Establishes a synchronization session with the Siebel Server through the Web service interface.
2. Directs the Web Service Connector in Outlook to call the DownloadMetadataFiles method of the PIM Client Metadata Service business service that resides on the Siebel Server.

   To broker requests through the EAI Siebel Adapter, CRM Desktop uses the business services that the Web services references. It uses the EAI Siebel Adapter business service to process the SiebelMessage payload in the requests.
3. Directs the PIM Client Metadata Service on the Siebel Server to get the login ID of the user from the session, and then identifies the responsibility that the user uses, the customization package that the responsibility references, and if the package is active. For more information, see "Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files" on page 32.
4. If the customization package is published and valid, then CRM Desktop queries all metadata files of that package and creates a Siebel message. It does the following work:
   - Sets the containsFiles argument to true.
   - Enters the relevant data in the packageId, responsibilityId, and hashValue arguments.
5. Applies the downloaded package for Outlook.
6. Downloads Siebel CRM data.

   For more information, see "How Siebel CRM Desktop Synchronizes Siebel CRM Data" on page 67.
7. Logs out of the synchronization session that it established in Step 1.

For more information, see "How Siebel CRM Desktop Handles Synchronization Errors" on page 74.

How Siebel CRM Desktop Synchronizes Data During an Incremental Synchronization

An *incremental synchronization* is a synchronization session that occurs any time after the initial synchronization. To determine the differences that exist in the data that is available to the user, Siebel CRM Desktop compares data in the Microsoft Outlook data storage to data in the Siebel database. It then does the following work:

- Inserts, updates, or deletes data on the Siebel Server according to changes that occurred in Outlook since the prior synchronization

- Inserts, updates, or deletes data in Outlook according to changes that occurred on the Siebel Server since the prior synchronization

CRM Desktop does this work for each difference until it synchronizes all data that resides in the Outlook data storage with data in the Siebel database. In all situations, the user works with data locally in Outlook and CRM Desktop sends these changes to the Siebel Server during an incremental synchronization, but not at the same time that it makes the change in Outlook. Depending on the frequency of the process, a change might not appear on the server immediately.

CRM Desktop does the following work to complete an incremental synchronization:

1. Connects to the Siebel Server to establish a synchronization session.
2. Authenticates the user.
3. Passes the values of the packageId and responsibilityId arguments that it caches in Outlook to the Siebel Server. CRM Desktop cached these values during the prior synchronization. It does this to avoid expensive iterative operations through all responsibilities and customization packages every time it calls the Web service.
4. Receives a reply from the Siebel Server. This reply indicates if new metadata is available for the user.
5. If new metadata is available, then CRM Desktop does the following work:
   - Determines if the customization package changed.
   - If the package changed, then it downloads the new package to a temporary folder in Outlook.
   - If the package is not changed, then it proceeds to Step 11.
6. Determines if the currently applied package is compatible with the downloaded package and then does one of the following:
   - If the package is compatible, then CRM Desktop synchronizes the current data to the Siebel Server.
   - If the package is not compatible, then CRM Desktop stops the synchronization and the user changes are lost. The data modified in the old package is not appropriate for the current version of CRM Desktop.

   For more information, see “How Siebel CRM Desktop Determines Compatibility” on page 75.
7. Determines if the currently applied package is compatible with the current version of CRM Desktop. If the package is not compatible, then CRM Desktop does not apply the downloaded package, it displays a product incompatibility error message, and then exits this process.

   For more information, see “How Siebel CRM Desktop Determines Compatibility” on page 75.
8. If the package is compatible, then CRM Desktop determines if the object structure in the downloaded package changed.
9. If the object structure did not change, then it applies the new package and proceeds to Step 11.
10. If the object structure changed, then it displays a dialog box that asks the user to do one of the following:
    - **Reinstall the object structure.** CRM Desktop does the following:

- Removes the old custom folders.
- Removes the old data.
- Installs the new package.
- Displays the second part of the First Run Assistant. This part allows the user to set synchronization filters and make other settings that set up CRM Desktop to use the new customization package.
- Starts a synchronization session after the user specifies these filters and other settings.

■ **Do not reinstall the object structure.** CRM Desktop does not install the new customization package. The next time the user attempts to synchronize, it displays the same dialog box that asks the user to reinstall the object structure.

11 If the customization packages are identical, then CRM Desktop does the following work:

a Sends a reply that indicates that it is not necessary to download the customization package because the package that resides on the Siebel Server is the same as the package that resides in Outlook.

b Exits this process.

12 Identifies the differences that exist between the data in Outlook and the data on the Siebel Server. To do this, it compares the change key values for all records that are available to the user in Outlook to the change key values that reside on the Siebel Server. The change key includes the record Id and the last time the server updated the record in the Siebel database. The value for the record Id resides in the ROW_ID column of the data table and the value for the time resides in the DB_LAST_UPD column of the data table. Depending on the differences, CRM Desktop changes the values in a data set to make sure the data between Outlook and the server is synchronized. For example, if CRM Desktop detects a new record during synchronization:

■ On the Siebel Server, then it creates a corresponding record in Outlook.

■ In Outlook, then it creates a corresponding record on the Siebel Server.

If the user changes synchronization filters, then CRM Desktop removes the Siebel CRM data that falls outside of the filters from Outlook. It does this during synchronization. A referenced record might remain in Outlook. For example, assume an account references a contact and this account does not match a filter. CRM Desktop continues to synchronize this account but makes it read-only in Outlook. It synchronizes the account details but it does not synchronize any account relationships that exist to other records.

13 Downloads Siebel CRM data.

For more information, see "How Siebel CRM Desktop Synchronizes Siebel CRM Data" on page 67.

The user can now view the newly downloaded data.

14 Logs out of the synchronization session that it established in Step 1.

How Siebel CRM Desktop Handles Changes to Login Credentials

If the user name or the server URL changes, then Siebel CRM Desktop reinitializes the data structure. It does this to remove any personal user data that might exist and to allow the user to synchronize data. Before CRM Desktop begins the reinitialization, it displays a warning to the user that any data that is not synchronized might be lost. If the user agrees to proceed, then the following occurs:

1. CRM Desktop removes the current customization.
2. The user logs in with new credentials.
3. CRM Desktop downloads the package from the Siebel Server and then starts the First Run Assistant.

How Siebel CRM Desktop Synchronizes Siebel CRM Data

Siebel CRM Desktop does the following work to synchronize Siebel CRM data, such as opportunities and accounts:

1. Calculates the number of records for each type of record, such as opportunities or accounts.
2. Gets the values of the change keys for all synchronization objects that are enabled, such as opportunities or accounts. For example, the record ID and the last updated date in the Siebel database.
3. Compares the set of IDs and timestamps in Microsoft Outlook to the set of IDs and timestamps on the Siebel Server to do the following:
   - Identify differences that exist between the data sets for inserts, updates, and deletes.
   - Identify conflicts and create a log entry in the synchronization conflict list for any conflicts.
4. For each difference, CRM Desktop does one of the following operations in Outlook or on the Siebel Server:
   - **Siebel insert.** Query the Siebel database to get the details of the new record and then insert the appropriate item in Outlook.
   - **Siebel update.** Query the Siebel database to get the details of the updated record and then update the appropriate item in Outlook. Note that a Siebel update overwrites all fields in the corresponding Outlook item, not just the updated fields.
   - **Siebel delete.** Delete the appropriate item in Outlook.
   - Microsoft Outlook **insert.** Use the user key that is defined in the metadata to query the Siebel database and then do one of the following:
     - If it does not find a match, then it inserts the appropriate record in the Siebel database and then queries the Siebel database to get the record ID and timestamp.
     - If it does find a match, then it returns a *synchronization issue,* which is an error that occurs during synchronization.
   - Microsoft Outlook **update.** Use the user key that is defined in the metadata to query the Siebel database, and then do one of the following:

- If it does not find an update for the modification number of the record, then it updates the appropriate record in the Siebel database and then queries the Siebel database to get the record Id and updated timestamp.
- If it does find an update for the modification number, then it returns a synchronization issue. Note that this handling is different than in the situation where CRM Desktop changes the same record in the Siebel database or when it compares IDs and timestamps. In this situation, CRM Desktop makes the change in the Siebel database during the actual update operation.

- Microsoft Outlook **delete.** Delete the appropriate record in the Siebel database.

5 If a conflict occurs, then CRM Desktop does the following work:

a Updates the synchronization issues and conflicts log on the client.

b Prompts the user to choose the changes to keep in each of the following situations:

- Update the record in Outlook and on the Siebel Server.
- Update the record in one data set and delete the record in the other data set.

6 Repeats Step 4 for each additional Siebel CRM data object that requires synchronization.

How Siebel CRM Desktop Manages Synchronization Duration

Several factors determine the duration of a synchronization, such as the amount of data that is available to the user, network bandwidth, server performance, client performance, and so on. To shorten this duration, you or the user can do the following:

- You can modify the application configuration. For more information, see "Controlling Synchronization" on page 131.
- The user can adjust settings through the synchronization filter dialog box. For more information, see "How Filters Reduce the Data That Siebel CRM Desktop Synchronizes" on page 71.

The duration of an incremental synchronization session is typically shorter than for an initial synchronization because CRM Desktop downloads all objects during an initial synchronization but during an incremental synchronization it only downloads the objects that changed since the last synchronization.

Situations Where Siebel CRM Desktop Reinstalls the Data Structure

Siebel CRM Desktop reinstalls the data structure in any of the following situations:

- The package update for the user involves a data schema change.
- The user logs in as a different user.

- There is a problem with the data structure. For example, assume the user deletes the Opportunities folder and then removes this deletion from the Deleted Items folder. If the user restarts Microsoft Outlook, then CRM Desktop does the following:
  - Informs the user that a problem with the data structure exists.
  - Removes the data structure.
  - Installs a new data structure.

If CRM Desktop must reinstall the data structure, then it does the following work:

1. Removes all Siebel CRM data, such as accounts, opportunities, shared contacts, and activities.
2. Removes every shared calendar appointment and task that originates in Siebel CRM. Each shared calendar appointment and task that originates in Outlook remain in Outlook.
3. Removes the custom data structure that it previously deployed to Outlook data storage. For example, to remove all custom folders in the user mailbox.
4. Installs the new data structure.

To reenter the appropriate Siebel CRM data in the Outlook data storage, the user must manually start a new, initial synchronization session.

The Customization Package Changed

During synchronization, Siebel CRM Desktop determines if the customization package for the user who is currently logged in changed in such a way that it must reinstall the data structure. The following changes in the data structure of the customization package can cause this situation:

- An object is added to or deleted from the mapping scheme.
- A field is added to an existing object or an existing field is modified.

If the customization package changed, and if CRM Desktop must reinstall the data structure, then it displays a prompt that is similar to the following:

```
A new configuration is available. Are you ready to download and apply it? Selecting
"Yes" will remove your current data, re-install the data structure, and download the
data again.
```

The Customization Package Changed But the Data Structure Has Not Changed

If Siebel CRM Desktop determines during synchronization that the customization package for the user who is currently logged in has changed in such a way that there is no change to the data structure, then it downloads and installs the new package and informs this user about this download. A modification to a security rule is an example of where the package changed but the data structure has not changed. In this situation, CRM Desktop does not start a new, initial synchronization.

How Siebel CRM Desktop Prevents Data Loss if the User Deletes Customization Package Files

If the user deletes the customization package, CRM Desktop restores the customization package from local storage the next time the user starts Microsoft Outlook. For more information about this local storage, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25.

Affect of a Connectivity Failure

An internet or network connectivity failure that occurs during synchronization can interrupt the synchronization. An interruption does not cause data loss or corruption. Synchronization can proceed from the last step that CRM Desktop ran successfully before the interruption.

Factors That Determine the Data That Siebel CRM Desktop Synchronizes

A Siebel user can typically access only a subset of data that is available in the Siebel database. This topic describes that factors that determine the data that a user can access. How you configure CRM Desktop determines many aspects of the data that it synchronizes. For example:

- Synchronization objects that are configured
- Internal filters that are applied
- View modes that are configured on each object
- Security and other configuration that exists on the Siebel Server

You specify this configuration before you deploy CRM Desktop to your users. The user can choose presets for a predefined filter and specify personal filters in the First Run Assistant. The internal filters and server application metadata configuration restricts access to some data, and the user filters apply a second layer of filtering. CRM Desktop applies these filters during initial synchronization and incremental synchronization.

How Filters Reduce the Data That Siebel CRM Desktop Synchronizes

Figure 9 illustrates how the number of Siebel CRM records that are available in the client reduces as these records encounter each set of filters.

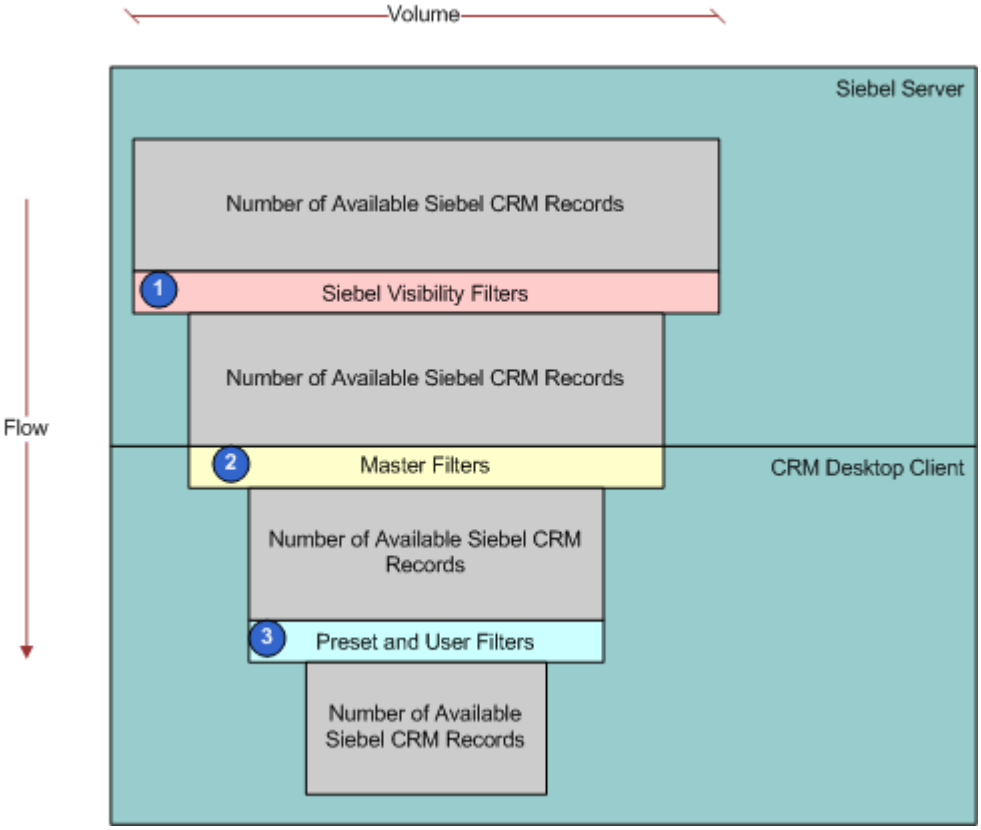


Figure 9. How Filters Reduce the Data That Siebel CRM Desktop Synchronizes

Explanation of Callouts

The following filters reduce the data that CRM Desktop synchronizes:

1. **Siebel visibility filters.** Visibility rules that are configured in the Siebel Runtime Repository and that the Siebel Server applies affects data access. CRM Desktop integrates with the Siebel Server through the Web service interface, so security, search specifications, and other logic that is configured at the integration or business object layer limits the data that CRM Desktop synchronizes to the client. The user interface configuration does not affect the results of queries or other operations that CRM Desktop performs.
2. **Master filters.** Internal synchronization filters that an administrator sets. They identify the Siebel CRM data that CRM Desktop synchronizes to the client. Search specifications on the Siebel Server and security settings in the Siebel Runtime Repository establish the first level of filtering. A set of filters that reside on the client can also restrict the data that CRM Desktop downloads to the client.

3 **Preset and user filters.** An administrator can create preset filters in the customization package and then specify the filter that CRM Desktop applies as the default filter. The user can use this default filter or choose another preset filter. The user can do this in the First Run Assistant during installation or later in the Filter Records Tab of the Synchronization Control Panel. To create a preset filter, the user can modify an existing preset filter. The user can apply different saved presets at different times, depending on the filter requirement. CRM Desktop uses these filters and the application configuration to identify the data to synchronize.

Depending on relationships in the data, CRM Desktop might synchronize an object that the Filter Records Tab disables for synchronization. For example, if the opportunity object is enabled but the account object is not enabled, then it still downloads any account data that the opportunity references. This download is required to make sure the data is complete. Also, CRM Desktop might still upload changes that the user makes in the client to the Siebel Server even if an object or synchronization filter is disabled. For example, if the user disables the account object and then creates an account in Outlook, then it uploads the account to the Siebel Server. For more information, see the following topics:

- Customizing How the First Run Assistant Performs the Initial Synchronization on page 93
- Controlling Synchronization Filters on page 131

Objects That Are Enabled for Synchronization

A set of objects that are enabled for synchronization determines the data that CRM Desktop can synchronize, depending on the configuration that CRM Desktop downloads for the user. These objects are defined in the application metadata that you deploy through the customization package that is available to the user. If the application metadata does not define an object, then CRM Desktop does not synchronize it. Application metadata also defines the field mappings that CRM Desktop uses in the synchronization. These mappings specify how CRM Desktop synchronizes objects in Outlook and on the Siebel Server. For more information, see “Customizing Field Mapping” on page 160.

NOTE: Oracle recommends that you do not drag CRM Desktop Objects, such as Accounts, into non-native folders such as Contacts, because they will not be synchronized by CRM Desktop. Each folder type has its own attributes. If you move an Account to any other folder, CRM Desktop assumes the Account object has been deleted. For more information about how CRM Desktop handles deletions, see “Controlling How Siebel CRM Desktop Adds Deleted Items to the Exclusion List” on page 208.

How Differences Between Microsoft Outlook and the Siebel Server Affect Synchronization

Siebel CRM Desktop downloads to Microsoft Outlook all data that resides on the Siebel Server that is available to the user for the initial synchronization. For an incremental synchronization, the changes that occur to data in Outlook and on the Siebel Server play a large role in determining the data that CRM Desktop synchronizes. The following changes can occur:

- Data is created, updated, or deleted in Outlook.
- Data is created, updated, or deleted on the Siebel Server.

For more information, see “How Siebel CRM Desktop Synchronizes Data During an Incremental Synchronization” on page 64.

Differences in Data Access Rules

Differences in data access rules that occur from one synchronization to the next can occur for the following reasons:

- The user downloaded a different customization package with a different configuration of synchronization objects, view modes, or internal synchronization filters.
- The configuration of the Siebel Runtime Repository changed. This can include security logic, search specifications, or other logic in the integration or business object layers.

How Siebel CRM Desktop Handles Synchronization Duplicates and Errors

This topic describes how Siebel CRM Desktop handles synchronization duplicates and errors. It includes the following topics:

How Siebel CRM Desktop Avoids Duplicate Data

Siebel CRM Desktop includes metadata in the client and configuration in the Siebel Runtime Repository that prevents it from creating duplicate data. This configuration is in addition to the following items:

- The standard user keys that reside in the Siebel database.

  Data deduplication that you can deploy to prevent duplicate data. For more information, see Resolving Synchronization Conflicts on page 154.

- The view mode that CRM Desktop uses for duplicate requests during synchronization for an object type. For more information, see "Controlling the View Mode During Synchronization According to Object Type" on page 149.

CRM Desktop uses Siebel integration objects to create the data structures that are available to synchronize with Outlook. These objects support the user key definition that is the first additional layer of duplicate prevention. For information about how to configure user keys for integration objects and how the EAI Siebel Adapter uses them, see *Overview: Siebel Enterprise Application Integration*.

CRM Desktop also supports user key configuration in the metadata for the client. If it detects the Outlook insert during synchronization, then it queries the synchronization object in the Siebel database with the user key to determine if any records exist that match the record that it is inserting. If it:

- **Does not find a match.** It proceeds with the insert operation.
- **Finds a match.** It raises a synchronization issue that prevents the insert. For more information, see "Resolving Synchronization Conflicts" on page 154.

How Siebel CRM Desktop Handles Synchronization Errors

If a top-level error occurs during synchronization, then the Synchronization Engine stops any further processing and displays a message to the user that describes the error. The following types of errors can occur:

- System error
- Resource allocation error
- General storage problem
- Application state malfunction
- Login failure
- Connectivity problem
- Missing xml or js files in the customization package

If an operation failure occurs in the Synchronization Engine, then CRM Desktop creates a synchronization issue and then attempts to do this operation again during the next synchronization session. The following types of errors can occur in this situation:

- Unexpected failure during an add, update, or delete operation.
- If the data for an object changed since CRM Desktop queried this object during the current synchronization session, then it cannot do the update and the delete operations until the next synchronization cycle. In this situation, it creates an issue and then handles this issue in the subsequent synchronization. This synchronization creates a collision because the object changed since the last synchronization. A *collision* is a data integrity problem that occurs if CRM Desktop modifies the same record on the Siebel Server and in Outlook between synchronization sessions. For more information, see “Resolving Synchronization Conflicts” on page 154.

CRM Desktop logs synchronization errors in synchronization log files and in the General Log log file. It does not enable log files, by default. For more information, see “Log Files You Can Use with Siebel CRM Desktop” on page 119.

How Siebel CRM Desktop Handles Errors While Downloading the Customization Package

If an error occurs while CRM Desktop downloads the customization package, then it displays an error message near the taskbar. This error notifies the user that the customization package changed but CRM Desktop cannot download it because of errors. The problem might be due to the fact that the user does not possess the privilege that the Siebel Server requires to download the package. In this situation, the user must contact the system administrator to get the necessary privileges and then get the customization package again.

How Siebel CRM Desktop Determines Compatibility

This topic describes how Siebel CRM Desktop determines compatibility. For a description of the work CRM Desktop does depending on compatibility, see "How Siebel CRM Desktop Synchronizes Data During an Incremental Synchronization" on page 64.

How Siebel CRM Desktop Determines Product and Version Compatibility

Siebel CRM Desktop uses the following element in the info.xml file to determine product and version compatibility:

```
<compatibility>
  <products>versions</products>
  <schemas>versions</schemas>
</compatibility>
```

where:

- **products**. Specifies product versions that are compatible with the package that CRM Desktop must install.
- **schemas.** Specifies package versions that are compatible with the package that CRM Desktop must install. If the current package is compatible with the new package that it must install, then CRM Desktop synchronizes the local data to the Siebel Server before it applies the new package.
- *versions* is a string that includes one or more version numbers. A dash (-) specifies a range of versions. A semi-colon (;) separates individual version numbers.

For example, the following code specifies that all product versions starting with version 3.05.15.00 through version 3.05.30.99 are compatible:

```
<compatibility preferred_product="3.05.30.00">
  <products>3.05.15.00-3.05.30.99</products>
  <schemas>3.04.00.00-3.05.30.99</schemas>
</compatibility>>
```

CRM Desktop returns a preferred version in the following situations:

- The product is not compatible.
- The product is compatible but the preferred version is not the same version as the current product version.

How Siebel CRM Desktop Determines Schema Compatibility

The schema subelement of the compatibility element in the info.xml file determines schema compatibility. If Siebel CRM Desktop can save the data that it creates or modifies in the old customization package, then the schema is compatible. It saves this data to the current version of the Siebel database. An example of schema incompatibility occurs if a required field in Siebel CRM does not contain a value because the old package does not require it.

How Siebel CRM Desktop Determines Object Structure Compatibility

Siebel CRM Desktop examines the object structure to determine compatibility. For example, if you create a new object type that Siebel CRM Desktop synchronizes, then it must reinstall the current folder structure and install the new folder structure with the new package. If the object structure that the new customization package defines is not different from the object structure that the old package defines, then CRM Desktop applies changes from the new customization package.

How Siebel CRM Desktop Handles Incompatible Customization Packages

If the current version of Siebel CRM Desktop is not compatible with the downloaded customization package, then CRM Desktop does not apply the package. Instead, it displays an error message that notifies the user and then adds an entry in the CRMDesktop*n*.log file, where *n* is an incremental number that uniquely identifies the log file name. It stores the error message in the most recent log file.

6 Installing Siebel CRM Desktop

This chapter describes how to install Siebel CRM Desktop. It includes the following topics:

Roadmap for Installing Siebel CRM Desktop

To install Siebel CRM Desktop, you do the following:

1. Process of Preparing the Siebel Server on page 77
2. Process of Installing the CRM Desktop Add-In on page 85

For information about ACR installation instructions, see *Siebel Maintenance Release Guide* on My Oracle Support.

Process of Preparing the Siebel Server

This process is a step in "Roadmap for Installing Siebel CRM Desktop" on page 77.

To prepare the Siebel Server for Siebel CRM Desktop, you do the following:

1. Preparing the Implementation Environment for Siebel CRM Desktop on page 77
2. Administering Metadata Files on page 78
3. Creating and Publishing the Customization Package on page 78
4. Administering Server Variables on page 80

Preparing the Implementation Environment for Siebel CRM Desktop

This task is a step in "Process of Preparing the Siebel Server" on page 77.

To prepare the implementation environment for Siebel CRM Desktop

- For more information about CRM Desktop supported environments, see the Certifications tab on My Oracle Support.

NOTE: For Siebel CRM product releases 8.1.1.9 and later and for 8.2.2.2 and later, the system requirements and supported platform certifications are available from the Certifications tab on My Oracle Support. For information about Certifications, see article Using Certifications on My Oracle Support for Siebel CRM Products (Doc ID 1492194.1) on My Oracle Support.

Administering Metadata Files

This task is a step in "Process of Preparing the Siebel Server" on page 77.

This topic describes how to administer predefined metadata files for Siebel CRM Desktop. The client uses these files to determine the data to synchronize and the validation rules to apply.

To administer metadata files

1. (Optional) Configure CRM Desktop to store object types in Outlook storage instead of in the local CRM Desktop database.

   For more information, see "Storing Siebel Object Types in Microsoft Outlook Storage" on page 98.

2. Create a .zip file containing all the metadata files described in Table 38 and Table 39.

   Make sure that every file is in the root directory of the zip file. This zip file must not contain a subdirectory.

3. Navigate to the Administration - CRM Desktop screen and then the Metadata Files view.
4. In the Metadata Files list, click New.
5. In the Type list, choose Outlook Metadata Package.
6. In the File Name field, locate the file you created in Step 2.

Creating and Publishing the Customization Package

This task is a step in "Process of Preparing the Siebel Server" on page 77.

You create a relationship between a responsibility and a customization package that determines the information that is available to the user. You can publish a package when CRM Desktop finishes the updates and it is ready to download this package to the client. Publishing makes a package *read only* so that you cannot make any more modifications on the package. For more information, see "Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files" on page 32.

To create and publish the customization package

1. Navigate to the Administration - CRM Desktop screen, and then the Packages view.

2 Create a new customization package, using values from the following table.

Field	Value
Package Name	Enter any value. It is recommended that you use a name that describes the purpose of the package configuration. For example, EMEA Sales Rep, or Field Sales Rep.
Responsibility	Choose the responsibility that is appropriate for the group of users that CRM Desktop uses with the package. Do not assign a user to more than one package. It is recommended that you maintain a separate set of CRM Desktop responsibilities where you can control the user assignment. This configuration helps to prevent creating relationships between a user and more than one responsibility and more than one package. If necessary, before you do this step, you can create a new responsibility and then assign specific users to this responsibility. For more information, see "Guidelines for Assigning Responsibilities to Customization Packages" on page 79.

3 In the Metadata Files list, click Add, locate the .zip file that you added in Step 2 on page 78, and then click OK.

4 In the Packages list, click the link in the Package Name field for the customization package you created in Step 2.

5 In the Package Details form, click Publish.

Siebel CRM changes the Status field of the Package Details form to Published.

For more information about how CRM Desktop uses a customization package, see "Relationships Between Users, Responsibilities, Customization Packages, and Metadata Files" on page 32.

Guidelines for Assigning Responsibilities to Customization Packages

If you develop a customization package, then you must make sure that you assign the user to only one customization package. Note the following guidelines:

- You must assign only one responsibility to a customization package.
- You must not assign more than one responsibility to a customization package.
- You can assign multiple responsibilities to a user but you can create a relationship between only one of these responsibilities with an active customization package.
- Make sure the responsibility and customization package that Siebel CRM Desktop assigns to the user is unique. For example, if CRM Desktop assigns two different responsibilities and two different customization packages to the same user, then a conflict might occur and your customizations might fail.

Republishing Customization Packages

If you change a metadata file, then you must republish the customization package that references this file. Siebel CRM Desktop downloads the changed metadata files as new metadata file records in the package.

To republish a customization package, it is recommended that you unpublish the old package and then create a new package. This allows you to make sure the new package works as expected. If necessary, you can adjust the new package until it works correctly. To revert to the old package, you can unpublish the new package and then publish the old package.

To republish a customization package

1 Unpublish the old customization package:

   a With administrator privileges, log in to a Siebel CRM client that is connected to the Siebel Server.

   b Navigate to the Administration - CRM Desktop screen and then the Packages view.

   c Query the Package Name field of the Packages list for the package you must republish.

   d In the Packages list, click the link in the Package Name field.

   e In the Package Details form, click Unpublish.

   f Make sure Siebel CRM changes the Status field of the Package Details form to Unpublished.

2 Create and publish a new customization package.

   For more information, see "Creating and Publishing the Customization Package" on page 78.

Administering Server Variables

This task is a step in "Process of Preparing the Siebel Server" on page 77.

To administer server variables

1 Set the maximum page size:

   a Log in to a Siebel CRM client that is connected to the Siebel Server.

   b Navigate to the Administration - Server Configuration screen and then the Servers view.

   c In the Components list, query the Component field for EAI Object Manager.

   d In the last applet, click the Parameters tab and then query the Parameter field for Maximum Page Size.

e In the Component Parameters list, configure the Maximum Page Size parameter using values from the following table.

Field	Value
Default Value	1000
Value on Restart	1000

f In the Components list, click Manual Start.

2 Set the DSMaxFetchArraySize parameter:

a Navigate to the Administration - Server Configuration screen and then the Enterprises view.

b Query the Profile field of the Profile Configuration view for Server Datasource.

c In the Profile Parameters list, click Advanced Profile Parameters, query the Alias field for DSMaxFetchArraySize, and then make sure the Value is set to the following:

```
-1
```

3 (Optional) Administer the Generic Siebel Owner system preference.

For more information, see "Controlling How Siebel CRM Desktop Assigns Calendar Appointment Owners" on page 110.

4 Stop and then restart the Siebel Server.

Overview of Installing the CRM Desktop Add-In

This topic describes an overview of installing the CRM Desktop add-in. It includes the following topics:

An *installation package* is a package that contains a Windows Installer (msi) file. Siebel CRM Desktop provides you with this file, and you can use it to install the CRM Desktop add-in on the client computer. It includes the following data:

- The installation information for the CRM Desktop add-in
- The predefined resource files and images for all languages that CRM Desktop supports

You can deploy CRM Desktop through third-party deployment software that you choose. You can use the distribution criteria in these products to distribute software to any group of users, operating systems, domains, workgroups, and so on. System Center Configuration Manager (SCCM) from Microsoft is an example of deployment software. To deploy the CRM Desktop add-in to multiple users, you can use deployment software to create a collection and then distribute the distribution package. A *collection* is the list of users, computers, workgroups or domains where you must distribute the software.

You can use third-party deployment software to do an installation in the background or to do a removal that uses the default installation parameters. With some deployment software, you can specify various installation parameters.

CRM Desktop displays the First Run Assistant after you complete the installation and the user starts Microsoft Outlook. For more information, see "Customizing the First Run Assistant" on page 89.

For more information about using Systems Management Server, see the documentation at the Microsoft TechNet website.

NOTE: Siebel CRM Desktop cannot co-exist with the Siebel Server Sync for Microsoft Exchange Server Add-in.

About Files, File Locations, and Profiles

Siebel CRM Desktop handles files and profiles according to the following conditions:

- CRM Desktop uses a specific profile and a default location for email delivery to store Siebel CRM data. For a description of the locations that you can use for the default email delivery for Microsoft Outlook, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25. For more information, see "Caution About Changing the Default Mail Delivery Location" on page 82.
- During installation, you choose the installation directory where the installer installs the CRM Desktop add-in. For more information, see "Using the Windows Command Line to Set Optional Parameters" on page 100.
- If you configure multiple Outlook profile in Outlook, then First Run Assistant applies the configuration for the CRM Desktop add-in only in one of these profiles. You cannot use it with more than one Outlook profile.
- If you install and then remove the Outlook profile, then the CRM Desktop add-in remains in an installed state and continues to display in the Add and Remove Programs dialog box. In this situation, the user can remove the CRM Desktop add-in.
- If you remove the .pst file from the Outlook profile where you install CRM Desktop, then you can still remove the CRM Desktop add-in. For more information, see "Caution About Changing the Default Mail Delivery Location" on page 82.

Caution About Changing the Default Mail Delivery Location

You must not change the default email delivery location in the Microsoft Outlook profile where you install Siebel CRM Desktop.

CAUTION: If you change the default email delivery location then Siebel CRM Desktop will not function correctly. Oracle does not support changing this location.

Installing an Add-in Profile That Includes a Microsoft Exchange Mail Account

You can install the CRM Desktop add-in in the Outlook profile that uses a Microsoft Exchange mail account even if an additional mailbox is available. Prior to Siebel CRM Desktop version 3.5, you cannot install the CRM Desktop add-in to the Outlook profile that uses a Microsoft Exchange mail account if an additional mailbox is available.

Changes That Siebel CRM Desktop Makes During Installation

This topic describes changes that Siebel CRM Desktop makes during installation. It makes these changes to the file system, Windows Registry, and settings in Microsoft Outlook.

Where Siebel CRM Desktop Stores Data in the File System

Siebel CRM Desktop places most files that it requires in the following folder:

```
%APPDATA%\Oracle\CRM Desktop\Profile\
```

where:

- *APPDATA* is an environment variable that the operating system automatically sets.

For example, CRM Desktop places most files that it requires in the following folder:

```
C:\Users\username\AppData\Roaming\Oracle\CRM Desktop\Profile
```

You can change this directory. For more information, see "Setting the Installation Directory of the CRM Desktop Add-In" on page 102.

Table 4 describes where CRM Desktop stores data in the file system.

Table 4. Example of Where Siebel CRM Desktop Stores Data in the File System

Micorsoft Windows Folder on Client	Description
%APPDATA%\Oracle\CRM Desktop\bin	CRM Desktop saves the following information: ■ Add-in dll files. ■ Resources that binary files use.For example, Microsoft Word helper for Outlook 2003 email processing. ■ Microsoft Visual Studio run-time libraries. ■ Help files.
%APPDATA%\Oracle\CRM Desktop\Profile	CRM Desktop saves the following information: ■ The Data folder. This folder includes package files. ■ CRM Desktop log files. ■ Various database files.
%APPDATA%\Oracle\CRM Desktop\Profile\Data	XML and JavaScript files of the customization package. For more information, see "Files That the Customization Package Contains" on page 434.
%APPDATA%\Oracle\CRM Desktop\Profile\Logs	For more information, see "Log Files You Can Use with Siebel CRM Desktop" on page 119.
%TEMP%	When you download the customization package CRM Desktop places some files in a temporary directory.

Changes That Siebel CRM Desktop Makes in the Windows Registry

CRM Desktop adds registry entries differently depending on one of the following options that you choose when you install the CRM Desktop add-in:

- **Anyone Who Uses This Computer.** CRM Desktop adds registry entries in the following registry keys:
  - HKEY_LOCAL_MACHINE\Software\Oracle\CRM Desktop
  - HKEY_CURRENT_USER\Software\Oracle\CRM Desktop for the user who is currently logged in
- **Only For Me.** CRM Desktop adds registry entries only in the following registry key
  - HKEY_CURRENT_USER\Software\Oracle\CRM Desktop

For more information, see "Installing the CRM Desktop Add-In" on page 87.

These settings include the following information:

- General settings for CRM Desktop, such as login information.
- In the Logging subkey, logging settings that CRM Desktop uses to tune logging behavior.
- In the StructureBackup subkey, backup information from the personal folders file. For more information, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25.

CRM Desktop registers COM classes in the Windows Registry when you install CRM Desktop. For more information about Windows Registry settings that Microsoft Windows requires to register COM classes, see the topic about Registering COM Applications at the Microsoft Developer Network Web site.

For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

Changes That Siebel CRM Desktop Makes to Settings in Microsoft Outlook

Siebel CRM Desktop adds the following items to the personal folders file:

- Custom folders
- Custom views
- Custom objects
- Custom forms

CRM Desktop runs as a Microsoft Outlook add-in, so it must register with Outlook. For more information about Windows Registry settings that Microsoft Windows requires to register an add-in, see the topic about Registry Settings for COM Add-Ins at the Microsoft Developer Network Web site.

For more information, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25.

Process of Installing the CRM Desktop Add-In

This process is a step in "Roadmap for Installing Siebel CRM Desktop" on page 77. You do the following work to install the CRM Desktop add-in:

1. Preparing Your Environment for Installation on page 85
2. Installing the CRM Desktop Add-In on page 87

For more information, see "Options for Installing the CRM Desktop Add-In" on page 89.

Preparing Your Environment for Installation

This task is a step in "Process of Installing the CRM Desktop Add-In" on page 85.

This topic describes how to make sure you configure the network and infrastructure to successfully install and start the CRM Desktop add-in.

To prepare your environment for installation

1. Choose your deployment software and review the conditions that apply for the installation.

   For more information, see "Overview of Installing the CRM Desktop Add-In" on page 81.

2. Make sure a direct connection to the Siebel Server is available.

   CRM Desktop uses information from one of the following sources to connect to EAI (Enterprise Application Integration):

   - **Parameters during installation.** This configuration is appropriate if you install CRM Desktop in the background. For more information, see "Setting the URL for the Siebel Server" on page 102.
   - **Connection settings dialog box.** This configuration is appropriate if you install CRM Desktop manually. For more information, see "Customizing How First Run Assistant Uses the Customization Package" on page 90.

3. Make sure the EAI object manager on the Siebel Server is enabled and online.
4. Make sure only a single Position is defined for the user account.

   The user cannot use Microsoft Outlook to change the position. It is recommended that you use only a single Position for a given user account.

5. Make sure the customization package for the user position is published for only one of the user responsibilities.
6. Verify that the email address you use in the Outlook account in the profile where you install CRM Desktop is the same as the email address for this employee record on the Siebel Server.
7. Make sure you uploaded and published the customization package on the Siebel Server.

   For more information, see "About the Customization Package" on page 33.

8. Make sure the number of records for each type of Siebel object, such as accounts, is limited to an amount that the local email folder and environment can accommodate.

   This amount depends on the following items:

   - Size of the Outlook folder
   - The Outlook version
   - Where CRM Desktop stores the databases
   - Connectivity
   - Hard disk space on the client and on the computer
   - Capabilities of the client computer
   - And so on

   It is recommended that you test these amounts in a test environment before you deploy CRM Desktop to all users. For more information, see "Controlling the Number of Records That Synchronize" on page 142.

Installing the CRM Desktop Add-In

This topic describes how to manually install the CRM Desktop add-in.

To install the CRM Desktop add-in

1 Make sure requirements for the operating system are met.

The CRMDesktop.msi installation package validates the operating system version and the Outlook version that is currently installed on the client computer. For more information, see "Preparing the Implementation Environment for Siebel CRM Desktop" on page 77.

2 Make sure Outlook is installed on the client computer and configured for use.

If it is not, then an error occurs and CRM Desktop ends the installation.

3 Make sure you possess rights on the client computer so that you can run the executable file that CRM Desktop provides in the installation package.

4 Manually copy the CRMDesktop.msi file to the client computer.

To use third-party deployment software to deploy the CRMDesktop.msi file to multiple users, see "Installing Siebel CRM Desktop in the Background" on page 98.

5 Locate the CRMDesktop.msi installation package on the client computer.

The following directory is a typical location:

```
C:\Documents And Settings\username\Desktop
```

6 Run the CRMDesktop.msi installer.

7 In the Welcome dialog box, click Next.

8 In the Customer Information dialog box, enter the user name and the organization.

9 Choose to install the add-in for one of the following, and then click Next:

- **Anyone Who Uses This Computer.** Any user who logs on to this computer can use the CRM Desktop add-in.
- **Only For Me.** Only the user who is logged on to the computer when CRM Desktop installs the CRM Desktop add-in can use this add-in.

10 In the Destination Folder dialog box, specify the folder where the installer must install CRM Desktop.

You can specify any directory. For more information, see "Setting the Installation Directory of the CRM Desktop Add-In" on page 102.

11 In the Ready to Install the Program dialog box, click Install.

You can install CRM Desktop for multiple users, so the user who is currently logged in can view the application files that it stores in the following default directory:

```
c:\Documents and Settings\username1\Application Data\Oracle
```

CRM Desktop stores the files for another user on this computer in the following directory:

c:\Documents and Settings*username2*\Application Data\Oracle

12 In the InstallShield Wizard dialog box, click Finish.

If you add a check mark to the Launch CRM Desktop check box, then CRM Desktop finishes the installation and then does one of the following depending on if Outlook is open:

- Outlook **is open.** Prompts the user to apply the configuration to the current Outlook profile.
- Outlook **is not open.** Starts Outlook and then prompts the user to apply the configuration to a profile that the user chooses.

  CRM Desktop displays a dialog box. For more information, see "How Siebel CRM Desktop Installs the Siebel CRM Desktop Profile" on page 88.

If you do not add a check mark to the Launch CRM Desktop check box, then CRM Desktop finishes the installation and then does one of the following depending on if Outlook is open:

- Outlook **is open.** To start CRM Desktop, you can choose the Start menu in Microsoft Windows, choose All Programs, Oracle, and then click Launch CRM Desktop.
- Outlook **is not open.** CRM Desktop opens the first time you open Outlook after the installation finishes.

CRM Desktop installs the CRM Desktop add-in the background. The user can use the First Run Assistant to set up this add-in the next time this user accesses Outlook.

NOTE: The digital signature for the CRM Desktop installers will be signed by Avora Holdings, Inc. instead of Oracle. This is expected behavior and does not indicate a security or other issue.

How Siebel CRM Desktop Installs the Siebel CRM Desktop Profile

Siebel CRM Desktop displays a dialog box that allows the user to apply the CRM Desktop configuration to this Outlook profile. It does this when Outlook runs for first time after you install the CRM Desktop add-in. The user can choose one of the following values:

- **Yes.** CRM Desktop applies the configuration and then displays the First Run Assistant.
- **No.** The CRM Desktop add-in closes. The dialog box that allows the user to apply the CRM Desktop configuration displays each time the user starts Outlook until the user chooses to apply this configuration.

  The following choice that you make in Step 9 on page 87 determines if CRM Desktop applies this behavior:

  - **Anyone Who Uses This Computer.** This behavior applies to any user who logs on to this computer.
  - **Only For Me.** This behavior applies only to the user who is logged on to the computer when CRM Desktop installs the CRM Desktop add-in.

Options for Installing the CRM Desktop Add-In

This topic describes options that are available for installing the CRM Desktop add-in. It includes the following topics:

Customizing the First Run Assistant

This topic describes how to customize the First Run Assistant. It includes the following topics:

The *First Run Assistant* is a wizard that guides the user through the first setup of the CRM Desktop add-in. CRM Desktop displays the CRM Desktop icon in the system tray and starts the First Run Assistant. It does this the first time the user starts Microsoft Outlook after you install the CRM Desktop add-in. The user can begin using Outlook after the user finishes using this assistant.

The First Run Assistant displays a dialog box at each step that allows the user to specify settings. This topic describes how you can customize the behavior of some of these dialog boxes. For more information, see “Overview of How Siebel CRM Desktop Synchronizes Data” on page 24.

Customizing How First Run Assistant Uses the Customization Package

Table 5 describes how you can customize the First Run Assistant to register and get the customization package. It lists work items in the order that the user performs them while the user uses this assistant. The user must install the CRM Desktop add-in first and then use the assistant. For more information, see "Installing the CRM Desktop Add-In" on page 87.

Table 5. How First Run Assistant Registers and Gets the Customization Package

Step	Description	Possible Customization
1	The user opens Outlook the first time after the CRM Desktop add-in is installed. It is the first time that Outlook is open after the add-in is installed, so First Run Assistant displays the welcome screen and then the user clicks it.	Not applicable
2	First Run Assistant examines the connection settings and does one of the following: ■ If CRM Desktop establishes a connection, then the assistant continues. ■ If CRM Desktop does not establish a connection, then the assistant displays settings from the Connection tab of the Options dialog box. CRM Desktop chooses the Use Internet Explorer Settings for Proxy-Server option, by default. The Manual Proxy-Server Configuration option allows the user to specify a proxy server. If your organization uses a proxy server, then you must provide the user with the following information: ■ The host name for the proxy server in the Server window. ■ The port number in the window that displays immediately to the right of the Server window. The proxy server requires a separate host name and a port number.	For more information, see "Customizing How Siebel CRM Desktop Connects to the Internet" on page 91.

Table 5. How First Run Assistant Registers and Gets the Customization Package

Step	Description	Possible Customization
3	CRM Desktop establishes a network connection and then theFirst Run Assistant displays the CRM Desktop-Login dialog box. The user enters the user name and password. This user name must include the First Name and Last Name or the User ID of the user record that resides in the Siebel database. The user can enter the First and Last name in any order. The USERID is the same user ID that the user uses for the Siebel Web Client. For example, Wasaka Takuda, or WTAKUDA. The password is the same password as the password that the user uses for the Siebel Web Client.	For more information, see the following topics: ■ Changing Behavior of the CRM Desktop-Login Dialog Box on page 92 ■ Using the Windows Registry to Control Siebel CRM Desktop on page 105
4	First Run Assistant automatically enters the URL that the Siebel Business Application uses to connect to the Siebel Server. It enters this URL in the Server URL window. For example: http://*server_name*/eai/enu	You can specify the URL. For more information, see "Setting the URL for the Siebel Server" on page 102.

Customizing How Siebel CRM Desktop Connects to the Internet

You can customize how Siebel CRM Desktop connects to the Internet.

To customize how Siebel CRM Desktop connects to the Internet

1. Use an XML editor to open the platform_configuration.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Locate the initialization section of the initialization_script platform section.

3. Add the following code to the section that you located in Step 2:

   ```
   application.settings.set("ProxyUsage", value);
   where:
   ```

   ■ *value* is an integer. Use values from the following table.

Value	Description
0	Use the proxy server setting that is set in Internet Explorer.
1	Use a direct connection to the Internet. This option does not use a proxy server.
2	Use a manual proxy server configuration.

4 Save and then close the platform_configuration.xml file.

5 Test your work.

Changing Behavior of the CRM Desktop-Login Dialog Box

You can change the behavior of the CRM Desktop-Login dialog box. For information about authentication options, see Chapter 12, "Customizing Authentication."

To change behavior of the CRM Desktop-Login dialog box

- Hide the Save Password check box that Siebel CRM Desktop displays in the CRM Desktop-Login dialog box. You set the following Windows Registry key to 1:

  `Siebel:HideSavePasswordOption`

  If the user clicks Save Password in the CRM Desktop-Login dialog box, then CRM Desktop saves an encrypted copy of the password locally in the client computer. If you suppress display of the Save Password check box, then the user must enter the password every time the user logs into CRM Desktop. For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

- Prevent CRM Desktop from displaying the CRM Desktop-Login dialog box. You do the following:

  a Set the following Windows Registry key to 1:

  `SuppressLoginDialog`

  b Set the save_password parameter and the Login externally.

  If you do not set the save_password parameter, then CRM Desktop requires the user to enter the password every time the user opens Outlook and then synchronizes.

  For more information, see "How Siebel CRM Desktop Suppresses the Desktop-Login Dialog Box" on page 92.

How Siebel CRM Desktop Suppresses the Desktop-Login Dialog Box

If you suppress display of the Desktop-Login dialog box, then Siebel CRM Desktop does the following:

- If the login, password, and URL connection parameters exist in the Windows Registry, and if save_password exists in the Windows Registry and is set to 1, then CRM Desktop attempts to validate the user credentials on the Siebel Server.
- If the Siebel Server returns an error for this login, then CRM Desktop displays the Desktop-Login dialog box and allows the user to attempt to login or to cancel the login. If the Siebel Server cannot validate the login credentials, then it returns an error.
- If a connection parameter is not present in the Windows Registry, or if save_password does not exist in the Windows Registry, or if it is set to 0, then the Siebel Server returns a Credentials Verification Failed error.

Customizing How the First Run Assistant Performs the Initial Synchronization

Siebel CRM Desktop installs the folder structure, as described in Table 5 on page 90, and then displays the second part of the First Run Assistant. It prompts the user to set preferences and to run the first synchronization session that downloads Siebel CRM records to Microsoft Outlook. Table 6 describes the work that you can do to customize how the assistant does this initial synchronization. It lists work items in the order that the user does them while the user runs the assistant.

Table 6. How First Run Assistant Performs the Initial Synchronization

Step	Description	Administrative Work
1	First Run Assistant installs the folder structure. It then displays the following choices in the Filter Records tab of the Synchronization Control Panel dialog box: ■ Leave the filters at their default settings. ■ Choose a filter from the predefined filter that CRM Desktop deploys with the CRM Desktop add-in. ■ Specify filter settings. The user can also specify the synchronization frequency and other settings that CRM Desktop uses.	For more information, see the following topics: ■ Controlling the Object Types That Siebel CRM Desktop Displays in the Filter Records Tab on page 131 ■ Controlling the Size and Type of Synchronized Records on page 139
2	The First Run Assistant displays a dialog box that allows the user to configure synchronization settings. CRM Desktop does the following, by default: ■ Enters a check mark in the Schedule for the Automatic Synchronization Interval check box ■ Enters a check mark in the Show Progress During Automatic Synchronization check box ■ Sets the frequency slide bar to Once an Hour	For more information, see "Controlling the Synchronization Intervals That Display in the Synchronization Tab" on page 137.
3	The First Run Assistant displays a dialog box that allows the user to share with CRM Desktop each new native Outlook calendar appointment, contact, or task that the user creates in Outlook. CRM Desktop includes a check mark in the Calendar Appointment, Contacts, and tasks check boxes, by default.	For more information, see "Customizing How Siebel CRM Desktop Shares Native Microsoft Outlook Items" on page 94
4	The First Run Assistant displays the Siebel CRM Desktop dialog box. For more information, see "Sharing a Calendar Appointment, Contact, or Task" on page 94.	For more information, see "Controlling How Siebel CRM Desktop Assigns Calendar Appointment Owners" on page 110.

The user finishes specifying the configuration settings, and then CRM Desktop automatically starts the synchronization and adds content to the Siebel CRM folders. This content depends on choices the user specifies in the First Run Assistant. The synchronization finishes, and then the user can find the Siebel CRM data that CRM Desktop downloaded in the corresponding Siebel CRM folders. The user can view Siebel contacts that Siebel CRM Desktop downloaded to the Outlook Contacts folders. CRM Desktop does not automatically share contacts that existed in Outlook before you installed CRM Desktop.The user can use icons or group contacts to separate unshared contacts from Siebel CRM contacts according to the Shared and Not Shared attribute.

Customizing How Siebel CRM Desktop Shares Native Microsoft Outlook Items

You can customize Siebel CRM Desktop to share or not share any new native Microsoft Outlook items that the user creates in Outlook, such as a Outlook calendar appointment, contact, or task.

To customize how Siebel CRM Desktop shares native Microsoft Outlook items

1. Use an XML editor to open the platform_configuration.xml.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Locate the initialization section of the initialization_scriptplatform section.

3. Add the following code to the section that you located in Step 2:

   ```
   application.settings.set("SharedByDefault:NewItems", value);
   ```

   where:

   - *value* is an integer. Use values from the following table.

Value	Description
0	Do not share Outlook item.
1	Share Outlook item.

4. Save and then close the platform_configuration.xml file.

5. Test your work.

Sharing a Calendar Appointment, Contact, or Task

The user can determine how Siebel CRM Desktop shares records with the Siebel Server according to the following settings:

- **Set default sharing for all new records.** The user can use the Advanced tab of the CRM Desktop - Options dialog box to change how Siebel CRM Desktop creates a new Outlook calendar appointment, contact, or task as shared or not shared.
- **Set sharing for individual records.** The user can click the Share Bar that CRM Desktop displays at the start of a record form to share or unshare a single record.

Suppressing the Dialog Boxes That First Run Assistant Displays

This topic describes how to suppress the dialog boxes that First Run Assistant displays.

To suppress the dialog boxes that First Run Assistant displays

1. Use a JavaScript editor to open the application_script.js file.
2. Modify the following code:

```
var fra = application.fra;
function fra_handler(fra)
{
  var current_form = null;
  var on_closed = function()
  {
    current_form = null;
    fra.exit_current_step(false);
  }
  function on_fra_step(id)
  {
    if (id == "advanced")
    {
      var xml = ui.get_dialog_xml("PropSheetHost");
      xml = helpers.replace_all
      ("$prop_sheet_layout", "options_advanced_page", xml);
      current_form = ui.create_dialog_from_xml(0, xml);
      current_form.on_closed.connect(on_closed)
      current_form.visible = true;
    }
  }
  fra.on_step.connect(on_fra_step);
  fra.add_builtin_step("welcome");
  fra.add_builtin_step("sync_filters");
  fra.add_builtin_step("sync_schedule");
  fra.add_step("advanced", session.res_string("sa-advanced_settings-caption"),
session.res_string("sa-advanced_settings-description"), "sa-advanced_settings-
picture", true);
  fra.add_builtin_step("convert_items");
  fra.add_builtin_step("first_sync");

}
  function create_fra_handler(fra)
{
return fra != null ? new fra_handler(fra) : null;
}
var g_fra_handler = create_fra_handler(application.fra);
```

where:

- **bold** indicates code you can modify, as described in Table 7 on page 96.

Table 7 describes the code that you can modify to suppress the dialog boxes that First Run Assistant displays.

Table 7. Code That Displays the Dialog Boxes That First Run Assistant Displays

Default Code	Description
`fra.add_builtin_step("welcome");`	This code displays the Welcome dialog box. It is recommended that you do not remove it.
`fra.add_builtin_step("sync_filters");`	This code displays the default synchronization filters. You can remove this code to hide these filters.
`fra.add_builtin_step("sync_schedule");`	This code displays the default synchronization schedule. You can remove this code to hide this schedule.
`fra.add_step("advanced", session.res_string("sa-advanced_settings-caption"), session.res_string("sa-advanced_settings-description"), "sa-advanced_settings-picture", true);`	This code displays the default advanced settings. You can remove this code to hide the advanced settings.
`fra.add_builtin_step("convert_items");`	This code displays the native contacts conversion. You can remove this code to hide the native contacts conversion.
`fra.add_builtin_step("first_sync");`	This code displays the first synchronization step, It is required. Almost no CRM Desktop functionality is available before the first synchronization. You must not remove this code.

Configuring Contact Conversion Options for First Run Assistant

The Convert Items option of the First Run Assistant displays the Confirm Outlook Contact Conversion dialog box that allows the user to convert existing Outlook contacts to unshared business contacts. You can modify this behavior.

Modifying the Default Button of the Confirm Outlook Contact Conversion Dialog Box

CRM Desktop sets the Yes button of the Confirm Outlook Contact Conversion dialog box as the default button. This topic describes how to modify this behavior so that CRM Desktop sets the No button as the default and runs the No behavior if the user presses the Enter key.

To modify the default button of the Confirm Outlook Contact Conversion dialog box

1 Use an XML editor to open the platform_configuration.xml.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Locate the following code:

```
<initialization_script>
  <![CDATA[
  ]]>
</initialization_script>
```

3 Modify the code you located in Step 2 to the following. You add the bolded code:

```
<initialization_script>
  <![CDATA[
application.settings.set("FRA:ConvertItemsMsgBoxDefaultNo", 1);
  ]]>
</initialization_script>
```

Always Converting Contacts

The First Run Assistant prompts the user to convert Outlook contacts to Siebel CRM contacts, by default. This topic describes how to configure Siebel CRM Desktop to convert Outlook contacts when the First Run Assistant runs but to not display this prompt.

To always convert contacts

1 Use an XML editor to open the platform_configuration.xml file.

2 Modify the code that you locate in Step 2 on page 97 to the following. You add the bolded code:

```
<initialization_script>
<![CDATA[
application.settings.set("FRA:SuppressConvertItemsMsgBox ", 1);
]]>
</initialization_script>
```

Never Converting Contacts

This topic describes how to configure CRM Desktop so that it never converts contacts.

To never convert contacts

1 Use a JavaScript editor to open the application_script.js file.

2 Locate the following function:

```
function fra_handler(fra)
```

3 In the function you located in Step 2, locate the following code:

```
fra.add_builtin_step("convert_items");
```

4 Comment the code you located in Step 3. For example:

```
//fra.add_builtin_step("convert_items");
```

Removing this code configures CRM Desktop to not display the Convert Items option of the First Run Assistant and to not convert existing Outlook contacts to unshared business contacts.

Storing Siebel Object Types in Microsoft Outlook Storage

The predefined customization package stores some Siebel object types in the local CRM Desktop database instead of storing them in Outlook storage in a .pst or .ost file. This topic describes how to configure CRM Desktop to store all Siebel object types in native Outlook storage and to store no objects in the local CRM Desktop database. For more information, see "How Siebel CRM Desktop Stores Siebel CRM Data" on page 25.

To store Siebel object types in Microsoft Outlook storage

1 Describe the structure of the object and then create mappings between fields, lists, and so on.

For details on how to do this, see "Creating the Custom Object" on page 232. This topic describes how to define object types in the type element in the siebel_basic_mapping.xml file instead of in the database element. CRM Desktop stores an object type in the local CRM Desktop database only if the database element defines this object type.

2 Repeat Step 1 for each object type that CRM Desktop must store in Outlook storage.

3 Administer the metadata files.

Installing Siebel CRM Desktop in the Background

This topic describes how to install Siebel CRM Desktop in the background.

Using Microsoft System Center Configuration Manager to Install Siebel CRM Desktop

This topic describes how to use Microsoft System Center Configuration Manager to install Siebel CRM Desktop. For more information, see the documentation about using System Center Configuration Manager at the Microsoft TechNet Web site.

To use Microsoft System Center Configuration Manager to install Siebel CRM Desktop

1 Log on to the computer that includes System Center Configuration Manager, and then Open Microsoft System Center Configuration Manager 2007 or Microsoft Systems Management Server 2003.

2 Add all custom properties to the Windows Installer transform (.mst) file.

For more information, see "Adding Custom Properties to the Windows Installer Transform File" on page 99.

3 Run the installer. Open a Windows command line and then enter the following command:

```
msiexec /I "CRMDesktop.msi" TRANSFORMS="crmdesktop.mst" ALLUSERS=1 /qb!
```

where:

- ALLUSERS=1 is an optional parameter. To install CRM Desktop for anyone who uses the client computer, you must include the ALLUSERS parameter.

Use the following guidelines:

- Run setup with one of the following administrative rights:
  - **Administrative rights.** Installs CRM Desktop for anyone who uses the client computer.
  - **User rights.** Installs CRM Desktop only for the person who is currently logged into Windows on the client computer. Make sure this user possesses the permissions that Windows requires to run the installer.

Adding Custom Properties to the Windows Installer Transform File

You must add all custom properties to the Windows Installer transform (.mst) file. A *custom property* is any property that MSDN (Microsoft Developer Network Platforms) does not describe. For example, SIEBEL_SERVER_PROTOCOL and SIEBEL_SERVER_PORT are custom properties. For a complete list of the custom properties you must add, see "Setting the URL for the Siebel Server" on page 102.

Using a Windows Group Policy to Install Siebel CRM Desktop

This topic describes how to use a Windows group policy to install Siebel CRM Desktop for each user. It includes an optional step that describes how to install it for anyone who uses the client computer. For more information, see the documentation about using group policies at the Microsoft TechNet Web site.

To use a Windows group policy to install Siebel CRM Desktop

1 Log on to the computer that includes your group policy manager.

2 Make sure the directory that stores the installer and the .mst file is available on the local network.

3 Open the Microsoft Group Policy Editor.

4 Create an installation package in the GPO snap-in in the following branch:

```
Computer Configuration - > Software Settings - > Software installation
```

5 Set the Deployment type to Assigned.

6 Create a Windows Installer transform .mst file.

7 Add the path to the Windows Installer .mst transform file.

8 Add all custom properties to the transform file.

You cannot use the command line with a group policy object (GPO). You must specify all properties in the .mst file. For more information, see "Adding Custom Properties to the Windows Installer Transform File" on page 99.

9 (Optional) To install CRM Desktop for anyone who uses the client computer, do the following:

a Set the ALLUSERS property to 1 in the Property table. You set this property in the transform file that you create in Step 6.

b Make sure each CRM Desktop user possesses the permissions to run this msi package.

An administrator might disallow the parameter that provides these permissions. If the user does not possess these permissions, then CRM Desktop does not run the installation when it creates the Outlook profile.

Using the Windows Command Line to Set Optional Parameters

You can use the Windows command line to set optional parameters that affect installation. You can run the CRMDesktop.msi installation package from the Windows command line interface on the client computer. Siebel CRM Desktop supports all parameters that you can set in the Windows Installer msiexec command line. For more information, see the documentation about command line options for Windows Installer at the Microsoft TechNet Web site.

To use the Windows command line to set optional parameters

1 On the client computer, open a Windows command line:

a In Windows, click Start and then click Run.

b In the Run dialog box, enter cmd and then click OK.

2 Navigate to the directory that contains the CRMDesktop.msi file.

For example:

```
C:\Documents and Settings\username\Desktop
```

3 Enter the Windows Installer command using the following format:

```
msiexec.exe /I CRMDesktop.msi optional_parameter_1 optional_parameter_n
```

where:

- *optional_parameter* is a parameter you can enter that CRM Desktop runs. For example:

```
msiexec /i "C:\Documents and Settings\username\Desktop\"CRMDesktop.msi"
SIEBEL_SERVER_HOST="siebelserver.com" SIEBEL_SERVER_PORT="80"
SIEBEL_SERVER_SUFFIX="SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1"
SIEBEL_SERVER_PROTOCOL="http" SIEBEL_SERVER_COMPONENT="eai/enu"
```

- Optional) Add the following optional parameter to enable the SOAP log:

```
SOAP_DUMP_ENABLED=1
```

- (Optional) Add the following optional parameter to enable the SYNC log:

```
SYNC_DUMP_ENABLED=1
```

For more information, see "Guidelines for Using Synchronization Log Parameters" on page 101

Note the following requirements:

- You must specify each optional parameter in the same command line after the name of the CRMDesktop.msi file.
- To separate each optional parameter, you must enter a space without a slash (/).
- You can arrange optional parameters in any order.

For information about how to set CRM Desktop SSO parameters, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380.

4 Press Enter.

The welcome dialog box of the Siebel CRM Desktop Setup wizard displays.

Guidelines for Using Synchronization Log Parameters

A synchronization log includes the following parameters. These parameters measure the average percentage of CPU load time that Siebel CRM Desktop uses during synchronization:

- **kernel_cpu.** Measures the entire system.
- **process_cpu.** Measures the process that runs CRM Desktop.

It is recommended that you do not use these parameters to debug CRM Desktop. Instead, it is recommended that you use other tools to measure CPU load time, such as the Process Explorer system utilities for Windows or the Windows Task Manager.

Hiding Dialog Boxes That Require User Input

You can use the optional QR parameter to hide dialog boxes that require user input.

To hide dialog boxes that require user input

- Append the QR parameter to the msiexec command.

For example:

```
msiexec.exe /I CRMDesktop.msi INSTALLDIR=c:\My_Custom_Directory/QR
```

If you add this parameter, then the CRMDesktop.msi installation package does not display dialog boxes that require user input.

Setting the Installation Directory of the CRM Desktop Add-In

You can use the optional INSTALLDIR parameter to change the default location where the CRMDesktop.msi installation package saves files during installation for a single user. CRMDesktop.msi installs to the following directory, by default:

```
c:\Documents and Settings\username\Application Data\Oracle\CRM Desktop\
```

To set the installation directory of the CRM Desktop add-in

- Enter the following parameter on the msiexec command line anywhere after the mandatory CRMDesktop.msi name parameter:

  ```
  INSTALLDIR=directory_path
  ```

  For example:

  ```
  \Documents and Settings\username\Desktop\CCRMDesktop.msi
  ```

  where:

  - *user name* is the name of the user, such as WTAKUDA.

Setting the URL for the Siebel Server

You can specify the URL that the Synchronization Engine uses to connect with the Siebel Server.

To set the URL for the Siebel Server

- Enter the following parameters on the msiexec command line anywhere after the mandatory CRMDesktop.msi name parameter:

  ```
  SIEBEL_SERVER_PROTOCOL=protocol SIEBEL_SERVER_HOST=host_name_or_address
  SIEBEL_SERVER_PORT=server_port SIEBEL_SERVER_COMPONENT=component_name
  SIEBEL_SERVER_SUFFIX=request_suffix
  ```

  where:

  - *protocol* is http. HTTP is the default value.
  - *host_name_or_address* is the computer name or IP address of the target server. This parameter is empty, by default. To use a fully qualified domain name for the server_address variable, you must set the EnableFQDN parameter in the configuration (cfg) file. For more information, see *Siebel System Administration Guide*.
  - *server_port* is 80. 80 is the default value.
  - *component_name* is eai/enu. eai/enu is the default value.
  - *request_suffix* is the following default value:

    ```
    ?SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1
    ```

  For example:

```
msiexec.exe /I CRMDesktop.msi SIEBEL_SERVER_PROTOCOL=http
SIEBEL_SERVER_HOST=sdcv440s133.siebel.com SIEBEL_SERVER_PORT=80
SIEBEL_SERVER_COMPONENT=eai/enu SIEBEL_SERVER_SUFFIX=
?SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1
```

No parameters are required.

Any information that you set in these parameters sets the parameter values in the Windows Registry, so the user is not required to set them. For example, the protocol variable of the SIEBEL_SERVER_PROTOCOL parameter overrides the Siebel:Protocol entry in the Windows Registry. For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105You can also use XML code to override the URL. For more information, see "XML Code That Customizes Forms" on page 469.

Troubleshooting Siebel CRM Desktop Installation

Siebel CRM Desktop might display a message during installation that is similar to one of the following error messages:

```
The System cannot open the device or file specified
```

```
Error 2755.Server returned unexpected error 110 attempting to install package
```

This problem might be due to the fact that the CRMDesktop.msi file is encrypted or is located in a directory where the user does not possess run permissions.

To troubleshoot Siebel CRM Desktop installation

1. Open Windows Explorer.
2. Right-click the .msi installation file and then click Properties.
3. In the General tab, click Advanced.
4. Make sure the following option does not contain a check mark and then click OK.

   Encrypt contents to secure data
5. Reinstall CRM Desktop.

7 Administering Siebel CRM Desktop

This chapter describes how to administer Siebel CRM Desktop. It includes the following topics:

Controlling the Behavior of Siebel CRM Desktop

This topic describes how you can control the behavior of Siebel CRM Desktop. It includes the following topics:

Using the Windows Registry to Control Siebel CRM Desktop

You can use Windows Registry keys to control Siebel CRM Desktop behavior. For example, you can specify the following items:

- Directory paths
- Passwords
- Synchronization parameters
- Connection timeouts
- Host names
- Ports
- Credentials

If you must reinstall CRM Desktop at some point in the future, then during this installation CRM Desktop deletes any changes you have made to the registry.

CAUTION: Modifying the Windows Registry can cause serious and permanent problems that you might not be able to resolve. You must be very careful to make only the modifications you require, and that the modifications you make do not negatively affect functionality or performance.

For more information, see "Changes That Siebel CRM Desktop Makes in the Windows Registry" on page 84.

To use the Windows Registry to control Siebel CRM Desktop

1. In Microsoft Windows, choose Start and then click Run.
2. In the Run dialog box, enter REGEDIT and then click OK.
3. Add or modify Windows Registry keys, as necessary.

   To automate changes to Windows Registry keys, you can use an administrative tool, such as Systems Management Server or Marimba.

   For information about the keys you can change, see "Registry Keys You Can Use with Siebel CRM Desktop" on page 411.

Configuring Siebel CRM Desktop to use HTTPS

You can configure the URL protocol to use HTTPS (Hypertext Transfer Protocol Secure). For more information, see "Setting the URL for the Siebel Server" on page 102.

To configure Siebel CRM Desktop to use HTTPS

1. Open a Windows command line, and then type `regedit.exe`.

   For more information, see "Using the Windows Command Line to Set Optional Parameters" on page 100.

2. Set the Siebel:Protocol registry key to https.

   For more information, see "Registry Keys That Affect Credentials" on page 415.

Overriding Windows Registry Keys That Locate the Siebel Server

You can use values in the connector_configuration.xml file to override the following registry settings:

- Siebel:ComponentName
- Siebel:RequestSuffix

The CRM Desktop add-in uses these entries to locate the Siebel Server. For more information, see "Setting the URL for the Siebel Server" on page 102 and "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

CAUTION: You cannot use the Login dialog box of the CRM Desktop add-in to edit the name and suffix of the server component. If the customization package includes values that cause the server connection to fail, then you must edit these values manually in the Windows Registry on the client computer.

To override Windows Registry keys that locate the Siebel Server

- Modify the following default_settings tag that resides in the platform tag of the connector_configuration.xml file:

  <*setting name*="Siebel:ComponentName" type="*string_or_int*"> *new_value*</setting>

  where:

  - *setting name* is the registry setting that CRM Desktop must override.
  - *type* is the key type. CRM Desktop supports the following types:
    - **string.** Specifies a string value.
    - **int**. Specifies an integer value.
  - *new_value* is the value that overrides the registry setting.

The default_settings tag and all attributes in the default_settings tag are optional. For more information, see "XML Code That Customizes Synchronization" on page 462.

Using the Metadata to Control Siebel CRM Desktop

You can use the metadata to control Siebel CRM Desktop. For example, you can set limits for the following items:

- Length of a repeating Calendar event
- Size of a file attachment
- Visibility of an object

The files that this topic describes are part of the customization package. You can use any editor that supports editing in JavaScript or XML, such as Notepad, to modify one of these files.

To use the metadata to control Siebel CRM Desktop

1 Set the maximum number of days, weeks, months, or years that CRM Desktop creates for a repeating Calendar event that does not match a Siebel CRM repeating pattern. You modify the recurrence_processing.js file. Use values from the following table.

Variable with Default Value	Description
var daily_max_length = 12	Sets the maximum number of months that CRM Desktop uses when it creates a repeating Calendar event. This Calendar event occurs daily.
var weekly_max_length = 12	Sets the maximum number of months that CRM Desktop uses when it creates a repeating Calendar event. This Calendar event occurs weekly.
var monthly_max_length = 24	Sets the maximum number of months that CRM Desktop uses when it creates a repeating Calendar event. This Calendar event occurs monthly.
var yearly_max_length = 60	Sets the maximum number of months that CRM Desktop uses when it creates a repeating Calendar event. This Calendar event occurs yearly.

2 Set limits. You modify the business_logics.js file. Use values from the following table.

Variable with Default Value	Description
var max_attach_file_size = 5	Sets the maximum size in megabytes of an attachment file in Outlook.
var action_selection_limit = 30	Sets the maximum number of items that CRM Desktop can process if the user runs a toolbar command. For example, if you set var action_selection_limit to 30, then the user can choose only 30 records in Outlook, and then use the toolbar in Outlook to run a command. An example command is Email to Contacts. You can modify this limit to avoid undesirable performance.

3 Edit the siebel_meta_info.xml file, using values from the following table. For more information, see "Customizing Meta Information" on page 164.

Variable	Description
max_commands_per_batch	Sets the maximum number of commands for each batch. For more information, see"Common Settings Tag of the Siebel Meta Information File" on page 487.
max_ids_per_command	Sets the maximum number of object IDs. For more information, see "Common Settings Tag of the Siebel Meta Information File" on page 487.

Variable	Description
open_with_url_tmpl	Sets a template for the code that CRM Desktop uses to create a URL to open the Siebel Web Client. For more information, see "Setting the URL That Siebel CRM Desktop Uses to Open the Siebel Web Client" on page 109.
ViewMode	Sets the visibility of an object. You can use one of the following values: ■ Sales Rep ■ Personal ■ Organization ■ All For example, `ViewMode = Sales Rep`. The value you set is specific to each object. In general, you set ViewMode to Sales Rep for an object that a position determines. Examples of an object that a position determines include a contact, account, or opportunity. For more information, see "Controlling the View Mode During Synchronization According to Object Type" on page 149.

Setting the URL That Siebel CRM Desktop Uses to Open the Siebel Web Client

You can specify the URL that Siebel CRM Desktop uses to open the Siebel Web Client when the user clicks the Siebel icon on the CRM Desktop toolbar. The user can navigate to the Siebel Web Client from the context of a record in Outlook to examine more details about the record. This feature is useful if the user must do work that requires data from the Siebel Web Client that is not available in Outlook. CRM Desktop does the following work:

- If the user clicks the Siebel icon on the CRM Desktop toolbar, or if the user clicks the Siebel icon on the Extension Bar of the form, then CRM Desktop opens a new browser window for the Siebel Web Client. Siebel CRM Desktop displays the detail form of the record that is currently chosen in Outlook. For example, a contact, an opportunity, or an account. The Siebel URL that you specify determines the browser window that CRM Desktop opens.
- If the user clicks the record but the record does not reside on the Siebel Server, or if the user does not possess visibility to this record, then CRM Desktop displays a message that is similar to This Record is Not Found in Siebel. This situation can occur if the user creates the record in Outlook but the record is not synchronized to the server.
- Does not display the button if the user chooses a contact that is not shared in Outlook.
- Does not display the button if the user chooses a native Outlook contact in Outlook.
- Displays an error message in the Siebel Web Client if the user does not possess direct visibility to the record. The user responsibility determines this visibility.

For more information, see "Customizing Meta Information" on page 164. For more information about templates, see the information about the open_with_url_tmpl variable in "Using the Metadata to Control Siebel CRM Desktop" on page 107.

To set the URL that Siebel CRM Desktop uses to open the Siebel Web Client

1. Use an XML editor to open the siebel_meta_info.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Locate the definition of the object type for the TypeId attribute of the object tag.

   Examples of these object types include account, contact, opportunity, activity, and so on.

3. Modify and then insert the open_with_url_tmpl tag in the object element of the object type that CRM Desktop must open in the Siebel Web Client.

   For more information, see "Open With URL Template Tag of the Siebel Meta Information File" on page 491

4. Repeat Step 1 and Step 3 for each type of object that CRM Desktop must open in the Siebel Web Client.

Controlling How Siebel CRM Desktop Handles CRM Data

This topic describes how to control how Siebel CRM Desktop handles CRM data. It includes the following topics:

Controlling How Siebel CRM Desktop Assigns Calendar Appointment Owners

You can use Siebel Multi-Org (multiple organization) to administer a calendar appointment that a non-Siebel user creates, such as an invitation from an external contact. For more information, see "How Siebel CRM Assigns Meeting Organizers" on page 44.

To control how Siebel CRM Desktop assigns calendar appointment owners

1. Log in to the Siebel CRM client with administrator privileges.
2. Navigate to the Administration - Application screen and then the System Preferences view.

3 In the System Preferences list, query the System Preference Name property for Generic Siebel Owner.

If you do not specify the Generic Siebel Owner parameter, then Siebel CRM sets each user who shares this meeting as the Activity Owner. If more than one of these users synchronizes the same Outlook meeting, then a duplication error occurs. For more information, see Resolving Synchronization Conflicts on page 154.

4 In the System Preference Value field, enter a user name.

Use the following guidelines:

- Make sure relationships exist between the user you specify as the Generic Siebel Owner and all organizations. This configuration allows any other user who creates a shared meeting to choose the user. If relationships do not exist between the user that you specify as the Generic Siebel Owner and all organizations, then the selection that the user makes fails and CRM Desktop displays an error message that indicates the user cannot choose the current record.
- Do not specify a real user as the Generic Siebel Owner. If you do this, then this user receives every calendar appointment that matches the criteria.
- It is recommended that you specify SADMIN as the Generic Siebel Owner for the following reasons:
  - SADMIN is not a real user.
  - Users are accustomed to viewing records that SADMIN creates.

Controlling How Siebel CRM Desktop Handles Email Attachments

Microsoft Outlook stores the Outlook email message and any attachments that this email contains an .msg file. The user can do one of the following to control how CRM Desktop handles an email attachment:

- Use the CRM Desktop - Options dialog box while using the First Run Assistant.
- Right-click the CRM Desktop icon in the system tray, choose Options, and then click the Advanced tab in the CRM Desktop - Options dialog box.

You can also use the Windows Registry to control how CRM Desktop handles an email attachment.

To control how Siebel CRM Desktop handles email attachments

1 Open the Windows Registry and then locate the following key:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop
```

For more information, see “Using the Windows Registry to Control Siebel CRM Desktop” on page 105.

2 Set the MailProcessing:AttachmentsHandling key to one of the following values:

Value	Description
0	Attach the entire .msg file to the Siebel CRM activity. This value is the default setting. If the registry key is absent, then CRM Desktop uses this default.
1	Attach the file attachments that the .msg file contains to the Siebel CRM activity.
2	Do not attach anything to the Siebel CRM activity.

Controlling the Maximum Size of an Attachment

You can control the maximum size of an attachment that Siebel CRM Desktop allows the user to attach to an object. The example in this topic sets the maximum size of an account attachment to 1 MB.

To control the maximum size of an attachment

1 (Optional) Set the maximum size of a file that CRM Desktop can attach to a specific object type that it synchronizes from the client to the Siebel Server:

a Use an XML editor to open the connector_configuration.xml file.

b Locate the following tag:

```
type id="Attachment"
```

Make sure you locate the tag that includes child `collection container_type` tags. This tag is typically the second instance of the `type id="Attachment"` tag in the connector_configuration.xml file.

c Modify the FileSize attribute of one of the child tags of the tag you located in Step b.

For example, to modify the maximum size for an account attachment, you modify the `value type="integer"` tag that the Account collection container contains. You use the following code. Bold text indicates the code you must change:

```
<type id="Attachment">
  <group link="or">
    <collection container_type="Account" foreign_key="ParentId">
      <primary_restriction>
        <group link="and">
          <binary field="FileSize" condition="le">
            <value type="integer">1048576</value>
          </binary>
```

where:

- 1048576 is the number of bytes in 1 MB.

d Save your changes and then close the connector_configuration.xml file.

2 (Optional) Set the maximum size of an attachment that CRM Desktop can synchronize from the Siebel Server to the client:

a Use an XML editor to open the siebel_meta_info.xml file.

b Set the value for the max_out_obj_size tag.

The max_out_obj_size tag sets the maximum size of an attachment for any object that CRM Desktop synchronizes from the Siebel Server to the client. If CRM Desktop encounters an attachment that is larger than the value you specify, then it displays an error message. For example, the following code sets this maximum size to 1 MB:

```
ErrMsg="#out_object_too_big_err_msg">1048576</max_out_obj_size>
```

where:

- 1048576 is the number of bytes in 1 MB.

This tag does not limit the size of a file that the user attaches in the CRM Desktop add-in.

c Save your changes and then close the siebel_meta_info.xml file.

3 (Optional) Set the maximum size of an attachment that the user can attach to a specific object type that CRM Desktop can synchronize from the Siebel Server to the client. This configuration does not affect an attachment that Siebel CRM creates automatically if it saves an email as a Siebel activity:

a Use an XML editor to open the business_logic.js file.

b Set the following variable to the required size:

```
var max_attach_file_size = number_of_megabytes; //MB
```

where:

- number_of_megabytes specifies the maximum size of an attachment that CRM Desktop allows the user to attach to an object, such as an account.

For this example, use the following code:

```
var max_attach_file_size = 1; //MB.
```

c Save your changes and then close the business_logic.js file.

Controlling How Siebel CRM Desktop Handles Archived Items

This topic describes how to control how Siebel CRM Desktop handles an archived item.

To control how Siebel CRM Desktop handles archived items

1 Use a JavaScript editor to open the business_logic.js file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Change the value for the following attribute:

```
var archive_activity_days = 7; // 7 days
```

3 Add your modification to a customization package.

4 Publish the customization package.

For more information, see "Creating and Publishing the Customization Package" on page 78.

How Siebel CRM Desktop Distinguishes Between Outlook Archive Items and Deleted Items

Siebel CRM Desktop does not distinguish between items in Outlook that Outlook archives and items that the user deletes. If Outlook archives an item, then CRM Desktop interprets this action as a deletion and it deletes the corresponding Siebel CRM record in the Siebel database. To modify this behavior, you can use the archive_activity_days variable in the business_logic.js file.

CRM Desktop does one of the following depending on the value of the end date that the deleted Outlook Calendar event contains:

- The end date occurs before the current system date minus the number of days that the archive_activity_days variable specifies. CRM Desktop does not delete the corresponding Siebel CRM record.
- The end date occurs after the current system date minus the number of days that the archive_activity_days variable specifies. CRM Desktop deletes the corresponding Siebel CRM record.

CRM Desktop sets the archive_activity_days variable to 7, by default. For example, if you leave the archive_activity_days variable at the default value, then CRM Desktop does the following:

- If a meeting occurred before the current system date minus 7 days, and if the meeting no longer exists in Outlook, then it treats the meeting as an archived item. It does not delete the corresponding Siebel CRM meeting from the Siebel database.
- If the user schedules a meeting to occur less than 7 days before the current system date, then it assumes the user intentionally deleted the meeting and it deletes the corresponding Siebel CRM meeting from the Siebel database.

How Siebel CRM Desktop Handles Archived Items

If Outlook archives a task, and if this task no longer exists in Outlook, then Siebel CRM Desktop examines the Status field in the To Do activity. It then does one of the following depending on if the value in the Status field of the To Do activity is Done:

- **Done.** It assumes the Siebel activity is an archived item and it does not delete the corresponding Siebel activity from the Siebel database.
- **Not Done.** it assumes the user intentionally deleted the Siebel activity and it deletes the corresponding Siebel activity from the Siebel database.

Storing Objects in a Database to Improve Performance

To improve performance, you configure Siebel CRM Desktop to store the child objects, seed data, lookup values, intersection records, and so on outside of the PST and OST files that Outlook typically uses for storage. CRM Desktop can store these items in an external database. To do this, you move these objects from the siebel_basic_mapping.xml file to the following tag:

database

To store objects in a database to improve performance

1. Use an XML editor to open the siebel_basic_mapping.xml file.
2. Add the database tag immediately following the existing sd2_meta tag:

```
<sd2_meta>
  <database>
...
</database>
```

3. Move the existing objects inside the region.

   For example:

```
<type id="Employee" icon="type_image:Employee:16">
  <field id ="Primary Organization Id">
    <type>
      <simple type="foreign_key"/>
    </type>
  </field>
  <field id ="Primary Position Id">
    <type>
      <simple type="foreign_key"/>
    </type>
  </field>
  <field id ="First Name">
    <type>
      <simple type="string"/>
    </type>
  </field>
  ...
</type>
```

   You must not move some objects. For more information, see "Objects That You Must Not Store in a Database" on page 116.

You must specify the following items in the type tag:

- **type id.** The name of the type.
- **icon.** The image from the resource that CRM Desktop must use for this type.

To define the field, you must specify the field Id of the field and the field type. CRM Desktop supports the following types:

- string

- integer
- double
- datetime
- boolean
- binary
- foreing_key (for ID fields).

Objects That You Must Not Store in a Database

The following objects must remain in the types tag. You must not move them:

- All built-in objects and their CRM Desktop equivalents, such as Contact, Email, task, Calendar, and so on
- All custom parent objects that CRM Desktop displays in the tree structure, such as Account or Opportunity
- The Action object, which serves as a proxy for Calendar, task, and Email items
- The Attachment object, which requires special handling and cannot be moved

Removing Siebel CRM Desktop

This topic describes how to remove the CRM Desktop add-in. It includes the following topics:

Removing the CRM Desktop Add-In for a Single User

This topic describes how to remove the CRM Desktop add-in if it is installed for a single user.

To remove the CRM Desktop add-in for a single user

1. Do the following:
   a. Do a synchronization in Microsoft Outlook.
   b. Backup personal data.

   To avoid a loss of data, it is recommended that the user synchronize and back up all personal data before removing the CRM Desktop add-in. For more information, see "How Siebel CRM Desktop Handles Items If the User Removes the CRM Desktop Add-In" on page 61.
2. Remove the CRM Desktop add-in:

a In Microsoft Windows, click the Start menu, choose Settings, and then open the Control Panel.

b In the Control Panel, open the Add or Remove Programs application.

c In the Currently Installed Programs window, click CRM Desktop, and then click Remove.

CRM Desktop automatically does the following work:

- Removes the data structure
- Removes Siebel CRM data
- Removes every shared calendar appointment and task that it created to support Siebel CRM activities
- Removes shared contacts

CRM Desktop treats native Outlook items in the following ways:

- Converts every unshared calendar appointment, task, and contact to a native Outlook item
- Leaves every shared calendar appointment and task that originated in Outlook as a native Outlook item in the corresponding Outlook folder
- Leaves every native Outlook item and Outlook email message in the corresponding Outlook folder

Removing the CRM Desktop Add-In for Multiple Users

You can use the System Center Configuration Manager (SCCM) to remove the CRM Desktop add-in in the background if it is installed for multiple users. You can use the SCCM management console to open the distribution package that you created when you installed CRM Desktop, open the Siebel CRM Desktop program, and then enter the following value in the command line:

```
msiexec /x "CRMDesktop.msi" /qr
```

You can enter the following command to remove CRM Desktop SSO for multiple users:

```
msiexec /x "InvisibleSSOModule.msi" /qr
```

For more information, see “Removing or Upgrading CRM Desktop SSO” on page 382.

For more information, see the topic about how to uninstall a product in the Windows Installer (msiexec) command line options section of the Microsoft TechNet Web site. For more information, see “How Siebel CRM Desktop Handles Items If the User Removes the CRM Desktop Add-In” on page 61.

Controlling the Data That Siebel CRM Desktop Removes

This topic describes how to control the data that Siebel CRM Desktop removes if the user removes CRM Desktop. The platform_configuration.xml file allows you to specify the custom data that it removes from Microsoft Outlook. If the user removes CRM Desktop or changes credentials, then it removes from Outlook all the custom data that the siebel_basic_mapping.xml file describes. You can configure CRM Desktop to not delete data for an object type. It does not remove data that the user creates in the native Outlook application that it shares with the Siebel Server, such as a Calendar event, task, or email message. For more information, see "XML Code That Customizes Platform Configuration" on page 462.

To control the data that Siebel CRM Desktop removes

1. Use an XML editor to open the platform_configuration.xml file.
2. Configure CRM Desktop to not delete data for a specific object type. You create a rule named skip for the appropriate type tag.

   For example:

   ```
   <type id="Action" rule="skip"/>
   ```

   In this example, CRM Desktop does not delete any data that the Action object type references.
3. Configure Siebel CRM Desktop to conditionally not delete data for a specific object type. You set the following attribute for the rule you defined in Step 2:

   ```
   language
   ```

   This language attribute defines the script language in the CDATA section. For example:

   ```
   <type id="Action" rule="script" language="JScript">
     <![CDATA[JavaScript code]]>
   </type>
   ```

   CRM Desktop supports only the JScript language.
4. Configure CRM Desktop to not delete data for multiple object types. You add a separate rule for each object type.

   For example:

   ```
   <type id="Action" rule="skip"/>
   <type id="Opportunity" rule="skip"/>
   ```

Administering Logging

This topic describes how to administer logging. It includes the following topics:

Log Files You Can Use with Siebel CRM Desktop

Table 8 describes log files that you can use with Siebel CRM Desktop. For information about how to enable these log files, see "Assigning Logging Profiles for Siebel CRM Desktop" on page 120.

Table 8. Log Files That You Can Use with Siebel CRM Desktop

Description	Where Stored
General Log. Includes information about general application events.	%APPDATA%\Oracle\CRMDesktop\Profile\Logs\GeneralLog CRM Desktop stores the first general log file as log.0000.txt. If this first file reaches 10 MB, then it creates a new file and increments the name of this file by 1. For example, log.0001.txt. A maximum of eight files can exist. If the eighth file reaches 10 MB, then CRM Desktop deletes the oldest General Log.
Exception Log. Includes information about CRM Desktop exceptions. These log files are for Oracle internal use only.	%APPDATA%\Oracle\CRMDesktop\Profile\Logs\ExceptionLog CRM Desktop stores the first exception log file as ex_trace.0000.txt. If this first file reaches 10 MB, then it creates a file and increments the name of this file by 1. For example, ex_trace.0001.txt. A maximum of eight files can exist. If the eighth file reaches 10 MB, then CRM Desktop deletes the oldest file.
Crash Log. Includes information about Outlook and CRM Desktop add-in failures.	%APPDATA%\Oracle\CRMDesktop\Profile\Logs\CrashDump A maximum of 48 crash log files can exist. If 48 files exist, and if CRM Desktop must create another crash log, then it deletes the oldest file.
SOAP Log. Includes information about requests that CRM Desktop sends to the Siebel Server and replies that it receives from the Siebel Server.	%APPDATA%\Oracle\CRMDesktop\Profile\Logs\SoapDump A maximum of 48 SOAP log files can exist. If 48 SOAP log files exist, and if CRM Desktop must create another SOAP log file, then it deletes the oldest SOAP log.
Synchronization Log. Includes synchronization events.	%APPDATA%\Oracle\CRMDesktop\Profile\Logs\SyncDump A maximum of 48 synchronization log files can exist. If 48 files exist, and if CRM Desktop must create another file, then it deletes the oldest file.

Assigning Logging Profiles for Siebel CRM Desktop

A *logging profile* is a set of parameters that determine logging settings. Siebel CRM Desktop comes with three predefined logging profiles and one custom profile. You can assign a predefined logging profile. You cannot change the set of parameters that a predefined logging profile uses. You can assign only one logging profile at a time.

To assign a logging profile for Siebel CRM Desktop

1. On the client computer, right-click the CRM Desktop icon in the system tray.
2. Choose Options and then click the Advanced tab in the CRM Desktop - Options dialog box.
3. Click Configure Logging and Reporting.
4. In the Logging Configuration dialog box, choose the Logging Policy using values from the following table.

Logging Policy	Log Enabled
Basic	General Log CRM Desktop sets the verbosity for the General Log to Info.
Detailed Exhaustive	CRM Desktop enables the following log files: ■ General Log ■ Exception Log ■ Crash Dump ■ Sync Dump ■ SOAP Dump CRM Desktop sets the verbosity for the General Log to the following: ■ Info for Detailed ■ Debug for Exhaustive
Custom	CRM Desktop uses a custom profile that you define. For more information, see "Creating Custom Logging Profile" on page 121.

5. Set Logging Verbosity.

   For more information, see "Setting Logging Verbosity" on page 121.

6. (Optional) Choose one or more of the following items:
   - Log Application Exceptions
   - Log Application Crashes
   - Log Sync Dumps

- Log SOAP Dumps

At run-time, CRM Desktop creates log entries for each item you choose. For more information, see "Log Files You Can Use with Siebel CRM Desktop" on page 119.

7 (Optional) Choose View Files for any item.

If you choose View Files, then CRM Desktop displays the folder that includes the log files for the corresponding log type.

8 Click OK.

Siebel CRM Desktop replaces the current logging settings with the setting that the profile you choose contains. It copies the parameters of this logging profile to the CRM Desktop logging settings in the Windows Registry in the following key:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\Logging
```

Setting Logging Verbosity

Logging verbosity is the level of detail that Siebel CRM Desktop writes to the General Log. You can set logging verbosity to one of the following values:

- **Debug.** Logs information messages, warnings, and all errors.
- **Info.** Logs only information messages.
- **Warning.** Logs only warnings.
- **Error.** Logs only errors.
- **Fatal.** Logs only fatal errors.

Creating Custom Logging Profile

The example in this topic describes how to create a logging profile that enables General Log, Synchronization Dump, and SOAP Dump log entries and sets verbosity for the General Log to Warning.

To create a custom logging profile

1 In Microsoft Windows, choose Start and then click Run.

2 In the Run dialog box, enter REGEDIT and then click OK.

3 Locate the following key:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\LoggingProfiles
```

4 Create a subkey in the key that you located in Step 3.

Enter a name for this subkey. At run time, CRM Desktop displays this name in the logging profile in the Logging Policy drop-down list of the Logging Configuration dialog box. You cannot localize a profile name. CRM Desktop stores this name in the registry with the settings that you specify for the profile. For more information, see "Assigning Logging Profiles for Siebel CRM Desktop" on page 120.

5 Create the following subkeys in the subkey that you create in Step 4. Create one subkey for each row that the following table contains.

Sub Key Name	Enable	Description
GeneralLog	Yes	Set the value of the log_level parameter to the following decimal value: 3000 This value configures CRM Desktop to log only warnings. For performance reasons, it is recommended that CRM Desktop log only warnings during normal operations. For more information, see "Parameters You Can Use with the General Log" on page 422.
ExceptionLog	No	For more information, see "Parameters You Can Use with the Exception Log" on page 425.
CrashDump	No	For more information, see "Parameters You Can Use with the Crash Log" on page 425.
SoapDump	Yes	For more information, see "Parameter You Can Use with the SOAP Log" on page 428.
SyncDump	Yes	For more information, see "Parameters You Can Use with the Synchronization Log" on page 429.

If you remove CRM Desktop at some point in the future, then it clears all registry settings during removal.

6 Assign your custom profile.

For more information, see "Assigning Logging Profiles for Siebel CRM Desktop" on page 120.

7 (Optional) You can use one of following items to distribute registry modifications across the network:

- Imported registration (.reg) files
- regini.exe
- Group policy
- System policy

Creating Installation Log Files for Siebel CRM Desktop

This topic describes how to create installation log files.

To create installation log files for Siebel CRM Desktop

1 Run the CRM Desktop installer with the following command line parameter:

```
"msiexec /lvx!* log_path /i CRMDesktop.msi"
```

where:

- *log_path* specifies where CRM Desktop stores the log file.

For example, run the following command:

```
"msiexec /lvx!* C:\admin logs\log.txt /i CRMDesktop.msi"
```

This command installs CRM Desktop and saves the installation logs in the log.txt file in the following directory:

```
C:\admin logs\
```

For more information, see "Using the Windows Command Line to Set Optional Parameters" on page 100.

Administering Logging on the Siebel Server

The EAI Object Manager performs logging for the business service methods, uses component events to log information for the Siebel Adapter business service, and creates log files that capture the following information on the Siebel Server:

- Input messages
- Time
- Detailed error messages
- Trace

EAI (Enterprise Application Integration) stores these logs in the following file:

Siebel Server install directory\eaiobjmgr_*language_code*.log

In UNIX, Siebel CRM Desktop stores these log files at the following location:

$SIEBEL_ROOT/enterprises/*enterprise_name*/*server_name* log

For more information, see the topic that describes event logs in *Siebel System Monitoring and Diagnostics Guide*.

To administer logging on the Siebel Server

- Set the EnableLogging parameter to true.

  For more information, see the topic about common event types for Application Object Manager diagnostics in *Siebel System Monitoring and Diagnostics Guide*.

Using Script to Modify Logging Levels

You can use the application.logger object in a script to modify logging levels.

To use script to modify logging levels

1. Use a JavaScript editor to open the application.js fileor the forms.js file.
2. Add the following code:

```
var log_settings = application.logger.settings.log_settings_1;
log_settings_2;
log_settings.save();
```

   where:

   - *log_settings_1* is set to one of the following values:
     - general_log_settings
     - exception_log_settings
     - crash_dump_settings
     - sync_dump_settings
     - soap_dump_settings
   - *log_settings_2* is set to one of the following values:
     - log_settings["enabled"] = 1
     - log_settings.set ("enabled", 1))

   For example:

```
var log_settings = application.logger.settings.general_log_settings;
log_settings["enabled"] = 1;
log_settings.save();
```

Troubleshooting Problems That Occur with Siebel CRM Desktop

This topic describes how to troubleshoot problems that occur with Siebel CRM Desktop. It includes the following topics:

For information about accessing log files, see "Where Siebel CRM Desktop Stores Data in the File System" on page 83.

Troubleshooting Problems That Occur When Siebel CRM Desktop Connects to the Siebel Server

To resolve a problem that occurs when Siebel CRM Desktop connects to the Siebel Server, look for it in the list of symptoms or error messages in Table 9. CRM Desktop uses the log.0000.txt file to log errors. It increments this log file name each time it creates another log file.

Table 9. Problems That Occur When CRM Desktop Connects to the Siebel Server

Symptom or Error Message	Solution
CRM Desktop creates an entry in the log.0000.txt file that is similar to the following error: `Synchronization canceled by error: siebel_service: failed attempting to connect to siebel` This error occurs if CRM Desktop cannot connect to the EAI Object manager.	You can make sure the following items are up and running: ■ Siebel Server ■ Server component for the EAI Object manager
CRM Desktop creates an entry in the log.0000.txt file that is similar to the following error: `Exception 'class siebel::siebel_cntr_exception' throwing: siebel_service: SOAP response has unexpected structure.` This error typically occurs if the architecture component that CRM Desktop calls returns an HTTP error instead of an expected SOAP message and CRM Desktop cannot parse the error. An Internal Server Error is an example of an HTTP error.	You can look for potential problems in the following logs: ■ SOAP log ■ Siebel Application Interface log ■ EAI log

Table 9. Problems That Occur When CRM Desktop Connects to the Siebel Server

Symptom or Error Message	Solution

CRM Desktop creates an entry in the log.0000.txt file that is similar to the following error:
 	
`Exception 'struct win32_exceptions::inet_cannot_connect' throwing: WinInet: Cannot connect to Internet!`
 	
`Synchronization canceled by error: WinInet: Cannot connect to Internet`
 	
This error occurs if CRM Desktop cannot connect to the Web server.
	You can do the following work:
	
	■ Make sure the Web server is running.
	
	■ Make sure the computer that runs the client can connect to the Web server.

When the user attempts to log in to the client, CRM Desktop displays a message that is similar to the following:
 	
`siebel_service: SOAP response has unexpected structure`
 	
It creates an entry in the log.0000.txt file that is similar to the following error:
 	
`Exception 'class siebel::siebel_cntr_exception' throwing: siebel_service: error occurred while retrieving meta data package: This user does not have an active package available.`
 	
This error occurs if the responsibility that is associated with the user is not associated with a customization package.
	Make sure the responsibility that is associated with the user is associated with a customization package. For more information, see "Creating and Publishing the Customization Package" on page 78.

Troubleshooting Problems That Occur During Synchronization

This topic describes how to troubleshoot problems that occur during synchronization. For more information, see "How Siebel CRM Desktop Handles Synchronization Errors" on page 74.

Resolving Exceeded Row Size Problems

The user might encounter an error that is similar to the following during the initial synchronization:

```
There were more rows than could be returned. Please refine your query to bring back fewer rows.
```

Siebel CRM Desktop returns this message because the number of records that the synchronization query attempts to return is larger than the allowable maximum.

To resolve an exceeded row size problem

1. Change the value for the DSMaxFetchArraySize parameter.

   For more information, see Step 2 on page 81.

2. Notify users to synchronize data.

3. If the problem persists, then get help from Oracle.

For more information, see "Getting Help From Oracle" on page 19.

Table 10. Problems That Occur When You Customize Siebel CRM Desktop

Symptom or Error Message	Solution
After you add a new field to a CRM Desktop form, CRM Desktop displays an error that is similar to the following: `Updating error of object name object on storage {CEDC3D49-DDBB-93A2-4992-E5B2CAE62932}: object_locked (Conversion of input Message to Property Set failed with the error : Cannot convert XML Hierarchy to Integration Object Hierarchy. (SBL-EAI-04111)` This error occurs if the definition of an integration object in the SRF does not match the definition of the custom CRM Desktop form.	You can do the following work: ■ Deploy your changes to the Siebel Runtime Repository.
After you add a new field to a form, CRM Desktop displays the form with the native Outlook form and then logs the following error: `Exception (struct ml::rethrowable_exception<struct resource_manager::resource_not_found>) encountered: "resource id 'lbl_location' not found"` This error occurs if you fail to add a localized value to the package_res.xml file.	You can do the following work: ■ Add the appropriate localized value to the package_res.xml file. For example: `<str key="lbl_location">Site:</str>` ■ Republish the package.

Table 10. Problems That Occur When You Customize Siebel CRM Desktop

Symptom or Error Message	Solution

You add a new validation rule to a form but you cannot save or close the form when you add a new record. CRM Desktop displays an error that is similar to the following:
 	
`Exception 'class scripting::execute_exception' throwing: Script run-time error. 'phonetest' is undefined (Microsoft JScript runtime error) Line: 6, Char 7`
 	
Consider the following custom validation rule:
 	
`<rule message="#msg_business_phone_validation"> <expression> <![CDATA[var phonetes = new RegExp(/^[+]?[0-9]{0,3}[\s]?[\(-.]?[0-9]{2,3}[\)-.]?[\s]?[0-9]{3}[-.]?[0-9]{2}[-.]?[0-9]{2}$/); item["Work Phone #"] != '' ? (item["Work Phone #"].match(phonetest) != null ? true : false) : true;]]> </expression> </rule>`
 	
This example includes a typographical error in the declaration of the phonetest variable. The variable name is missing the last t character.
 	
For more information, see "Validation Rules You Can Configure for Custom Forms" on page 162.
	You can do the following work:
	
	■ Fix the variable name.
	
	■ Republish the package. For more information, see "Republishing Customization Packages" on page 80.

After you add a new field to a form, CRM Desktop displays the form with the native Outlook form and then logs the following error:
 	
`Exception 'struct xml::parse_error' throwing: Errors occurred during parsing the document! Code: 0xc00ce504 Line: 4 Column: 4 Public ID: unknown System ID: unknown. A name was started with an invalid character.`
 	
The following message in this log indicates that the XML is not valid:
 	
`'struct xml::parse_error'`
	You can do the following work:
	
	■ Make sure the custom XML you created uses the correct XML format and does not contain errors.
	
	■ To help locate the invalid syntax, identify all changes. You can use a diff tool that compares the XML to a previously working package and then highlights the changes.
	
	■ To identify an XML format error, use an XML schema validation tool.

Table 10. Problems That Occur When You Customize Siebel CRM Desktop

Symptom or Error Message	Solution

You remove an existing field from a form. CRM Desktop displays the form momentarily but then the form goes blank. CRM Desktop logs the following errors:
 	
`14:46:21:262-2628: Exception 'class scripting::execute_exception' throwing: Script run-time error. 'undefined' is null or not an object (Microsoft JScript runtime error) Line: 77, Char 6`
 	
`14:46:21:262-2628: Exception (struct ml::rethrowable_exception<class scripting::execute_exception>) encountered: "Script run-time error. 'undefined' is null or not an object (Microsoft JScript runtime error) Line: 77, Char 6"`
 	
These errors occur if JavaScript references a control that does not exist or if the control is not spelled correctly.
	You can do the following work:
	
	■ To identify the control Id values, review the deleted elements.
	
	■ Perform a global search for code that references these values.
	
	■ Modify or remove all JavaScript code that references these control Ids.
	
	■ Republish the customization package.

8 Controlling Synchronization

This chapter describes how to control synchronization. It includes the following topics:

For other ways to administer options that affect synchronization, see “Using the Windows Registry to Control Siebel CRM Desktop” on page 105.

Controlling Synchronization Filters

This topic describes how to control synchronization filters. It includes the following topic:

Controlling the Object Types That Siebel CRM Desktop Displays in the Filter Records Tab

You can specify the filter settings for every user and deploy them when you install the CRM Desktop add-in. This option allows you to customize the filters and other settings that CRM Desktop displays. It displays the following objects in the Filter Records tab of the Synchronization Control Panel dialog box, by default:

- Contacts
- Accounts
- Opportunities
- Activities

You can customize the Synchronization Control Panel dialog box to display or not display these object types.

To control the object types that Siebel CRM Desktop displays in the Filter Records tab

1 Use an XML editor to open the connector_configuration.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Locate the type tag for the object you must display or hide.

For example:

```
type id="Opportunity"
```

3 Locate the view tag of the type you located in Step 2.

4 Set the suppress_sync_ui attribute to one of the following values:

- **True.** Hides the object.
- **False.** Displays the object.

For more information, see "View Tag of the Connector Configuration File" on page 465.

5 Save your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

6 (Optional) Add a new object to the list of objects that CRM Desktop displays on the Filter Records tab. You do the following:

a In the XML code, add a new type and view tag for the new object.

Service Requests is an example of a new object.

b Set the suppress_sync_ui attribute for this new object to false.

Controlling the Synchronization Exceptions Button In the Filter Records Tab

The Exclusions List allows the user to exclude an individual record from synchronization even if this record matches a defined filtering criteria. Siebel CRM Desktop does the following work:

1 It uses the following filters to identify the records that it must synchronize from the Siebel Server:

- **User filters.** Filters that the user creates.
- **Master filters.** A *master filter* is a type of predefined filter that the user cannot change. For example, a master filter can cause CRM Desktop to not synchronize a contact that includes an inactive status from Siebel CRM to Outlook.

2 Excludes the records that are in the Exclusions List. It excludes each record in the list only if some other record does not reference this record.

3 If the Exclusions List includes a record, and if no other record references this record, then CRM Desktop removes it from Outlook.

To control the Synchronization Exceptions button in the Filter Records tab

1. Use an XML editor to open the connector_configuration.xml file.
2. Locate the following features tag:

   ```
   </features>
   ```

3. Make sure the following attribute that resides in the features tag that you located in Step 2is set to true:

   ```
   enable_sync_exclusions
   ```

   This configuration displays the Synchronization Exceptions button on the Filter Records screen of the Synchronization Control Panel.

Examples of How Siebel CRM Desktop Uses the Exclusions List

Assume the following:

- Contact 1 references account 1.
- Contact 1 matches a filter but account 1 does not match a filter.

In this example, Siebel CRM Desktop synchronizes account 1 because contact 1 references it.

For another example, assume the following:

- Contact 1 references account 1.
- Contact 1 matches a filter and account 1 matches a filter.
- Outlook displays contact 1 in the Contacts folder of the CRM and Personal Contacts view and account 1 in the Accounts folder of the Siebel Accounts view. It does this after the first synchronization finishes.
- The user deletes account 1 and then CRM Desktop displays a prompt that is similar to the following:

  ```
  Are you sure you want to delete record(s) from Siebel and Microsoft Outlook?
  ```

- The user clicks No and then CRM Desktop moves account 1 to the Exclusions List. At the next synchronization, CRM Desktop synchronizes account 1 because contact 1 references it.

How Siebel CRM Desktop Adds Accounts, Contacts, and Opportunities to the Exclusions List

If the user deletes an account, contact, or opportunity in the Explorer view, then Siebel CRM Desktop displays a prompt that is similar to the following:

```
Are you sure you want to delete records from Siebel CRM and Outlook? Click Yes to
delete records from Siebel CRM and Outlook. Click No to delete records from Outlook
only and to not synchronize updates from Siebel CRM."
```

The user can choose one of the following values:

- **Yes.** CRM Desktop deletes the record in Outlook and then deletes it from the Siebel database on the Siebel Server during the next synchronization. If you enable delete confirmation, then it requests the user to confirm the deletion before it deletes the record from the Siebel database. For more information, see "Controlling How Siebel CRM Desktop Deletes Records During Synchronization" on page 151.
- **No.** CRM Desktop deletes the record from Outlook and adds it to the Exclusions List. If this record is associated with another record in Outlook through the lookup field, then Outlook displays it the next time the user synchronizes. A lookup field is a field that CRM Desktop uses to look up an object and then associate it with the current record. The account field on the activity record is an example of a lookup field. For example, assume the following:
  - A contact references a primary account.
  - The user deletes the account from the Explorer view and then clicks No at the confirmation prompt.
  - CRM Desktop removes the account from the Accounts folder, the Accounts field on the Contact form, and from the Accounts MVG dialog box.
  - The user synchronizes and then Outlook displays the record in the Accounts Lookup dialog box and in the Account field on the Contact form.

Controlling the Date Range in the Filter Records Tab

The user can choose a predefined value from the Value drop-down list in the CRM Desktop Filter - Edit Criterion dialog box to modify the date range that Siebel CRM Desktop uses in a synchronization filter. If the user changes the date range, then the number of records that match the filter also change. For example, the user can set the filter to synchronize activities that start a month ago. If the user changes this date, then CRM Desktop adjusts the number of activities it synchronizes. You can customize these predefined date ranges. For more information, see "Filters in the CRM Desktop Filter - Edit Criterion Dialog Box" on page 430.

Customizing the Predefined Date Ranges

You can customize the Predefined Date Ranges that CRM Desktop uses for the filters that it displays in the Value drop-down list in the CRM Desktop Filter - Edit Criterion dialog box.

To customize the predefined date ranges

1. Use an XML editor to open the connector_configuration.xml file.
2. Locate the sliding_dates_presets tag.
3. Change the value for a preset name.

   The following code comes predefined with CRM Desktop. To change a predefined date range, you change the number that the value attribute contains:

```
<sliding_dates_presets exact_date="#sliding_dates_exact"
relative="#sliding_dates_from_today">
  <preset name="#sliding_dates_yesterday">
```

```
      <value>-1</value>
    </preset>
    <preset name="#sliding_dates_tomorrow">
      <value>1</value>
    </preset>
    <preset name="#sliding_dates_month_ago">
      <value>-30</value>
    </preset>
    <preset name="#sliding_dates_month_ahead">
      <value>30</value>
    </preset>
    <preset name="#sliding_dates_today">
      <value>0</value>
    </preset>
  </sliding_dates_presets>
```

4 Change the corresponding resources in the package_res.xml file.

For more information, see "Controlling the Synchronization Intervals That Display in the Synchronization Tab" on page 137.

Controlling the Fields That Display in a Filter

You can use the IsHidden property of the field in the siebel_meta_info.xml file to control the fields that are available in a filter. For more information, see "Sharing a Calendar Appointment, Contact, or Task" on page 94.

To control the fields that display in a filter

1 Use an XML editor to open the siebel_meta_info.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Locate the first instance of the tag that defines the field you must modify.

For example the following tag in the Contact.Account object defines the Account Status field:

```
<field Name='Account Status' Label='Account Status' DataType='DTYPE_TEXT'
HasPicklist='yes' PicklistIsStatic='yes' PicklistCollectionType='ACCOUNT_STATUS'
PicklistTypeId='PickList_Generic' IOElemName='AccountStatus' />
```

3 Do one of the following:

- Make the field not available in filter criteria. You add the following property to this tag:

  ```
  IsHidden='yes'
  ```

- Make the field available in filter criteria. You add the following property to this tag:

  ```
  IsHidden='no'
  ```

Note that the DataType property must not be DTYPE_ID.

4 Repeat Step 2 through Step 3 for each of the other objects you must modify.

For example, the Contact.Account object also includes the account status.

Controlling Synchronization Time, Day, and Size

This topic describes how to control synchronization time, day, and size. It includes the following topics:

Overview of Controlling Synchronization Frequency

Application settings that are related to synchronization frequency determine how often and the kind of data Siebel CRM Desktop synchronizes. The user can specify frequency or you can configure it:

- The user can use the Synchronization tab of the Options dialog box to specify the interval that CRM Desktop uses to automatically start a synchronization. The user can double-click the CRM Desktop icon in the system tray or choose the Synchronize Now option from the options menu. The user can choose to synchronize all changes or only local changes. For more information, see "Synchronizing All Changes or Only Local Changes" on page 141.
- You can configure the application metadata to determine how often CRM Desktop synchronizes each object. You can configure data that changes less often on the Siebel Server to synchronize less frequently. Example data includes list of value and other reference data, such as employees or positions. For more information, see the description about the SyncFrequency tag in "Object Tag of the Siebel Meta Information File" on page 488.

For more information about controlling synchronization frequency, see "Controlling Synchronization Time, Day, and Size" on page 136.

Controlling the Synchronization Intervals That Display in the Synchronization Tab

This topic describes how to control the synchronization intervals that Siebel CRM Desktop displays in the Synchronization tab of the CRM Desktop - Options dialog box.

To control the synchronization intervals that display in the Synchronization tab

1. Use an XML editor to open the platform_configuration.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. In the initialization_script-CDATA section, add the following code:

```
<initialization_script>
  <![CDATA[
application.settings.set("CustomSyncPeriods", "10 = page-sync-periods-10-
minutes; 20 = page-sync-periods-20-minutes; 30 = page-sync-periods-30-minutes; 60
= page-sync-periods-1-hour; 1440 = page-sync-periods-1-day");
  ]]>
</initialization_script>
```

   where:

   - The following code describes an interval in minutes and a resource string that you add in Step 4:

```
10 = page-sync-periods-10-minutes
```

3. Save and then close the platform_configuration.xml file.

4. Use an XMLeditor to open the package_res.xml file and then add the following code:

```
<str key="page-sync-periods-10-minutes">Every 10 minutes</str>
<str key="page-sync-periods-20-minutes">Every 20 minutes</str>
<str key="page-sync-periods-30-minutes">Every 30 minutes</str>
<str key="page-sync-periods-1-hour">Once an Hour</str>
<str key="page-sync-periods-1-day">Once a Day</str>
```

   For more information, see "Controlling the Predefined Synchronization Intervals" on page 137.

5. (Optional) Set the default synchronization mode.

   For more information, see "Synchronizing All Changes or Only Local Changes" on page 141.

6. Republish the customization package on the Siebel Server.

   For more information, see "Republishing Customization Packages" on page 80.

Controlling the Predefined Synchronization Intervals

The following values come predefined for a Synchronize All Changes session:

- 60 minutes (One time for each hour)
- 720 minutes (Two times for each day)

- 1440 minutes (One time for each)
- 10080 minutes (One time for week)

The following values come predefined for a Synchronize Local Changes session:

- 10 minutes (Every ten minutes)
- 20 minutes (Every twenty minutes)
- 30 minutes (Every thirty minutes)
- 60 minutes (Every sixty minutes)

Resource strings in the package_res.xml file determine the predefined synchronization intervals. You can customize these intervals. For example, the following resource string controls synchronization to occur every 10 minutes:

```
<str key="page-sync-periods-10-minutes">Every 10 minutes</str>
```

How Siebel CRM Desktop Automatically Synchronizes If it Is Offline

If Siebel CRM Desktop is offline during a scheduled synchronization, then it does not run synchronization automatically. It runs the next synchronization according to the predefined schedule. For example, assume the automatic full synchronization is scheduled to run one time for each day. If the next synchronization occurs at 8:00 P.M., and if CRM Desktop is offline at 8:00 P.M., then it does not run the synchronization. The next automatic full synchronization occurs at 8:00 P.M. on the next day.

Controlling the Time and Day When Synchronizations Occur

You can control the time of day and the day when Siebel CRM Desktop synchronizes to manage the load that synchronization puts on the Siebel Server. For example, to avoid an overloaded server, you can delay synchronizations that might normally occur at 9:00 A.M. on a Monday morning after a weekend sales conference to a later time.

CRM Desktop delays synchronization for the number of milliseconds that you specify. It adds this delay before it sends each server request to the Siebel Server. The delays are summative. For example, if 500 requests exist, and if the delay is 1200 milliseconds, then it delays the synchronization for 10 minutes.

To control the time and day when synchronizations occur

1. Use an XML editor to open the siebel_meta_info.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434 and "Customizing Meta Information" on page 164.

2. Add a tag named delays_schedule.
3. In the delays_schedule tag, set the following attribute to meet your scheduling requirements:

```
request_delay DaySpec
```

For more information, see "Coding the Delays Schedule Tag" on page 139.

4 Repeat Step 3, as necessary.

Coding the Delays Schedule Tag

You must use the following format if you code the delays_schedule tag:

- To specify a day of the week, use MON, TUE, WED, THU, FRI, SAT, or SUN.
- To specify a day, use one of the following formats:
  - *yyyy*/*mm*/*dd*
  - *mm*/*dd*
  - *dd*

  where:
  - *yyyy*/*mm*/*dd* is the year, month, and day.
- You can use the DaySpec attribute to specify a delay in milliseconds.

If you include the day and date, then the date takes precedence. In the following example, Siebel CRM Desktop uses 2011/12/16. It does not use MON:

```
<request_delay DaySpec="MON" "2011/12/16" StartTime="12:00:00" EndTime="13:00:00"
DelayMsecs="6000" />
```

Example Code That Controls the Time and Day When Siebel CRM Desktop Synchronizes

The following code controls the time and day when Siebel CRM Desktop synchronizes:

```
<delays_schedule>
  <request_delay DaySpec="SUN" StartTime="10:22:17" EndTime="16:34:07" DelayMsecs="6000" />
  <request_delay DaySpec="MON" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="TUE" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="WED" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="THU" StartTime="12:00:00" EndTime="17:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="FRI" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="SAT" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="2009/09/01" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="2009/12/31" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
  <request_delay DaySpec="05/07" StartTime="12:00:00" EndTime="13:00:00" DelayMsecs="6000" />
</delays_schedule>
```

Controlling the Size and Type of Synchronized Records

You can control the size and type of records that Siebel CRM Desktop synchronizes. CRM Desktop comes with one predefined filter named Default filter. If you use this filter, then CRM Desktop downloads the following records from the Siebel Server to Outlook:

- All accounts, contacts, and opportunities that the user can view
- All activities that the user owns

- All notes that are related to the downloaded records
- All attachments that are related to the downloaded records that are no larger than 5 MB in size and that include one of the following file extensions:
  - doc
  - docx
  - xsl
  - xslx
  - msg
  - txt
  - rtf
  - html
  - ppt
  - pptx
  - pdf
  - mht
  - mpp
  - vsd

CRM Desktop downloads any child record that is related to a parent record that the user can view. This configuration allows the user to view the child in the appropriate window of the parent record. For example, the Contacts list displays the following information:

- All unshared contacts
- All shared contacts that the user can view

If the user attempts to open the detail form for a record that CRM Desktop does not allow the user to view, then it displays a read-only version of the detail form. This read-only form includes only some of the details.

To control the size and type of synchronized files

1. Configure the Filter Presets tag of the connector_configuration.xml file.

   For more information, see "Filter Presets Tag of the Connector Configuration File" on page 467, and "Example Code That Sets the Size and Type of Field" on page 468.

2. Notify the user to use the Filter Records tab of the Synchronization Control Panel.

   The Filter Records Tab allows the user to restrict the file type and maximum file size. The settings you configure in the Filter Presets tag of the connector_configuration.xml file override settings the user makes in the Filter Records tab. For example, assume you set a maximum file size of 5 MB, and the user sets this limit to 9 MB. In this situation, CRM Desktop restricts the file size to 5 MB.

Synchronizing All Changes or Only Local Changes

A user can right-click the CRM Desktop icon in the system tray and then choose one of the following menu items to manually start a synchronization:

- Synchronize All Changes
- Synchronize Local Changes

You can enable or disable Synchronize Local Changes. This menu item allows the user to synchronize only local changes to the Siebel Server. It does not synchronize all changes. The installer enables it, by default. CRM Desktop synchronizes only new and modified records from Outlook to the Siebel Server during a Synchronize Local Changes session. It compares each modified record to the corresponding record that resides on the Siebel Server. Collisions might occur during this synchronization. For more information, see "Resolving Synchronization Conflicts" on page 154.

To synchronize all changes or only local changes

1. Open the Windows Registry and then locate the following key:

   ```
   HKEY_CURRENT_USER\Software\Oracle\CRM Desktop
   ```

   For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

2. Right-click the EnablePeriodicSyncUpstream entry and then click Modify.
3. Enter one of the following values in the Value Data window of the Edit DWORD dialog box:
   - **1.** Enables Synchronize Local Changes. The default value is 1.
   - **0.** Disables Synchronize Local Changes.
4. Restart Microsoft Outlook.

   If you disable Synchronize Local Changes in Step 3, then CRM Desktop removes the Synchronize Local Changes menu item from the menu that it displays if the user right-clicks the CRM Desktop icon in the system tray. It also removes the Synchronize Recent Changes to Server Automatically check box and slider from the Synchronization tab on the CRM Desktop - Options dialog box.

Note the following:

- If a full synchronization and a local synchronization are scheduled to automatically run at the same time, then CRM Desktop runs the full synchronization and skips the local changes synchronization until the next time the local synchronization is scheduled to run. *Local synchronization* is a type of synchronization that synchronizes only local changes.
- If the full synchronization is scheduled to occur more frequently than the value that the UpstreamSyncMinThreshold parameter contains, then CRM Desktop disables local synchronization. The default value for the UpstreamSyncMinThreshold parameter is 600000 milliseconds, which is 10 minutes.

For more information, see "Controlling the Predefined Synchronization Intervals" on page 137.

Controlling the Number of Records That Synchronize

To produce the most desirable synchronization performance and scalability, and to maintain acceptable Outlook performance when the user works with Siebel CRM data, it is recommended that Siebel CRM Desktop synchronize no more than 10,000 records for a single user. If it synchronizes more than 10,000 records, then the following undesirable results might occur:

- Longer synchronization times
- More server resources required to support user synchronization sessions
- Slower Outlook performance when working with Siebel CRM data

To control the number of records that synchronize

1. With administrator privileges, log in to a Siebel CRM client that is connected to the Siebel Server.
2. Navigate to the Administration - Server Configuration screen, and then the Servers view.
3. In the components tab, choose EAI Object Manager.
4. Set the value for the Maximum Page Size parameter to the maximum number of records that CRM Desktop can synchronize for a single user.

   If CRM Desktop attempts to synchronize more records at run-time than the value you set, then it returns an error message to Outlook that indicates that the number of records requested is too large. For more information, see "Resolving Exceeded Row Size Problems" on page 126.

NOTE: By default, Siebel CRM Desktop allows only 20,000 sychronizations per object. For example, 20,000 synchronizations for Accounts or 20,000 synchronizations for Opportunities. Oracle does not recommend increasing this number because it could cause performance issues with Siebel CRM Desktop. Oracle recommends that you use master filters to reduce the number of objects synchronized by Siebel CRM Desktop. For more information about master filters, see "Master Filter Expression Tag of the Siebel Meta Information File" on page 492. If there is a compelling business need that prevents the use of master filters to decrease the records retrieved, the 20,000 limit can be adjusted up using the Windows Registry key at HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\LimitSnapshotObjects.

Configuring Siebel CRM Desktop to Disregard Erroneous Data That Users Modify

You can configure Siebel CRM Desktop to synchronize an object in only one direction. Using one-way synchronization can be useful to restore object types that the user modifies or removes in Outlook or to not allow the user to modify an object type. For example, employees or positions. One-way synchronization can also reduce the time required to synchronize. The synchronization engine can detect these modifications and refresh the object in Outlook without causing a collision. CRM Desktop does the following work for each object that it synchronizes in only one direction:

- Does not create a collision
- Does not display a delete confirmation

- Does not start a job on the Siebel Server

This configuration allows Siebel CRM Desktop to disregard erroneous data that the user enters or erroneous modifications that users make to data so that it does not synchronize these modifications to the Siebel Server. To do this, you configure an object to synchronize in only one direction. The example in this topic configures the Employee object to synchronize in only one direction.

A user might use some Outlook features that compromise the validation rules that CRM Desktop uses. For example, CRM Desktop cannot control how the user uses the native All Fields tab that Outlook displays on Outlook objects. This tab is a predefined Outlook feature that CRM Desktop cannot disable or intercept. A user might open a Contact record, click the All Fields tab, remove the values from the First Name and Last Name fields, and then save this record even though CRM Desktop requires these names.

For more information, see "Resolving Synchronization Conflicts" on page 154.

To configure Siebel CRM Desktop to disregard erroneous data that users modify

1. Use an XML editor to open the connector_configuration.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Locate the type tag of the object that you must configure for one-way synchronization.

   For example:

   ```
   type id="Employee"
   ```

3. Locate the synchronizer tag of the type tag you located in Step 2.

4. Set the read_only attribute of the type tag you located in Step 3 to true.

   For example:

   ```
   <type id="Employee">
     <view label="#obj_employee" label_plural="#obj_employee_plural"
   small_icon="type_image:User:16" normal_icon="type_image:User:24"
   large_icon="type_image:User:48" suppress_sync_ui="true">
     </view>
     <synchronizer name_format=":[:(First Name):] :[:(Last Name):]"
   frequency="604800" threshold="0" read_only="true">
       <links>
         <link>Primary Organization Id</link>
         <link>Primary Position Id</link>
       </links>
     </synchronizer>
   </type>
   ```

   To configure an object type to synchronize in only one direction, you set it to read only in Outlook. CRM Desktop synchronizes a read-only object in only one direction from the Siebel Server to the client.

5. Save your changes and then republish the customization package.

   For more information, see "Republishing Customization Packages" on page 80.

Controlling the Number and Size of Batch Requests

Siebel CRM Desktop sends requests to the Siebel Server in batches. One Web Service call is one batch. Each batch includes multiple commands. Each command can request multiple IDs. To avoid overloading the server, you can specify the number of IDs that CRM Desktop requests for each command and the number commands that each batch contains.

For example, assume the following occurs:

- CRM Desktop must get 300 accounts
- A batch contains only one command
- One command requests the IDs for all 300 accounts

The server fails in this situation. To avoid this problem, you can reduce the number of commands that each batch contains and the number of IDs that each command requests. CRM Desktop sets these values to 50, by default. If CRM Desktop uses this default value to process the 300 accounts, then it creates six separate commands where each command requests 50 account IDs. It creates one batch that includes these six commands.

To control the number and size of batch requests

1. Use an XML editor to open the siebel_meta_info.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434 and "Customizing Meta Information" on page 164.

2. Create or find the existing section under the root tag that contains the following name:

   ```
   common_settings
   ```

   For more information, see "Common Settings Tag of the Siebel Meta Information File" on page 487.

3. Set the following subtags:

   - **max_commands_per_batch.** Defines the maximum number of commands that each batch contains. The default value is 50.
   - **max_ids_per_command.** Defines the maximum number of IDs that the command requests. The default value is 50.

Controlling Other Configurations That Affect Synchronization

This topic describes how to control other configurations that affect synchronization. It includes the following topic:

- Controlling How Siebel CRM Desktop Deletes Records During Synchronization on page 151
- Resolving Synchronization Conflicts on page 154
- Resolving Synchronization Conflicts on page 154

Configuring How CRM Desktop Gets Updates That Occur During Synchronization

You can configure how Siebel CRM Desktop gets object updates from the Siebel Server that occur during a synchronization session. You can use this configuration to update a field that originates on the Siebel Server, such as an object Id or an automatically generated number. You can also use it to update a field Id that is involved with a calculated value that EAI (Enterprise Application Integration) inserts or updates in the Siebel database during synchronization.

The example in this topic adds a new Siebel ID field to the account object in the mapping schema. Siebel CRM Desktop uses this field internally to store the Siebel object Id in a text format. It updates this field during synchronization.

To configure how CRM Desktop gets updates that occur during synchronization

1. Use an XML editor open the siebel_meta_info.xml file.
2. Locate the tag of the object that represents the field that CRM Desktop must update during synchronization.

   For example, locate the following object tag:

   ```
   TypeId="Account"
   ```

3. Add the following tag to the tag that you located in Step 2:

   ```
   <field Name='Siebel ID' Label='Siebel ID' DataType='DTYPE_TEXT' BackUpd='any'
   IsFilterable='no' IsCalculated='yes' Formula=':[:(Id):]' />
   ```

   where:

   - BackUpd is the attribute that allows CRM Desktop to get updates that occur during synchronization. You can set it to one of the following values:
   - **insert.** Updates values that occur during an insert.
   - **update.** Updates values that occur during an update.
   - **any.** Updates values that occur during an insert or an update.

   During synchronization, CRM Desktop inserts the value that the Id field contains into the Siebel ID field.
4. Save and then close the siebel_meta_info.xml file.
5. Open the siebel_basic_mapping.xml file.
6. Add the following code anywhere in the file:

```
<field id="Siebel ID">
  <reader>
    <mapi_user>
      <user_field id="sbl Siebel ID" ol_field_type="1"></user_field>
        <convertor>
          <string/>
        </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Siebel ID" ol_field_type="1"></user_field>
        <convertor>
          <string/>
        </convertor>
    </Outlook_user>
  </writer>
</field>
```

Make sure that the value you use for the `user_field id` attribute is identical to the value that you use for the `field Name` attribute in Step 3 except that the `user_field id` attribute includes the `sbl` prefix.

7 Save your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Configuring CRM Desktop to Synchronize Private Activities

This topic describes how to configure Siebel CRM Desktop to synchronize private activities that the user creates in Outlook.

To configure CRM Desktop to synchronize private activities

1 Use an XML editor open the siebel_meta_info.xml file and then locate the following object:

```
<object TypeId="Action"
```

2 Remove the following code from the object you located in Step 1:

```
<master_filter_expr>
<![CDATA[
[Private] IS NULL OR [Private] = 'N'
]]>
</master_filter_expr>
```

This filter prevents CRM Desktop from synchronizing private activities.

3 Save your changes and then close the siebel_meta_info.xml file.

4 Use a JavaScript editor to open the sb_helpers.js file.

5 Locate the following code in the check_sharing_possibility function:

```
if (helpers.get_defaults(ctx.session) == null) message =
"msg_absent_first_sincronization_share";
else if (check_offline && !is_online(ctx)) message = "msg_cant_share_in_offline";
else if (item.type_id != "Mail" && item.snapshot.Private) message =
"msg_cant_sync_private";
```

If CRM Desktop attempts to share a private calendar appointment, then the Private flag in this code creates an error.

6 Replace the code you located in Step 5 with the following code:

```
if (helpers.get_defaults(ctx.session) == null) message =
"msg_absent_first_sincronization_share";
else if (check_offline && !is_online(ctx)) message = "msg_cant_share_in_offline";
//else if (item.type_id != "Mail" && item.snapshot.Private)
//message = "msg_cant_sync_private";
```

Commenting the last two lines of this code prevents CRM Desktop from creating an error if it attempts to share a private calendar appointment.

7 Save your changes and then close the sb_helpers.js file.

8 Use a JavaScript editor to open the forms.js file.

9 Locate the following code in the ol_item_form function:

```
if (!is_mail)
{
ctx.validator.add_custom(function(validate_ctx) {
return !(shared() && validate_ctx.snapshot()["Private"]) },
null,"msg_cant_sync_private");
}
```

This code validates that CRM Desktop cannot share a private calendar appointment. You must disable it.

10 Replace the code you located in Step 9 with the following code:

```
/*
if (!is_mail)
{
ctx.validator.add_custom(function(validate_ctx) {
return !(shared() && validate_ctx.snapshot()["Private"]) }, null,
"msg_cant_sync_private");
}
*/
```

11 Save your changes and then close the forms.js file.

12 Use a JavaScript editor to open the business_logic.js file, and then locate the following code:

```
if (ol.item_ex().get_property("Private") == true || ignore_ol_item(ol)).
```

This code determines if the private flag is set. If this flag is set, then this code stops CRM Desktop from processing.

13 Replace the code you located in Step 12 with the following code:

```
if (ignore_ol_item(ol))
```

14 Save and test your changes, and then republish the customization package.

Allowing Users to Open Top-Level Objects from the Control Panel

You can configure Siebel CRM Desktop to allow the user to open top-level objects from the Synchronization Control Panel if a synchronization issue occurs.

To allow users to open top-level objects from the Control Panel

1 Use a JavaScript editor to open the business_logic.js file.

2 Specify the top-level objects in the following code:

```
//Triggers
helpers.for_each(["Opportunity", "Account", "Contact", "Action", "Mail",
"Event", "Task"], function(type)
{
  scheme.triggers.add_simple_trigger(form_helpers.native_show, null, type, null,
"show");
  scheme.triggers.add_simple_trigger(form_helpers.native_show, null, type, null,
"cp_show");

});
```

The array for `helpers.for_each` contains the list of top-level objects that CRM Desktop can open. You must add the name of the object to this array. For example, for Service Request, you add the following code:

```
(["Opportunity", "Account", "Contact", "Action", "Mail", "Event", "Task", "top-
level_object"]
```

where:

- *top-level_object* specifies a top-level object that your customization includes. For example, use the following code for the Service Request object:

```
(["Opportunity", "Account", "Contact", "Action", "Mail", "Event", "Task",
"Service Request"]
```

Controlling the View Mode During Synchronization According to Object Type

This topic describes how to control the view mode that Siebel CRM Desktop uses during synchronization for an object type. The view modes that are configured in the client application metadata affect data access. You can configure each synchronization object with a different level of data access. This configuration is implemented as a view mode argument that CRM Desktop passes to the EAI Siebel Adapter business service during synchronization. It establishes basic access control in the client.

For example, data about opportunities is available to the sales representatives who are on the team for the opportunity. The default configuration for CRM Desktop specifies that the opportunity synchronization object must use the sales representative view mode. Several view mode arguments are available. For example, All, Organization, Sales Rep, or Personal. For more information, see "About the EAI Siebel Adapter Business Service" on page 29.

To control the view mode during synchronization according to object type

1. Use an XML editor open the siebel_meta_info.xml file.
2. Locate the `type` tag of the object that must use a view mode during synchronization.

   For example, locate the following tag:

   ```
   object TypeId='Account.Account_Note'
   ```

3. Locate the `viewmodes` subelement of the type tag you located in Step 2.
4. Set the `viewmodes` subelement using the following format:

   ```
   viewmodes view_mode_context1="view_mode_value"
   view_mode_context2="view_mode_value" view_mode_context3="view_mode_value"
   ```

   where:

   - *view_mode_context* is set to one of the following values:
     - **General.** Requests server records of synchronized object types that match the user synchronization filters with master filter restrictions applied. If you specify this value, then it overrides the value in the viewmode attribute. For more information, see Step 3 on page 108.
     - **Dedup.** Detects duplicate records when CRM Desktop must add records to Outlook or to the server database. If you do not specify this value, then it sets the value for the Deduplication view mode to the value that you set for the General view mode. For more information, see Resolving Synchronization Conflicts on page 154.
     - **QBID.** (Query By Id) Requests objects that CRM Desktop must synchronize to Outlook to maintain referential integrity because this object is related to a synchronized record. It requests this object even if the object does not match a user synchronization filter. You must specify a Query by Id for each object type that CRM Desktop queries by Id. The default value is All.
   - *view_mode_value* is set to one of the following values:

- All
- Sales Rep
- Personal
- Organization

For example:

```
viewmodes General="Organization" Dedup="All"
```

Example That Controls the View Mode During Synchronization According to Object Type

The following code controls the view mode that Siebel CRM Desktop uses during synchronization for the Account.Account_Note object type:

```
<object TypeId='Account.Account_Note' Label='#obj_account_note'
LabelPlural='#obj_account_note_plural' EnableGetIDsBatching='true'
IntObjName='CRMDesktopAccountIO' SiebMsgXmlElemName='AccountNote'
SiebMsgXmlCollectionElemName='ListOfAccountNote' >
  <viewmodes General="Organization" Dedup="All" />
  <open_with_url_tmpl>
    <![CDATA[
      :[:(protocol):]://:[:(hostname):]::[:(port):]/sales/_:[:(lang):]/
?SWECmd=GotoView&SWEView=Account+Note+View&SWERF=1&SWEHo=:[:(hostname):]&SWEBU=1&SWEAp
plet0=Account+Entry+Applet&SWERowId0=:[:(parent_id):]&SWEApplet1=Account+Note+Applet&S
WERowId1=:[:(own_id):]
    ]]>
  </open_with_url_tmpl>
  <extra_command_options>
    <option Name='PrimaryKey1M' Value='Id' />
    <option Name='ForeignKey1M' Value='Account Id' />
    <option Name='Cardinality' Value='1M' />
    <option Name='ServerServiceVersion' Value='2' />
  </extra_command_options>
  <field Name='Account Id' Label='Account Id' DataType='DTYPE_ID' IsNullable='no'
IsFilterable='no' IsRefObjId='yes' RefObjTypeId='Account' RefObjIsParent='yes'
IsPartOfUserKey='yes' IOElemName='AccountId' />
  <field Name='Conflict Id' Label='Conflict Id' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes' IOElemName='ConflictId' />
  <field Name='Created' Label='#fld_account_account_note@created'
DataType='DTYPE_DATETIME' IsPartOfUserKey='yes' IOElemName='Created' />
  <field Name='Created By' Label='Created By' DataType='DTYPE_ID' IsFilterable='no'
IsRefObjId='yes' RefObjTypeId='Employee' IOElemName='CreatedBy' />
  <field Name='Created By Name' Label='#fld_account_account_note@created_by_name'
DataType='DTYPE_TEXT' IOElemName='CreatedByName' />
  <field Name='Created Date' Label='Created Date' DataType='DTYPE_UTCDATETIME'
IsHidden='yes' IOElemName='CreatedDate' />
  <field Name='DS Updated' Label='DS Updated' DataType='DTYPE_DATETIME'
IsFilterable='no' IsHidden='yes' IsTimestamp='yes' IOElemName='DBLastUpd' />
  <field Name='Id' Label='Id' IsPrimaryKey='yes' DataType='DTYPE_ID' IsFilterable='no'
IsHidden='yes' IsPartOfUserKey='yes' IOElemName='Id' />
  <field Name='Mod Id' Label='Mod Id' DataType='DTYPE_ID' IsFilterable='no'
```

```
IsHidden='yes' IOElemName='ModId' />
  <field Name='Note' Label='#fld_account_account_note@note' DataType='DTYPE_NOTE'
IOElemName='Note' />
  <field Name='Note Type' Label='#fld_account_account_note@note_type'
DataType='DTYPE_TEXT' HasPicklist='yes' PicklistIsStatic='yes'
PicklistCollectionType='FS_NOTE_TYPE' PicklistTypeId='List_Of_Values'
IOElemName='NoteType' />
  <field Name='Private' Label='Private' DataType='DTYPE_BOOL' IsHidden='yes'
IOElemName='Private' />
  <field Name='Updated' Label='Updated' DataType='DTYPE_DATETIME' IsHidden='yes'
IOElemName='Updated' />
  <field Name='Updated By' Label='Updated By' DataType='DTYPE_ID' IsFilterable='no'
IsHidden='yes' IOElemName='UpdatedBy' />
</object>
```

Controlling How Siebel CRM Desktop Deletes Records During Synchronization

You can control how Siebel CRM Desktop deletes records during a synchronization. The *delete confirmation* feature allows the user to cancel, during synchronization, a deletion that the user made in Outlook. If you enable this feature, then CRM Desktop does the following work:

- Displays the Confirm Synchronization tab on the Synchronization Control Panel dialog box.
- Uses the Confirm Synchronization tab to allow the user to confirm the delete operation. If the user deletes records in Outlook, then CRM Desktop displays the Confirm Synchronization tab during synchronization. If the user confirms, then it removes the deleted records from the Siebel database on the Siebel Server. For more information, see "How the Number of Deleted Records Determines Delete Confirmation" on page 152.

To control how Siebel CRM Desktop deletes records during synchronization

1. Use an XML editor open the connector_configuration.xml file.
2. Configure CRM Desktop to display the Confirm Synchronization tab on the Synchronization Control Panel dialog box:
   a. Add the following code to the root tag of the connector_configuration.xml file:

      ```
      <features deletion_confirmation_mode="enable"/>
      ```

      For more information, see "Setting the Delete Confirmation Mode Attribute" on page 152.

   b. Specify the objects that CRM Desktop displays in the Delete on Siebel list in the Confirm Synchronization tab.

      For more information, see "Specifying the Type of Object the User Can Confirm for Deletion" on page 153.
3. Configure CRM Desktop to suppress display of the Confirm Synchronization tab in the Synchronization Control Panel dialog box. You add the following code to the root tag of the connector_configuration.xml file:

```
<features deletion_confirmation_mode="suppress"/>
```

How the Number of Deleted Records Determines Delete Confirmation

The number of records that the user deleted determines if Siebel CRM Desktop displays the Confirm Synchronization tab. For example:

- If the user deletes three or more accounts, ten or more contacts, or five or more opportunities, then CRM Desktop displays the Confirm Synchronization tab.
- If the user deletes only one or two accounts, then CRM Desktop does not display the Confirm Synchronization tab.

For more information, see "Threshold That Siebel CRM Desktop Uses to Display the Confirm Synchronization Tab" on page 432.

Setting the Delete Confirmation Mode Attribute

You use the deletion_confirmation_mode attribute of the connector_configuration.xml file to control the Confirm Synchronization tab on the Synchronization Control Panel dialog box.

Table 11 describes the values you can use for the deletion_confirmation_mode attribute.

Table 11. Values for the Delete Confirmation Mode Attribute

Value	Description
suppress	Disables delete confirmation. Displays the Confirm Synchronization tab in the Synchronization Control Panel.
enable	Enables delete confirmation. Displays the Confirm Synchronization tab in the Synchronization Control Panel. Displays the Revert Deletions button and the Accept Deletions button.
revert_only	Displays the Confirm Synchronization tab in the Synchronization Control Panel but enables only the Revert Deletions button. Displays but does not enable the Accept Deletions button. This is the default setting.
user_confirm	Displays the Confirm Synchronization tab in the Synchronization Control Panel but displays only the Accept Deletions button. Displays but does not enable the Revert Deletions button.

The following example sets the deletion_confirmation_mode attribute to revert_only. The ellipses (. . .) indicates code that this book omits from this example for brevity:

```
<root>
  <features deletion_confirmation_mode="revert_only"
  . . .
</features>
```

Specifying the Type of Object the User Can Confirm for Deletion

You can specify the type of object that Siebel CRM Desktop displays in the Delete on Siebel list in the Confirm Synchronization tab. For example, you can specify CRM Desktop to display only opportunity records.

To specify the type of object the user can confirm for deletion

1. Use an XML editor to open the connector_cinfiguration.xml file.
2. Locate the object type that CRM Desktop must display in the Delete on Siebel list in the Confirm Synchronization tab list.

   For example, for opportunities, you locate the following object type:

   ```
   type id="Opportunity"
   ```

3. In the object you located in Step 2, add the synchronizer tag.

   For example, add the following tag:

   ```
   <synchronizer name_format=":[:(Name):]" threshold="5">
   ```

   For more information, see "Setting the Synchronizer Tag" on page 153.

4. Repeat Step 2 through Step 3 for each type of object that CRM Desktop must display in the Delete on Siebel list.

Setting the Synchronizer Tag

The synchronizer tag in the connector_configuration.xml file controls the type of records that Siebel CRM Desktop displays in the Delete on Siebel list in the Confirm Synchronization tab list. It includes a threshold attribute. This attribute is set to 5 in the following example. It causes CRM Desktop to display the Confirm Synchronization tab only if the user deleted five or more opportunities since the last synchronization:

```
<type id="Opportunity" state_field="ObjectState">
  <view label="#obj_opportunity" label_plural="#obj_opportunity_plural"
small_icon="type_image:Opportunity:16" normal_icon="type_image:Opportunity:24"
large_icon="type_image:Opportunity:48"></view>
    <synchronizer name_format=":[:(Name):]" threshold="5">
      <links>
      </links>
      <natural_keys>
      </natural_keys>
    </synchronizer>
</type>
```

Table 12 describes the values for the threshold attribute of the synchronizer tag.

Table 12. Values for the Threshold Attribute of the Synchronizer Tag

Value	Description
0	Do not display delete confirmation for the object type.
1	Display delete confirmation for the object type.
Any value greater than 1	If you specify any value that is greater than one, then do the following: ■ If the value you specify is greater than the number of deleted objects, then do not display delete confirmation for the object. ■ If the value you specify is less than or equal to the number of deleted objects, then display delete confirmation for the object.

Resolving Synchronization Conflicts

This topic describes how to configure Siebel CRM Desktop to resolve synchronization conflicts. It includes the following topics:

Overview of Synchronization Conflicts

Auto Resolver is a Siebel CRM Desktop feature that allows you to configure CRM Desktop to automatically resolve a synchronization conflict instead of allowing the user to manually resolve this conflict. One of the following synchronization conflicts might occur when Siebel CRM Desktop synchronizes local data with data from the Siebel Server:

- Duplicating a record
- Simultaneously updating a record on the client and on the server
- Updating a record on the client and deleting this record on the server
- Updating a record on the server and deleting this record on the client

Consider the following items:

- **Activity processing.** Assume a user receives an event that CRM Desktop automatically shares. The user then downloads the same event from the Siebel Server during a synchronization. This situation might cause a *duplication conflict*, which is a type of conflict where two separate records include exactly the same information. You can configure the Auto Resolver so that CRM Desktop uses the record from the client database or from the Siebel Server database as the primary record, and then uses the data in this primary record to update the data in the nonprimary record.

- **Data update.** Assume a user updates a date in a field on the client, and that the Siebel Server updates this same date on the server. In this situation, an *update conflict* occurs during synchronization between the client and server databases.

This topic uses the following terms:

- *remote* identifies the Siebel Server.
- *local* identifies the client.

It is recommended that you configure autoresolve for each individual field, and only when necessary. It is recommended that you do not configure Siebel CRM Desktop to resolve all conflicts. Sometimes the client will include the correct value and sometimes the Siebel Server will include the correct value. Using autoresolve might result in Siebel CRM Desktop maintaining an incorrect value. If the client or server might include the correct value, then it is recommended that you allow the user to choose the correct value.

How the Auto Resolver Resolves Conflicts

The Auto Resolver Manager (ar_manager) includes a function that it assigns when an on_conflict event occurs. If this event occurs, then the synchronizer indicates the conflict and then provides information about this conflict. This conflict information might include the following items:

- Local object Id
- Remote object Id
- Set of local fields
- Set of remote fields
- Conflict type as a duplication conflict or a general conflict
- And so on

The Auto Resolver does the following work to resolve a conflict:

1. Examines the first rule that the autoresolver.js file contains.
2. If the rule that it examines in Step 1:
   - **Is applicable for the conflict.** The Auto Resolver attempts to apply this rule to this conflict. If this rule:
     - **Resolves the conflict.** The Auto Resolver determines if another conflict exists. If it finds another conflict, then it repeats Step 1 and Step 2 until it processes all conflicts. If it does not find another conflict, then it quits this process.
     - **Does not resolve the conflict.** The Auto Resolver examines the next rule that the autoresolver.js file contains. It continues to process these rules sequentially in the order that they occur in the autoresolver.js file until it resolves the conflict or until it processes all rules.
   - **Is not applicable for the conflict.** The Auto Resolver examines the next rule that the autoresolver.js file contains.

3 If the Auto Resolver applies all rules but the conflict remains, then this conflict remains unresolved, and the Auto Resolver displays it in the Conflicts tab of the Control Panel dialog box of the CRM Desktop add-in.

Configuring Siebel CRM Desktop to Resolve Synchronization Conflicts

This topic describes how to configure Siebel CRM Desktop to resolve synchronization conflicts.

To configure Siebel CRM Desktop to resolve synchronization conflicts

1 Use a JavaScript editor to open the autoresolver.js file.

2 Navigate to the following section:

```
//Opportunity
```

3 Add the following code:

```
ar_manager.add_rule({ type: "object_type", field: " field_ID ", resolution_fn:
ar_helpers.resolution_resolve_to("source", boolean) });
```

where:

- **ar_manager.** Is the Auto Resolver manager. It is an interface to the C++ code that CRM Desktop runs to resolve the conflict.
- **add_rule.** Is the C++ method that ar_manager sends to the C++ code.
- **type.** Identifies the object type where CRM Desktop applies the rule.
- **field.** Identifies the field that contains the data that CRM Desktop uses to resolve the conflict.
- **resolution_fn.** Identifies the rule function that CRM Desktop passes to the C++ code. This function returns a string value that indicates if the Auto Resolver resolved the conflict.
- **resolution_resolve_to.** Defines the rule. The rule is configured in the autoresolve_helpers.js file. It includes the following values:
  - *source* is remote or local, where remote is the Siebel Server and local is the client. It identifies the source that CRM Desktop uses to resolve the conflict.
  - *boolean* is true or false.

For example:

```
ar_manager.add_rule({ type: "Opportunity", field: " Account Id ", resolution_fn:
ar_helpers.resolution_resolve_to("local", true) });
```

In this example, if a synchronization conflict exists in the Account field of the Opportunity form, then this rule uses the value in the Account field from the client database as the permanent value only if the resolution_fn returns the following value:

```
local
```

For example:

```
ar_manager.add_rule({ type: "Opportunity", field: " Account Id ", resolution_fn:
ar_helpers.resolution_resolve_to("local", false) });
```

In this example, if no values exists for this field on the client, then the Auto Resolver does not apply a resolution.

For another example, you add the following code to the //Opportunity section:

```
ar_manager.add_rule({ type:"Action", field:"Email To Line", resolution_fn:
ar_helper.resolution_resolve_to("local", false) });
```

Examples of Auto Resolver Rules

This topic describes some of the predefined rules that Auto Resolver uses. You can use these rules as a guide when you create your custom rules. The autoresolver.js file contains these predefined rules. You must not modify any predefined rule.

Rule That Resolves Association Conflicts

The following predefined rule resolves association conflicts:

```
/*** ASSOCIATION AUTORESOLVER ***/

ar_manager.add_rule(new ar_helpers.associations_resolve_rule(ar_ctx))
```

where:

- `ar_helpers.associations_resolve_rule(ar_ctx)` identifies the code that CRM Desktop uses for the rule.
- `ar_ctx` identifies the list of the required objects. For example:

  ```
  var ar_ctx = {
    "util": util,
    "conflicts_manager": conflicts_manager,
    "application": application,
    "data_model": business_logic.create_siebel_meta_scheme2()
  ```

  where:

  - `util` is a C++ object that contains a number of methods.
  - `conflicts_manager` is the object that CRM Desktop creates in the interface to the C++ code that the Auto Resolver uses.
  - `application` is a C++ object that contains a number of methods.
  - `data_model` is a C++ object that contains a number of methods.
  - You cannot modify a reference to any of these C++ objects.

Rule That Resolves Conflicts for Objects Types

The following predefined rule resolves conflicts for certain object types regardless of field values:

```
/*** TYPE RULES ***/

ar_manager.add_rule({ types: ["Action", "Account", "Contact", "Opportunity"],
resolution_fn: ar_helpers.condition_update_delete_conflict_stopper });
```

where:

- `types: ["Action", "Account", "Contact", "Opportunity"]` is a list of object types where CRM Desktop applies the rule.
- `condition_update_delete_conflict_stopper` is the rule that CRM Desktop runs. The autoresove_helpers.js file contains this rule.

Rule That Resolves Conflicts for Fields

The following predefined rule resolves conflicts for certain object types, including field values:

```
/*** FIELD RULES ***/
// Not synchronized
ar_manager.add_rule({ types: ["Contact", "Opportunity", "Account"], field: "Primary
Position Id", resolution_fn:
ar_helpers.resolution_resolve_not_synced_field_to("remote", false) });
```

where:

- `types: ["Contact", "Opportunity", "Account"]` is a list of object types where CRM Desktop applies the rule
- `field: "Primary Position Id"` identifies the field that contains the data that CRM Desktop uses to resolve the conflict.
- `resolution_resolve_not_synced_field_to` is a rule that the Auto Resolver applies only for an object that CRM Desktop has not synchronized to the client. This situation can occur in a deduplication conflict.

The Auto Resolver applies this rule only for an object that it has not synchronized to the client. This situation can occur in a deduplication conflict.

9 Customizing Siebel CRM Desktop

This chapter describes how to customize Siebel CRM Desktop. It includes the following topics:

Overview of Customizing Siebel CRM Desktop

This topic describes an overview of how to customize Siebel CRM Desktop. It includes the following topics:

You can customize Siebel CRM Desktop to do the following:

- Extend the set of data that is available to the user.

- Add custom business logic to support the work that the user performs.

CRM Desktop can synchronize this information with Siebel CRM data in Microsoft Outlook. You can also make the data available to support offline usage if the user possesses limited or no connection to the Internet. You can create an interface where CRM Desktop stores the information the user requires to complete a business process. This configuration allows the user to work in a single application rather than navigating between multiple applications.

CAUTION: XML files and JavaScript files contain predefined configuration information that is critical to Siebel CRM Desktop operations. If you modify any of these files, then you must be very careful. Only make the minimal modifications that you require. It is recommended that you unit test each change you make.

For more information, see the following topics:

- Files That the Customization Package Contains on page 434
- Metadata That Siebel CRM Desktop Uses on page 32
- Troubleshooting Problems That Occur When You Customize Siebel CRM Desktop on page 269

Customizing How Siebel CRM Desktop Processes Objects

You can use Siebel CRM Desktop functionality, such as deduplication or data validation, to add custom logic for object processing in Microsoft Outlook. You can also use JavaScript to create your own custom logic. You can use JavaScript to create, delete, or modify objects. You can migrate your Siebel CRM data to Outlook, including logic that supports a business process.

You can use a standard CRM Desktop form to display Siebel CRM objects, or you can create a completely new and custom form. A custom form can include native Outlook controls, such as a text box, or new controls that you develop for CRM Desktop, such as a lookup or a multiselect list You can you use JavaScript to implement these forms through a custom toolbar.The user can use Outlookview controlson custom forms to display relationships between Siebel CRM objects. These views are fully configurable and support the functionality of native Outlook views. You can specify the default views that CRM Desktop uses to display Outlook objects

For more information, see Resolving Synchronization Conflicts on page 154.

Customizing Field Mapping

The siebel_basic_mapping.xml file describes objects you can add to Microsoft Outlook. It defines mapping between a Siebel CRM field and the Outlook field. You can also extend a set of fields that native Outlook objects reference. Each object description includes the following information:

- The forms_xx file defines the forms. The siebel_basic_mapping file identifies the form that each object uses.
- The icons that Siebel CRM Desktop uses to display the object. CRM Desktop uses these icons in Outlook views.
- The folder name for the object in the Outlook navigation pane.
- A set of custom Outlook views that CRM Desktop applies to the Outlook view.

For more information, see "XML Code That Maps a Field" on page 457.

Customizing Synchronization

The connector_configuration.xml file identifies the objects that Siebel CRM Desktop synchronizes. It includes synchronization settings that affect the following types of filters:

Deduplication filters. It uses special criteria that the connector_configuration.xml file describes to determine if an item already exists in the Siebel database when Siebel CRM Desktop synchronizes an item from Outlook to the Siebel database. For more information, see "How Siebel CRM Desktop Avoids Duplicate Data" on page 73 and Resolving Synchronization Conflicts on page 154.

- **Preset filters.** Defines preset filters for a custom synchronization. These presets help the user to synchronize only the data that the user requires. You can configure any filter preset as the default.

For more information, see "XML Code That Customizes Synchronization" on page 462.

Customizing Forms

The forms_xx.xml file includes the definitions that Siebel CRM Desktop uses for forms that you customize. This file allows you to customize forms, remove fields, change field names, and set custom list views. For more information about the forms_xx.xml file, see "Files in the Customization Package" on page 434

A form tag describes each form. CRM Desktop includes the following customization capabilities for these files:

- Changing field labels.
- Removing fields from the form that are not applicable to the work environment.
- Identifying fields on forms.
- Designating field type. CRM Desktop supports the following:
  - Native Outlook field types, such as text box or check box
  - Custom CRM Desktop controls, such as currency control, lookup, and multiselect list
- Positioning a field on the form.
- Modifying custom CRM Desktop controls.
- Creating entirely new forms that use the look and feel of Outlook. To enter data in these forms, CRM Desktop uses fields that the user defines.
- Programmatically applying custom behavior and logic to a form or field.
- Setting Outlook view controls on a custom form to display relationships between Siebel CRM objects.

- Modifying interface elements, such as text in a system message, a dialog box, a label, or a caption.

For more information, see "XML Code That Customizes Forms" on page 469.

Validation Rules You Can Configure for Custom Forms

The forms.js file includes a description of the validation rules that Siebel CRM Desktop uses for the object that it displays on any form. You can configure CRM Desktop to examine the format of information that the user enters in a field and then inform the user if this information does not adhere to this format. Validation uses JavaScript functions, so you can combine these functions with JavaScript RegEx (JavaScript Regular Expressions)to configure a wide variety of validation.For more information, see "Defining Validation Rules" on page 246.

Business Logic That You Can Configure for Custom Forms

You can use JavaScript to implement business logic with the forms_xx.xml file.JavaScript capabilities in Siebel CRM Desktop allow you to access any field of an object, or any property of the Outlook form. You can use this access to run code on a specific event, such as opening a form, saving a form, and so on. You can configure CRM Desktop to fill fields automatically, format field values automatically, and disable or enable a control, depending on criteria.

Customizing Toolbars

The toolbars.xml file describes the custom toolbars that Siebel CRM Desktop displays on a native Outlook form, a custom form, or in the Outlook window with programmable actions on the toolbar. You might be able to use some of the basic functionality that comes predefined with CRM Desktop to meet your basic requirements. For more information, see "XML Code That Customizes Toolbars" on page 480.

Customizing Dialog Boxes

The dialogs.xml file defines the layout for a custom dialog box that Siebel CRM Desktop uses, such as the dialog box that it uses with an MVG, an address, or for email processing. The dialogs.xml file is an extension of the forms_xx.xml file. It includes the same structure as the dialogs root tag instead of the forms tag in the forms_xx.xml file. For more information, see "XML Code That Customizes Dialog Boxes" on page 482.

To customize the behavior of the MVG dialog box, you can modify the XML file that CRM Desktop includes in the customization package.You can customize the following behaviors of the MVG dialog box:

- Object that it uses to create an association in the MVG dialog box.
- Fields that it specifies for the association.
- Format of the fields that it specifies for the association.

- Format it uses to display the Primary record. The following is the default format:

  position (name)

  For example, District Manager 1 (Wasaka Takuda). You can specify the fields that CRM Desktop displays and the order it uses to display them.

- Fields that it displays in the Outlook view that represent the associations you create. You can configure CRM Desktop to display only the record attributes. For example, if it stores the Employee name in the position record, then it can display only the Employee name for the position.
- The user permissions. The customization package uses security validation rules to describe the user permissions that work with the MVG. For example, if the user is not the primary user, then the user cannot delete users from the collection because CRM Desktop turns off the delete button for any user who is not a primary user.
- Behavior of a lookup control. A lookup control searches through the File As field of associated objects to search for a record.
- Hide the details of the parent record, such as the Opportunity Name.
- Add or remove association attributes for the associated record.
- Use an OK button instead of the Save and Close icon.

For more information, see "Customizing Picklists" on page 271.

Customizing Views

The views.xml file describes the view configurations that Siebel CRM Desktop uses in CRM Desktop forms and in Outlook views. Each Outlook view uses an XML file that is native to Outlook. CRM Desktop modifies these native Outlook views so that it can display Siebel CRM data. For example, it predefines the Siebel Accounts view in Outlook that it displays in the Accounts folder. For more information about:

- Customizing a view in CRM Desktop, see "XML Code That Customizes Views" on page 483.
- Customizing a view that is native in Outlook, see the Microsoft Outlook documentation on the Microsoft Web site.

Customizing the SalesBook Control

The lookup_view_defs.xml file sets configuration options for the SalesBook control in Microsoft Outlook. This control defines references between objects. Siebel CRM Desktop uses it primarily in lookup controls. You can use the lookup_view_defs.xml file to specify the objects that it makes available through each SalesBook control. You can also set filters for the objects that you must display in the SalesBook control. For more information, see "XML Code That Customizes the SalesBook Control" on page 484.

Customizing Meta Information

The siebel_meta_info.xml file includes the following meta information:

- A description of the object types that Siebel CRM Desktop supports.
- Fields that are defined and the type for each field.
- XML element names that CRM Desktop uses to build or parse a Siebel message. It uses information about the relations between objects from the file for the Siebel message.
- The definition of each object that CRM Desktop supports. This definition includes a unique name, an XML element, and an XML collection element that CRM Desktop uses in a Siebel message.

Every object field includes a name, a Siebel data type, and an XML element. CRM Desktop uses this information in a Siebel message to display values and filters for that field.

For more information, see"XML Code That Provides Meta Information" on page 486.

Customizations That Oracle Does Not Support

This topic describes customizations that Oracle does not support.

Files That You Must Not Modify

Table 13 describes the files that implement predefined functionality. You must not modify these files under any circumstances.

Table 13. Files You Must Not Modify

File Name	Description
actions_support.js	Includes toolbar action support functions.
activity_processor.js	Includes business logic for activity processing for a Calendar event, task, or email.
autoresolve_helpers.js	Includes helpers that the autoresolver.js file uses.
data_model.js	Includes logic for handling relations between objects and joint fields. The business_logic.js file initializes this data model.
form_helpers.js	Includes helpers for event handling in the client.
helpers.js	Includes utility functions that CRM Desktop reuses in different files.
idle.js	Includes the Idle processing manager and common idle handlers.
md5.js	Implements the MD5 Message Digest Algorithm.
mvg_dialogs.js	Includes MVG control logic.
raw_item_functions.js	Includes General functions that access Outlook items that bypass the C++ wrapper.

Table 13. Files You Must Not Modify

File Name	Description
recurrence_processing.js	Includes transformation functions that the activity_processor.js file uses for repeating patterns that occur between Siebel CRM and Outlook.
sb_helpers.js	Includes utility functions that different files use but that are specific to Siebel CRM.
security_manager.js	Includes security model descriptions and security manager descriptions that the security_utils.js file uses.

Functions That You Must Not Modify

Table 14 describes functions in the business_logic.js file that you must not modify. Activity processing includes a complex scripting model that supports a Calendar event, task, and email that is native in Outlook. It correlates these items with Siebel CRM activities. You must not modify these functions. For more information, see "Customizing Form Functions" on page 167.

Table 14. Functions You Must Not Modify

Function Name
function safe_get_sender_address(item)
function get_mail_recipients_by_priority(ctx, mail)
function activity_processor(ctx)
var util_get_proxy_from_association = function(ctx, association)
var product_activity_descriptor = function()
var process_event_modifications = function(ctx, ap, modification_ctx)
var activity_processor_extention = function(ctx, ap)

Using Siebel Tools

This topic describes some of the basic tasks you might perform in Siebel Tools if you customize Siebel CRM Desktop. For more information, see *Using Siebel Tools*.

Checking Out Projects in Siebel Tools

This topic describes how to check out a project in Siebel Tools.

To check out a project in Siebel Tools

1. Identify the project you must check out.

   For example, if you must modify an applet, then note the value in the Project property for the applet.

2. In the Object Explorer, click Project, and then query the Name property for the project you identified in Step 1.

3. Make sure the Locked property contains a check mark.

Displaying Object Types in Siebel Tools

You can display object types in the Object Explorer that you use to configure Siebel CRM Desktop.

To display object types in Siebel Tools

1. Open Siebel Tools.
2. Choose the View menu and then click Options.
3. Click the Object Explorer tab.
4. Scroll down through the Object Explorer Hierarchy window until you locate the Integration Object tree.
5. Make sure the Integration Object tree and all child objects of the Integration Object tree include a check mark.

   If all child objects in the Integration Object tree include a check mark, then Siebel Tools displays a black check mark with a white background for the tree.

6. Repeat Step 4 for any other object types you must modify.
7. Click OK.

Customizing Form Handlers

This topic describes how to customize form handlers. It includes the following information:

You can also specify form handlers that do validation. For more information, see “Validating the Data That Users Enter” on page 225.

Overview of Customizing Form Handlers

A *form handler* is a JavaScript function that CRM Desktop defines in the forms.js file. Each form that displays a top-level object, such as an account or contact, includes a form handler. If the user double-clicks one of these objects, then CRM Desktop does the following work:

- Determines if a form handler exists for this object.
- If a form handler exists, then CRM Desktop calls the function that the form handler specifies.

The form handler uses the following format:

```
objectType_form()
```

For example:

```
contact_form()
```

You can specify a form handler between the following tags in the definition of the object in the forms_xx.xml file:

```
<script>>
</script>
```

For example, the following code specifies a form handler for contacts:

```
<script><![CDATA[
  include("forms.js", "forms");
  var ctx = {"application": application,"ui": ui,"application_script":
  application_script,"form": form};
  var current_form = new forms.contact_form(forms.create_form_ctx(ctx));
  ]]>
</script>
```

Customizing Form Functions

A *form function* is a type of function that resides in the form handler function that Siebel CRM Desktop uses for the object. It is specific to the form that contains the object. You can use a form function to validate user entry or to add business logic, such as to enter information in a hidden field, create a link, and so on. For more information, see "Functions That You Must Not Modify" on page 165.

To customize form functions

1. Use a JavaScript editor to open the forms.js file.
2. Locate the following section:

   ```
   //FORM FUNCTIONS
   ```

3. Add the form functions that your customization requires to the section that you located in Step 2.

   To view example form functions that do validations, see "Example of Creating a Custom Validation" on page 229.

Using Regular Expressions in Form Functions

To use a regular expression in a form function, you use a variable that contains the filtering criteria and a string that Siebel CRM Desktop must examine. You use the following format:

```
function function_name()
{
var fields = form.item.snapshot;
var filter = regular_expression
return filter.test(fields['field_name']);
}
```

where:

- *function_name* identifies the name of your custom function.
- *regular_expression* contains a string. You use two forward slashes (//) to indicate this regular expression.
- *field_name* identifies the field that contains the string that you define in the regular expression.

For example, the following example uses a regular expression to make sure the user enters a value of manager in the Job Title field:

```
function validate_job_title()
{
var fields = form.item.snapshot;
var filter = /manager/g
return filter.test(fields['Job Title']);
}
```

Customizing Event Connectors

An *event connector* is a type of form function that can run another function when CRM Desktop triggers a form event or a control event. This topic describes how to add two different example event connectors.

To customize event connectors

1. Use a JavaScript editor to open the forms.js file.
2. Locate the following section:

   ```
   //FORM EVENTS
   ```

3. Add the event connectors that your customization requires to the section that you located in Step 2.

   To link a function to the form event, you use the following format:

   ```
   ctx.form.event_type.connect(function_to_call);
   ```

   If you link a control to the form event, you use the following format:

   ```
   ctx.events.connect(control_name, control_event, function_to_call);
   ```

   For example, the following code connects a form function to a form event:

```
ctx.form.on_saving.connect(form_saving);
```

In this example, CRM Desktop calls the form_saving function before it saves the record. It connects the on_saving event that resides in each form to a function that resides in the form handler for each object that the form.js file defines. To view another example of this usage, see "Adding Postdefault Values to Fields" on page 188.

For another example, the following code connects the event that CRM Desktop uses with the sales_method field to the on_sales_method_changed function:

```
ctx.events.connect(ctx.form["sales_method"], "changed",
on_sales_method_changed);
```

This connect uses the following parameters to make the call:

- Form control name
- Event for the control
- Function name

To view more examples of this usage, see "Updating One Field If the User Modifies Values In Another Field" on page 189, and "Linking Fields and Testing Your Hierarchical Picklist" on page 330.

Customizing Triggers

A *trigger* is a type of function that Siebel CRM Desktop calls if an event occurs. If CRM Desktop creates or removes a link, and if a trigger exists for this link, then CRM Desktop calls this trigger.

To customize triggers

1. Use a JavaScript editor to open the forms.js file.
2. Locate the form handler you must modify.

   You can add a trigger anywhere in the form handler or in the business_logic.js file in the section that starts with the following code:

   ```
   with (scheme.triggers)
   ```

3. Add the triggers that your customization requires to the form handler that you located in Step 2.

   You use the following format to add a trigger:

   ```
   ctx.triggers.add_simple_trigger(function_name, link_source, link_destination,
   link_tag, link_operation);
   ```

   where:

   - *function_name* specifies the name of the function that CRM Desktop must call.
   - *link_source* specifies the object type that CRM Desktop uses as the link source.
   - *link_destination* specifies the object type that CRM Desktop uses as the link destination.

- *link_tag* identifies the tag. The tag you specify depends on the following function that defines the relationship that exists between objects. This function resides in the business_logic.js file:
  - **add_direct_link.** You set link_tag to `direct`. For example, to support a user who uses the autocomplete list to create the link from an opportunity object to an account object.
  - **add_mvg_link.** You set link_tag to `mvg`. For example, to support a user who uses a multivalue group to create a link between object types.
- *link_operation* specifies when CRM Desktop runs the trigger. You can use one of the following values:
  - **created.** CRM Desktop runs the trigger if the user creates the link. For example, if the user adds a new account to an opportunity.
  - **removed.** CRM Desktop runs the trigger if the user removes the link. For example, if the user deletes an existing account from an opportunity.

In the following example, if the user adds an account to an opportunity, then CRM Desktop calls the on_update_account function:

```
ctx.triggers.add_simple_trigger(on_update_account, "Opportunity", "Account",
"direct", "created");
```

For an example that uses a trigger, see "Allowing Users to Open Top-Level Objects from the Control Panel" on page 148.

Customizing Defaulting

Defaulting is a type of form function that automatically enters a value in the field of a form when the user creates a new item. It gets the functions and values for these fields from the business_logic.js file that defines the object, and then uses these functions and values to enter the field values. Defaulting uses the following code:

```
ctx.form_links_manager.init_new()
```

This code exists in every form handler. It calls the init_new method that resides in the *form_links_manager*, which is an object that determines the behavior of a form and specifies the relationship that exists between controls that reside in this form and the fields that these controls reference. The init_new method gets information about an object from the business_logic.js file, including information about how the defaulting options are set for the object. You specify these options in the `//Defaulting` section of the business_logic.js file. For more information about adding default values, see "Adding Default Values to Fields" on page 185.

To customize defaulting

1. Use a JavaScript editor to open the forms.js file.
2. Locate the following section:

```
//Defaulting
```

3 Add the defaulting that your customization requires to the section that you located in Step 2.

You use the following format to define defaulting:

```
scheme.objects.get_object(object_type).get_field(object_field)["source"] =
"string_value";
```

where:

- *object_type* identifies the type of object where CRM Desktop must enter a default value. For example, to add a default value for an action.
- *object_field* identifies the name of a field where CRM Desktop must enter a default value. This field resides in the object type, such as Status.
- *source* identifies how CRM Desktop gets the default value that it enters.
- *string_value* identifies where CRM Desktop gets the string that it uses for the default value.

Table 15 describes how you set *string_value* depending on how you set *source*.

Table 15. Specifying a Custom Default

Source	Description

initial_value	CRM Desktop gets the default value from a string that you define. You use the following format:
	`scheme.objects.get_object(object_type).get_field(object_field)["source"] = "string";`
	For example:
	`scheme.objects.get_object("Contact").get_field("Status")["initial_value"] = “Active”;`

initial_value_res	CRM Desktop gets the default value from a string key that the business_logic.js file or the package_res.xml file contains. You use the following format:
	`scheme.objects.get_object(object_type).get_field(object_field)["source"] = "string_key";`
	For example, the following code enters a value in the Status field of an action. It uses the lang_action_initial_status string key from the package_res.xml file to get this value:
	`scheme.objects.get_object("Action").get_field("Status")["initial_value_res"] = "lang_action_initial_status";`
	where:
	■ `lang_action_initial_status` identifies the following tag that resides in the package_res.xml file:
	`<str key="lang_contact_initial_status">Active</str>`
	This tag sets the value of the string to Active.

Table 15. Specifying a Custom Default

Source	Description

initial_value_fn	CRM Desktop gets the default value from a function. This function resides in the business_logic.js file. You can use initial_value_fn if you use custom logic to determine the default value. You use the following format:
	`scheme.objects.get_object(object_type).get_field(object_field)["source"] = function_name;`
	The following example enters a default value in the Percent Complete field of an action. It calls the percent_f function that resides in the business_logic.js file to get this value. The percent_f function calculates and then returns the default value:
	`scheme.objects.get_object("Action").get_field("Percent Complete")["initial_value_fn"] = percent_f;`
	`function percent_f() { return null; };`

initial_links_fn	CRM Desktop gets the default value from a function. You can use initial_links_fn if the user must create a link between objects. initial_links_fn is similar to initial_value_fn except the function must return an array instead of a string. This array must include one or two objects, depending on the types of links that you require Siebel CRM Desktop to create. The following example adds a new link anytime Siebel CRM Desktop creates a new action:
	`scheme.objects.get_object("Action")["initial_links_fn"] = prefill_owner;`
	`function prefill_owner(ctx)`
	`{`
	`  var current_user_id = helpers.get_current_user_id(ctx.session);`
	`  if (current_user_id != null)`
	`  {`
	`  return ([`
	`  { "with_id": current_user_id, "tag": "mvg" },`
	`  { "with_id": current_user_id, "tag": "direct" }`
	`  ]);`
	`  }`
	`  else return [];`
	`}`

Registering Form Controls

This topic describes how to register form controls. It includes the following topics:

If you add one of these controls, then you must register it.

Registering Autocomplete Controls

This topic describes how to register an autocomplete control. Note the following:

- An *autocomplete control* is a type of control that displays a many-to-one relationship between the current object type and another object type. For example, between many opportunities and an account. It automatically displays objects in a drop-down list while the user enters characters in a field. For example, if the user enters the letters `Big` in the Account field of the Opportunity form, then Siebel CRM Desktop displays account names that start with `Big`, arranged in ascending alphabetic order. The user can then choose an account from this list.
- An *autocomplete list* is a type of drop-down list that displays a many-to-many relationship between the current object type and another object type. For example, between parent opportunities and child contacts. It automatically displays child objects in a drop-down list while the user enters characters in a field. For example, if the user enters the letters `Doe` in the Contact field of the Opportunity form, then CRM Desktop displays a drop-down list that includes contact names that start with Doe, arranged in ascending alphabetic order. The user can then choose a contact from this list.

To view an example that registers an autocomplete control, see "Modifying the Business Logic and Testing Your Work" on page 319, and "Defining the View" on page 317. For an alternative to using an autocomplete control, see "Configuring Autocomplete Lists and Primary Selectors for MVGs" on page 368.

To register autocomplete controls

1. Use a JavaScript editor to open the forms.js file.
2. Locate the form handler you must modify.

   For more information, see Customizing Form Handlers on page 166.
3. Add the following code to the form handler you located in Step 2:

   ```
   register_autocomplete_control(ctx, object_type, autocomplete_control,
   salesbook_button, options)
   ```

   where:

   - `ctx` is a default parameter. You do not modify this parameter.
   - *object_type* identifies the object type that CRM Desktop displays in the autocomplete control and in the SalesBook control.
   - *autocomplete_control* specifies the name of the autocomplete control that the form uses. The forms_xx.xml file defines this name.
   - *salesbook_button* specifies the name of the button control that the user clicks to open the SalesBook control.

- *options* allows you to specify more options. For more information, see "Including Options When Registering Autocomplete Controls" on page 174.

For example:

```
register_autocomplete_control(ctx, "Account", "account_id",
"btn_account_select", { "online_sb": new mvg_dialogs.online_sales_book() });
```

This code does the following work:

- Displays accounts in the autocomplete control
- Uses account_id as the name of the autocomplete control that the form uses
- Displays the btn_account_select button that the user clicks to open the SalesBook control
- Uses options to specify other information.

Including Options When Registering Autocomplete Controls

You can include the following options when you register an autocomplete control:

- **online_sb.** Displays the Online button in the SalesBook control.
- **sources.** Defines options for an object that resides in the business_logic.js file.

You use the following format to add these options:

```
var custom_options =
{"option":value,"option":value}
```

where:

- Curly brackets enclose the options.
- A colon separates each option.
- Double quotation marks (") enclose each key.
- Values can reside in the form that a JavaScript object uses or in a JavaScript array.

For example, assume you specified to use the accounts:salesbook view in the SalesBook for an account. You specified this configuration in the options for the selector. Assume you must now configure Siebel CRM Desktop to use a custom SalesBook control for the account object. To do this, you can use the following code:

```
var account_options =
{
  "online_sb": new mvg_dialogs.online_sales_book(),
  "sources":[
    {
      "caption": "obj_account_plural_1",
      "view_id": "accounts:salesbook",
      "search_by": ["Name"],
      "allow_new": false,
      "online": {
        "view_id": "accounts:online_salesbook",
        "like_template": "*{keyword}*",
        "filter_fn": function(ctx, keywords_filter) { return keywords_filter; }
```

```
        }
      }
    ]
  }
```

where:

- `account_options` is a custom variable you create that contains the options. You can use this variable to register the function. You can also specify these options in the function that does the registration.
- `online_sb` is the first option. The following value is predefined. You must not modify it:

  ```
  new mvg_dialogs.online_sales_book()
  ```

- `sources` is the second option. You must specify the values for this option as a JavaScript array that includes the settings for the JavaScript object, such as `caption` and `view_id`.

Registering View Controls

This topic describes how to register a view control. To view an example that modifies the registration of a view control, see "Removing Child Objects" on page 260.

To register view controls

1. Use a JavaScript editor to open the forms.js file.
2. Locate the form handler you must modify.

   For more information, see Customizing Form Handlers on page 166.

3. Add the following code to the form handler you located in Step 2.

   ```
   register_view_control_with_button(ctx, object_type, view_control, add_button,
   delete_button, unlink_button, options)
   ```

   where:

   - `ctx` is a required parameter. You do not modify this parameter.
   - *object_type* identifies the object type that CRM Desktop displays in the view.
   - *view_control* identifies the name of the view control that CRM Desktop displays in the form. The forms_xx.xml file defines this control.
   - *add_button* identifies the name of the button control that the user clicks to add a record.
   - *delete_button* identifies the name of the button control that the user clicks to delete a record.
   - *unlink_button* identifies the name of the button control that the user clicks to remove a link that links one record to another record. If the view displays child objects, then you must set unlink_button to the following value:

     ```
     null
     ```

- *options* allows you to specify more options. You can specify options to register a view control the same way you specify options to register an autocomplete control. For more information about these options and the format you must use, see "Including Options When Registering Autocomplete Controls" on page 174.

  In addition to these options, it is recommended that you include the following option for every view you register. This option configures CRM Desktop to use custom view controls on an object form:

  ```
  {"custom_view_ctrl": true }
  ```

For example:

```
register_view_control_with_button(ctx, "Action", "activities_view",
"btn_add_activity", "btn_remove_activity", null, {"custom_view_ctrl": true });
```

This code does the following work:

- Displays actions in the view
- Displays the activities_view view control in the form
- Displays the btn_add_activity button that the user clicks to add an activity
- Displays the btn_remove_activity button that the user clicks to delete an activity
- Sets unlink_button to null because the view displays child objects
- Uses custom view controls on an object form

Registering MVG Controls

This topic describes how to register an mvg_dialog control. To view examples that use this control, see "Controlling the Search in Siebel Button That Does Online Lookup" on page 198"Controlling the Pin Period for Contacts in the Activity Form" on page 202, and "Configuring Autocomplete Lists and Primary Selectors for MVGs" on page 368.

To register MVG controls

1. Use a JavaScript editor to open the forms.js file.
2. Locate the form handler you must modify.

   For more information, see Customizing Form Handlers on page 166.

3. Add the following code to the form handler you located in Step 2:

   ```
   register_mvg_dialog(ctx, object_type, primary_selector, show_dialog_button,
   options)
   ```

   where:

   - ctx is a required parameter. You do not modify this parameter.
   - *object_type* identifies the object type that CRM Desktop displays in the mvg_dialog control.

- *primary_selector* specifies the name of the primary selector control that CRM Desktop displays in the mvg_dialog control. The forms_x.xml file defines this selector.
- *show_dialog_button* identifies the name of the button control that the user clicks to open the MVG dialog box.
- *options* allows you to specify more options. For more information, see "Including Options When Registering MVG Controls" on page 177.

Including Options When Registering MVG Controls

You can specify options to register an MVG control the same way you specify options to register an autocomplete control. For more information about these options and the format you must use, see "Including Options When Registering Autocomplete Controls" on page 174.

In addition to these options, you can include the following option:

```
"use_autocomplete_list": true
```

This option configures CRM Desktop to use an autocomplete_list control instead of a primary_selector control. If you use this option, then you must also modify the primary_selector control to use the autocomplete_list control in the form_xx.xml file.

The following example includes all the options that you can use when registering an MVG control:

```
var additional_contact_options = {
    "online_sb": new mvg_dialogs.online_sales_book(),
      "use_autocomplete_list": false,
      "sources":[
        {
          "caption": "obj_account_plural_1",
          "view_id": "contacts:salesbook",
          "search_by": ["Name"],
          "allow_new": false,
          "online": {
            "view_id": "contacts:online_salesbook",
            "like_template": "*{keyword}*",
            "filter_fn": function(ctx, keywords_filter) { return keywords_filter; }
          }
        }
      ]
    }

register_mvg_dialog(ctx, "Contact", "ActionToContact", "btn_mvgContact",
additional_contact_options);
```

Customizing Field Behavior

This topic describes how to customize the user interface. It includes the following topics:

Displaying Siebel CRM Fields

The example in this topic displays the Mail Stop field on the Contact form in the client. You make this field available through the Siebel API and then customize CRM Desktop to synchronize and display the field. You modify the following files:

- siebel_meta_info.xml
- siebel_basic_mapping.xml
- forms_xx.xm
- package_res.xml

For more information, see "Files That the Customization Package Contains" on page 434.

To display a Siebel CRM field

1. Open Siebel Tools and then display the object type named Integration Object.

   For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2. Make sure the Mail Stop field exists on the Contact business component.

   If it does not, then add it now.

3. Add the Mail Stop field to the CRMDesktopContactIO integration object.

   In order for CRM Desktop to synchronize data with the Siebel database, you use CRM Desktop integration objects and integration components to make the objects and fields that you use in this example available. The Contact object is already available but the Mail Stop field is not. To make the Mail Stop field available, you add it to the Contact integration component for each of the required integration objects:

   a. In the Object Explorer, click Integration Object and then locate the CRMDesktopContactIO integration object in the Integration Objects list.

   b. In the Object Explorer, expand the Integration Object tree, click Integration Component, and then locate the Contact integration component in the Integration Components list.

   c. In the Object Explorer, expand the Integration Component tree and then click Integration Component Field.

d In the Integration Component Fields list, add a new record using values from the following table.

Property	Value
Name	Mail Stop
Data Type	DTYPE_TEXT

4 Repeat Step 3 for the CRMDesktopAccountIO integration object.

5 Repeat Step 3 for the CRMDesktopOpportunityIO integration object.

6 Compile all locked projects.

After compiling finishes, the Mail Stop field is available through the API and you can configure CRM Desktop to use the field. For more information, see *Using Siebel Tools*.

7 Specify the objects and fields to synchronize:

a Use an XML editor to open the siebel_meta_info.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

b Locate the following tag:

```
object TypeId='Contact'
```

Several child field tags reside in the `object TypeId='Contact'` tag. These children identify the fields for the Contact object.

c Add the following field tag as a child of the tag that you located in Step b:

```
<field Name='Mail Stop' Label='Mail Stop' DataType='DTYPE_TEXT'
IOElemName='MailStop' />
```

d Repeat Step b and Step c for the following tag:

```
object TypeId='Account.Contact'
```

e Repeat Step b and Step c for the following tag:

```
object TypeId='Opportunity.Contact'
```

f Save and close the siebel_meta_info.xml file.

8 Map the Mail Stop field from the Contact object in the Siebel database to a field in CRM Desktop:

a Use an XML editor to open the siebel_basic_mapping.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

b In the siebel_basic_mapping.xml file, add a new field tag to the type tag using values from the following table.

Tag	Value
type id	Contact

c Add the following code to the tag you created in Step b.

```
<field id="Mail Stop">
  <reader class="mapi_user">
  <user_field id="sbl Mail Stop" ol_field_type="1"></user_field>
    <convertor class="string"></convertor>
  </reader>
  <writer class="Microsoft Outlook_user">
    <user_field id="sbl Mail Stop" ol_field_type="1"></user_field>
    <convertor class="string"></convertor>
  </writer>
</field>
```

d Save and close the siebel_basic_mapping.xml file.

9 Insert a label and the Mail Stop field following the Job Title field on the Contact form:

a Open the forms_12.xml file and then locate the cell that contains the #lbl_job_title label control.

b Insert the following XML code immediately after the cell that contains the #lbl_job_title label control:

```
<cell size="22">
  <control id="lbl_MailStop" class="static" tab_order="6">
    <text>#lbl_mail_stop</text>
  </control>
</cell>
```

The following code specifies a key that the package_res.xml file uses to determine the localized value for the label:

```
#lbl_mail_stop
```

c Locate the section that is labeled with the following comment:

```
left side fields
```

This section resides in the tag that resides in the form that contains the SBL Contact ID.

d Add the text field control. You insert the following XML code immediately before the cell that contains the ContactToAccount MVG control, and just following the cell that contains the status_image control:

```
<cell size="22">
  <control class="edit" id="MailStop" tab_order="7">
    <field value="string">Mail Stop</field>
  </control>
</cell>
```

e Locate the cell size tag, and then change it to the following value:

```
<cell size="185">
```

To make room for the new field, you must increase the cell size that contains all of the child objects. In this example, you change the cell size from 155 to 185.

10 Add the following code to the package_res.xml file:

```
<str key="lbl_mail_stop">Mail stop:</str>
```

Add this code as a child of the res_root tag under the following comment:

```
<!-Contact Form
```

This code provides localized valuesand images to the client. It allows the Contact form to display the Mail Stop label through a key value. The package_res.xml file provides localized values and images to CRM Desktop. You added the new Mail Stop field to the Contact form, so you must provide the text for the label. When you modified the forms_12.xml file, you created a label control that contains #lbl_mail_stopfor the text value. This control identifies the key to use in the package_res.xml file.

11 Republish the updated package files.

During the next synchronization, CRM Desktop uses the updated files to apply the modifications to the Contact form. The Mail Stop field is available on the Contact form and CRM Desktop synchronizes the values in this field with the Siebel Server. For more information, see "Republishing Customization Packages" on page 80.

Hiding Siebel CRM Fields

The example in this topic describes how to hide the Opportunity field so that Siebel CRM Desktop does not display it in the client.

To hide Siebel CRM fields

1 Remove the following objects from the connector_configuration.xml file:

- Remove all child objects, such as Opportunity.Contact.Association.
- Remove all association objects, such as Opportunity.Contact.Association.
- Remove links to opportunities that exist in the links and natural_keys sections.

2 Remove the following objects from the siebel_basic_mapping.xml file:

- Remove all child objects, such as Opportunity.Contact.Association.
- Remove all association objects, such as Opportunity.Contact.Association.
- Remove links to opportunities from objects that include fields that are similar to OpportunityId.

3 Remove the following controls for this Opportunity object from the forms_xx.xml file:

- Remove view controls from forms that display opportunities and objects that are related to opportunities.
- Remove lookup controls and autocomplete controls from other forms that reference the Opportunity object.

4 Remove related toolbar buttons from the toolbars_xx.xml file.

The button Id or an action contains the name of this object. For example, new_opportunity or email_to_team_opportunity.

5 In the business_logic.js file, remove the following definitions where CRM Desktop uses this Opportunity object:

- add_mvg_link
- add_direct_link

You can remove other code that affects opportunities, but this code must not affect other CRM Desktop functionality.

Adding Custom Fields

A *custom field* is any new field that you create that is not a Microsoft Outlook default field on a Microsoft Outlook form. If Siebel CRM Desktop must synchronize the contents of a custom field with data from the Siebel Server, then you must configure the XML files so that they identify the data that CRM Desktop must read and synchronize. If you do not configure these XML files, then CRM Desktop will not synchronize the data that the new field displays with the data that resides on the Siebel Server. The example in this topic adds the EEG Trainings field.

To add custom fields

1 Create a custom field named EEG Trainings in Siebel Tools, and then add it to an integration object.

For more information, see "Displaying Siebel CRM Fields" on page 178.

2 Specify the meta information for the EEG Trainings field:

a Use an XML editor to open the siebel_meta_info.xml file.

b Locate the following tag of the object that you must modify:

```
<object>
. . .
</object>
```

c Add the following code to the tag you located in Step b:

```
<field Name='EEG Trainings' Label='#fld_contact@EEG_Trainings_'
DataType='DTYPE_TEXT' IOElemName='EEGTrainings' />
<field Name='Suppress All Emails' Label='#fld_contact@Suppress_All_Emails_'
DataType='DTYPE_BOOL' IOElemName='SuppressAllEmails' />
```

d Save, and then close the siebel_meta_info.xml file.

3 Specify the field mappings for the EEG Trainings field:

a Use an XML editor to open the siebel_basic_mapping.xml file.

b Locate the following tag of the object that you must modify:

```
<type>
. . .
</type>
```

c Add the following code to the tag you located in Step b:

```
<field id="EEG Trainings">
  <reader>
    <mapi_user>
      <user_field id = "sbl EEG Trainings" ol_field_type = "1"></user_field>
      <convertor>
      <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <outlook_user>
      <user_field id = "sbl EEG Trainings" ol_field_type = "1"></user_field>
      <convertor>
      <string/>
      </convertor>
    </outlook_user>
  </writer>
    </field>
    <field id="Suppress All Emails">
  <reader>
    <mapi_user>
      <user_field id = "sbl Suppress All Emails" ol_field_type = "6"></
  user_field>
    <convertor>
    <bool/>
    </convertor>
      </mapi_user>
    </reader>
  <writer>
    <outlook_user>
      <user_field id = "sbl Suppress All Emails" ol_field_type = "6"></
    user_field>
      <convertor>
        <bool/>
      </convertor>
    </outlook_user>
    </writer>
</field>
```

d Save, and then close the siebel_basic_mapping.xml file.

4 Specify the display format for the EEG Trainings field:

a Use an XML editor to open the forms_xx.xml file.

b Locate the following tag of the object that you must modify:

```
<form>
. . .
</form>
```

c Add the following code to the tag you located in Step b:

```
<cell size="21"> <stack layout="vert">
<cell> <static id="lbl_EEG_Trainings" tab_order="112">
  <text>#lbl_EEG_Trainings</text> </static>
```

```
</cell>
  </stack>
    </cell>
      <cell size="21"> <stack layout="vert">
    <cell>
      <checkbox id="checkbox_Suppress_All_Emails" tab_order="115">
        <field>Suppress All Emails</field>
        <text>#lbl_Suppress_All_Emails</text>
      </checkbox>
    </cell>
  </stack>
</cell>
<cell size="21">
  <edit id="EEG_Trainings" max_chars="100" tab_order="113">
    <field value="string">EEG Trainings</field>
  </edit>
</cell>
```

d Save, and then close the forms_xx.xml file.

Making Fields Read-Only

To make a field read-only, you disable the corresponding control on the form that references the field that you must make read-only. The example in this topic makes the Opportunity Name field on the opportunity form read-only.

To make a field read-only

1 Use a JavaScript editor to open the forms.js file.

For more information, see "JavaScript Files in the Customization Package" on page 437.

2 Locate the function that is associated with the form you must modify.

For example, to make an item on the Opportunity form read-only, locate the following function:

```
// OPPORTUNITY FORM SCRIPTS //
function opportunity_form(ctx)
{
}
```

For more information, see "Customizing Form Functions" on page 167.

3 In the function you located in Step 2, locate the following statement:

```
set_controls_access()
```

4 Add the following code after the statement you located in Step 3:

```
ctx.form[control_id].enabled = value;
```

where:

- *control_id* is the name of the control in the forms_xx.xml file.

- *value* determines if the field is read-only. You can use one of the following values:
  - **True.** Make the field editable.
  - **False.** Make the field read-only.

  For this example, you add the following code:

  ```
  ctx.form["opportunity"].enabled = false;
  ```

5. The ctx.form file includes form controls that Siebel CRM Desktop maps to corresponding fields. For information about the forms_xx.xml file, see "Files That the Customization Package Contains" on page 434.
6. Open the client and then navigate to the Opportunity form.
7. Open an opportunity and make sure you cannot modify the Opportunity Name.
8. Republish the customization package.

   For more information, see "Republishing Customization Packages" on page 80.

Adding Default Values to Fields

This topic describes how to configure Siebel CRM Desktop to add a default value to a field when the user creates a new record. The example in this topic configures CRM Desktop to add the following default value to the Opportunity Name field:

```
CRM Opportunity
```

For more information, see "Customizing Defaulting" on page 170.

To add a default value to a field

1. Use a JavaScript editor to open the business_logic.js file.
2. Add the following code to the end of the create_siebel_meta_scheme2 function:

   ```
   scheme.objects.get_object("object_type").get_field("field_name")["default_source"] = default_value;
   ```

   where:

   - *object_type* is the name of the object type identifier of the object type that resides in the siebel_basic_mapping.xml file. This file includes definitions for object types. Each definition includes an Id attribute. This attribute is the object type identifier that you must use for the object_type. For more information, see "Customizing Field Mapping" on page 160.
   - *field_name* is the name of a field that resides in the object type definition in the siebel_basic_mapping.xml file.
   - *default_source* identifies the source for the default value. For more information, see "Setting Default Values" on page 186.

- *default_value* defines the default value that CRM Desktop adds.

For this example, you add the following code:

```
scheme.objects.get_object("Opportunity").get_field("Name")["initial_value"] =
"CRM Opportunity";
```

3 Save and close the business_logic.js file and then test your work.

4 Republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Setting Default Values

This topic describes the values that you can use to specify the source of the default value. You specify this value in the default_source variable of the create_siebel_meta_scheme2 function.

Setting Fixed Default Values

To set a fixed value, you set the default_source variable of the create_siebel_meta_scheme2 function to the following value:

```
initial_value
```

You typically use this format with a number that does not require a translation, such as a Boolean number or an empty string where the default value is empty. You use the following format:

```
scheme.objects.get_object("object_type").get_field("field_name")["initial_value"]
= value;
```

Example 1

To set the Appt PIM Flag field that resides in the Action object, you use the following code:

```
scheme.objects.get_object("Action").get_field("Appt PIM Flag")["initial_value"] =
true;
```

This example sets the default value to true. It does not use a list of values. The Appt PIM Flag field is a Boolean field.

Example 2

To set the country code for the phone number to a default value when the user creates a new account, you can use the following code:

```
scheme.objects.get_object("Account").get_field("MainPhone
Number")["initial_value"] = "+31";
```

Setting Default Values That Are Language Dependent

To get a language dependent value from the resource file, you set the default_source variable of the create_siebel_meta_scheme2 function to the following value:

```
initial_value_res
```

This value uses the value that Siebel CRM Desktop defines in the package_res.xml file and in each language-specific package_res_XX_yy.xml file. It allows you to use a different default value for each language. If the MLOVs (multi-value lists of values) are correctly configured on the Siebel Server, then CRM Desktop stores the correct Language-Independent Code on the server. You use the following format:

```
scheme.objects.get_object("object_type").get_field("field_name")["initial_value_res"] = resource_key;
```

Example

The following example specifies to set the default value of the Display field that resides in the Action object to the value that the lang_action_display_activities_only resource key contains:

```
scheme.objects.get_object("Action").get_field("Display")["initial_value_res"] =
"lang_action_display_activities_only";
```

The following values in the resource file determine the default value:

- The English resource file includes the following code:

  ```
  <str key="lang_action_display_activities_only">Activities Only</str>
  ```

  At run time CRM Desktop sets the default value in the English client to Activities Only.

- The Dutch resource file includes the following code:

  ```
  <str key="lang_action_display_activities_only">Alleen activiteiten</str>
  ```

  At run time CRM Desktop sets the default value in the Dutch client to Alleen activiteiten.

Setting Default Values for Functions

To set the default value for a function, you set the default_source variable of thecreate_siebel_meta_scheme2function to the following value:

```
initial_value_fn
```

This value uses a function that returns a value. It allows you to do more complex lookups. You use the following format:

```
scheme.objects.get_object("object_type").get_field("field_name")["initial_value_fn"] = function_name;
```

Example

The following example comes predefined with Siebel CRM Desktop. It calls the following get_default_currency_code function to determine the currency code that CRM Desktop sets for the account:

```
scheme.objects.get_object("Account").get_field("Currency
Code")["initial_value_fn"] = get_default_currency_code;
```

The following predefined code defines the get_default_currency_code function. You can also write your own function that provides a default value:

```
function get_default_currency_code(ctx)
{
  var defaults = helpers.get_defaults(ctx.session);
  var currency_id = null;
  if ((defaults != null) && (defaults["CurrencySymbol"] != null))
{
  var currency = ctx.session.open_item(defaults["CurrencySymbol"]);
    if (currency != null)
    {
    currency_id = currency.id();
    }
  }
return currency_id;
}
```

Setting Default Values For Links

To set the default links that Siebel CRM Desktop uses for an object type, you set the InitialLinksCallback property of the descriptor that this object type uses to an instance of the class that CRM Desktop gets from the CallbackObject class. CRM Desktop uses this property to create the initial links. The following example uses a multi-value group that returns multiple records:

```
scheme.objects.get_object("Account")["initial_links_fn"] = prefill_team;
```

This example uses a multi-value group that associates multiple records. To add multiple team members instead of setting a single value, you can use a function that you specify elsewhere and use the following source:

```
initial_links_fn
```

Adding Postdefault Values to Fields

This topic describes how to add a value to a field if the user does not enter any data in this field when the user creates a new record. Adding a value in this way is known as adding a *postdefault* value. The example in this topic modifies the predefined opportunity form to make sure Siebel CRM Desktop sets the postdefault value for the Lead Quality field in the opportunity to 5-Poor if the user does not set a value for this field.

To add a postdefault value to a field

1. Use a JavaScript editor to open the form that includes the field you must modify.
2. Make sure the following function exists:

   ```
   ctx.form.on_saving.connect(function_name);
   ```

   where:

   - *function_name* is the name of the custom function that handles the event.

   This function handles the on_saving event for the form. For example, the predefined opportunity_form function includes the following code:

```
ctx.form_on_saving.connect(form_saving);
```

To set the form_saving function, the predefined code uses the following code. It makes sure the opportunity is capitalized. It occurs earlier in the script:

```
var form_saving = function()
{
  var fields = ctx.form.item.snapshot;
  ctx.form.item["Name"] = fields["Name"].replace(/\b[a-z]/g,function(w){return
w.toUpperCase()}); // Capitalization
}
```

For more information, see "Customizing Form Functions" on page 167.

3 Add the following code to the end of the form_saving function:

```
if (validator.empty_field_validator("Quality"))
  ctx.form.item["Quality"] = "5-Poor";
```

To determine if the user set the value for the Quality field, this if statement uses the empty_field_validator function in the following way:

- If the field is empty, then the empty_field_validator function returns a value of true and the ctx.form.item statement sets the value of the Quality field to 5-Poor.
- If the field is not empty, then the empty_field_validator function returns a value of false and the code exits the if statement.

4 (Optional) Support an environment that does not use English. You do the following:

a Replace the `ctx.form.item["Quality"] = "5-Poor";` code that you added in Step 3 with the following code:

b `ctx.form.item["Quality"] = ctx.session.res_string("lang_lead_quality_poor");`Add the following code to the package_res.xml file:

```
<str key="lang_lead_quality_poor">5-Poor</str>
```

The code you added in Step 3 works in an environment that uses English but fails in a multilingual environment because you hard-coded the field value. The call to ctx.session.res_string allows you to retrieve a string from the resource file and to write JavaScript code that is language independent.

5 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Updating One Field If the User Modifies Values In Another Field

You can configure Siebel CRM Desktop to update the value in one field if the user changes the value in another field. This functionality is known as *on field update*. The example in this topic configures CRM Desktop to do the following work:

- If the name of the opportunity is Test, then set the Lead Quality to 5-Poor.

- If the name of the opportunity is not Test, then set the Lead Quality to 3-High.

To update one field if the user modifies values in another field

1 Add the following function:

```
function on_name_changed()
{
  var NameValue = ctx.form.item.snapshot['Name'];
  if (NameValue == "Test") ctx.form.lead_quality.value =
      ctx.session.res_string("lang_lead_quality_poor");
  else
    ctx.form.lead_quality.value =
      ctx.session.res_string("lang_lead_quality_high");
}
```

This on_name_changed function gets the value for the Name field from the snapshot. It then uses this value to set the value in the field that resides on the form to a value that the resource file contains. A *snapshot* is a function that allows CRM Desktop to get the current value of a field. For example, the following code gets the current value of the Name field:

```
Ctx.form.item.snapshot['Name']
```

For more information, see "Customizing Form Functions" on page 167.

2 Call the function you added in Step 1. You add the following event connector:

```
ctx.events.connect(ctx.form["opportunity"],"on_focus_lost",on_name_changed);
```

It is recommended that you use the on_focus_lost event handler for an edit control. If the opportunity field is a dropdown list, then it is more appropriate to use the changed event and to use the following code:

```
ctx.events.connect(ctx.form["opportunity"],"changed",on_name_changed);
```

For more information, see "Customizing Event Connectors" on page 168.

3 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Creating Calculated Fields

You can configure Siebel CRM Desktop to display a calculated field in the client so that it behaves in a way that is similar to how a calculated field behaves in the client of a Siebel Business Application. The example in this topic configures a calculated field in the client. If the user changes the opportunity name, then the value in this field also changes. You do the following:

- Expose a calculated field to an integration object and synchronize it with CRM Desktop. This configuration allows CRM Desktop to get a correct starting value for the calculated field. In this example, you use a calculated field named JVD Calculated. It includes the following calculated value:

```
[Name] + " - Calculated"
```

- Use a CRM Desktop calculated field to make a working copy of the original calculated field. You do this because you cannot configure CRM Desktop to make changes to a calculated field from Siebel CRM while the user is using CRM Desktop. Doing so might cause a synchronization error.
- Use JavaScript in the form to make sure the copy of the calculated field changes if the user changes the opportunity name. This configuration allows you to use the same behavior that occurs in the client of a Siebel Business Application for the calculated field.

For more information about each of these items, see "Alternative Ways to Create Calculated Fields" on page 193.

To create a calculated field

1. Open Siebel Tools and then display the object type named Integration Object.

   For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2. In the Object Explorer, click Integration Object.
3. In the Integration Objects list, query the Name property for CRMDesktopOpportunityIO.
4. In the Object Explorer, expand the Integration Object tree and then click Integration Component.
5. In the Integration Components list, add a new integration component using values from the following table.

Property	Value
Name	JVD Calculated
Data Type	DTYPE_TEXT
Length	285
External Sequence	240
External Name	JVD Calculated
External Data Type	DTYPE_TEXT
XML Sequence	240
XML Tag	JVDCalculated

   This integration component exposes the Siebel CRM calculated field to the integration object for the opportunity.

6. Deploy your changes to the Siebel Runtime Repository.
7. Use an XML editor open the siebel_meta_info.xml file.
8. Locate the following object:

   ```
   TypeId="Opportunity"
   ```

9. Add the following code to the object you located in Step 8:

```
<field Name='JVD Calculated' Label='JVD Calculated' DataType='DTYPE_TEXT'
IsFilterable='no' IsHidden='no'IOElemName='JVDCalculated' />
```

This code adds the JVD Calculated field to the metadata object for Opportunity. For more information, see "JavaScript Files in the Customization Package" on page 437.

10 Set the calculated field in Siebel CRM Desktop. You add the following code to the siebel_basic_mapping.xml file:

```
<field id="JVD Calculated">
    <reader>
      <mapi_user>
        <user_field id="sbl JVD Calculated" ol_field_type="1"></
          user_field>
        <convertor>
          <string/>
        </convertor>
      </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Calculated" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

11 Use an XML editor to open the forms_xx.xml file.

For more information about the forms_xx.xml file, see "Files in the Customization Package" on page 434.

12 Locate the Opportunity form. You locate the following code:

```
<form id="SBL Opportunity">
```

13 Add the calculated field to the opportunity form. You add the following code to Opportunity form:

```
<edit id="jvd_calculated">
  <field value="string">JVD Calculated</field>
</edit>
```

14 Use a JavaScript editor to open the forms.js file.

15 Make the field read-only. You add the following code to the opportunity_form function:

```
ctx.form.jvd_calculated.enabled = false;
```

A calculated field in Siebel CRM is read-only. It is recommended that you also make the calculated field in CRM Desktop read-only. For more information, see "Making Fields Read-Only" on page 184.

16 Save your work.

17 Configure CRM Desktop to update the field value. You add the following code to the file:

```
function jvd_calculate()
{
  var NameValue = ctx.form.item.snapshot['Name'];
  ctx.form.jvd_calculated.value = NameValue + " -
Calculated";
}
```

This function gets the value of the Name field. It then writes to the jvd_calculated field the value of this Name field plus the following string concatenated to the opportunity name:

– Calculated

To calculate the value, Siebel CRM uses the following code:

```
[Name] + " – Calculated"
```

For more information, see "Customizing Form Functions" on page 167.

18 Add the following code to the on_focus_lost event:

```
ctx.events.connect(ctx.form["opportunity"],"on_focus_lost",jvd_calculate);
```

For more information, see "Customizing Event Connectors" on page 168.

This code makes sure CRM Desktop calls this function if the user changes the value in a field that the calculation uses. In this example, the calculated value depends only on the opportunity name, so this code only calls this function if the user changes the value in the Opportunity Name field.

This code creates a dependency between an event and a function. If the form field is:

- A dropdown list, then you use the changed event.
- An edit box, then you use the on_focus_lost event.

The opportunity name field is an edit box so you configure CRM Desktop to call the function on the on_focus_lost event.

19 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Alternative Ways to Create Calculated Fields

This topic describes alternative ways to create a calculated field.

Exposing Calculated Siebel CRM Fields

To create a calculated field, you can expose a calculated field that exists in Siebel CRM to an integration object and then synchronize it to Siebel CRM Desktop. However, if this field is not read-only in the client, then CRM Desktop attempts to synchronize the new value back to Siebel CRM, and this synchronization fails.

Specifying Fields in the Siebel Meta Info File as Calculated Fields

You can specify a field in the siebel_meta_info.xml file as a calculated field. To do this, you set the IsCalculated attribute to yes and then use the following code to specify a value for the Formula attribute:

```
:[ :] Block, should contain field.
:( ) Container for fieldname.
```

For example, you can use the following code to combine the First Name field and the Last Name fields:

```
:[:(First Name) :(Last Name):]
```

This primary allows you to concatenate fields with the possibility to add some static characters to the concatenation. It does not allow you to configure CRM Desktop to do a calculation. You specify this code in the siebel_meta_info.xml file. CRM Desktop only determines the calculated value during synchronization. From this point the field is read-only.

Using JavaScript to Mimic Calculated Fields

You can write JavaScript that mimics the behavior of a calculated field. You can configure CRM Desktop to run this JavaScript code only in reply to something that happens in the form, such as the user changing the value in a field. This configuration updates a field if the data changes in the form but you cannot use it to control the value of a calculated field that Siebel CRM Desktop displays when the user opens a form.

Customizing UI Behavior

This topic describes how to customize behavior in the user interface. It includes the following topics:

- Localizing Strings on page 223
- Localizing the Forms Files on page 224

Customizing the Product Name

The client displays the following text in a number of locations:

- Siebel CRM
- CRM Desktop
- Microsoft Outlook

You can change this text to a custom value.

To customize the product name

1. Use an XML editor open the package_res.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Create or modify any of the following attributes, as required:

   - `<str key="app_name">CRM Desktop</str>`
   - `<str key="pim_name">Outlook</str>`
   - `<str key="remote_app_name">Siebel</str>`

   For example, in the remote_app_name attribute, change Siebel to your company name.

3. Save and test your changes and then republish the customization package.

   For more information, see "Republishing Customization Packages" on page 80.

Customizing the Email Address of the Support Team

The defines various resources for the customization package. In this file, you can specify the email address of the support team where the user sends feedback.

To customize the email address of the support team

1. Use an XML editor open the package_res.xml file.
2. Modify the following code of:

```
<!-- Feedback page -->
  <str key="support_email">email_address</str>
```

   where:

- *email_address* is the email address where Siebel CRM Desktop sends requests for support

For example:

```
<str key="support_email">support@your_company.com</str>
```

If you specify the email address in the support_email variable, and if the user clicks Send Feedback on the Feedback tab in the Options dialog box, then CRM Desktop does the following work:

- Opens a new email message.
- Automatically enters the value that you specify in the support_email variable. It enters this information in the To line of this email message.

If the user clicks the Send Feedback button on the Feedback tab in the Options dialog box, and if you do not specify the email address, then CRM Desktop opens the email without an email address in the To line. CRM Desktop does not come predefined with a support email address. You must specify it.

Controlling Buttons That Send Email Messages and Set Up Meetings

The contact form in the client includes the following buttons:

- Email to Account Team
- Meeting with Account Team

This topic describes how to configure Siebel CRM Desktop to display these buttons in Outlook 2007. For more information, see "Scenario for Managing Account Information" on page 17.

To control buttons that send email messages and set up meetings

1. Use a JavaScript editor to open the application_script.js file.
2. Locate the following code:

```
var account_team_options = helpers.merge_contexts(team_options,
{"type_dependence": { "Account": "enabled" } });
```

3. Change the code you located in Step 2 to the following:

```
var account_team_options = helpers.merge_contexts(team_options,
{"type_dependence": { "Account": "enabled","Contact":"enabled" } });
```

   This code displays the Email to Account Team and the Meeting With Account Team buttons.

4. Use a JavaScript editor to open the package_res_xx.xml file and then add the following code:

```
<str key="btn_email_to_contact_team">Email to Contact Team</str>
<str key="btn_meeting_with_contact_team">Meeting with Contact Team</str>
```

   This code changes the button labels so that they are consistent with the contact information that CRM Desktop displays in the contact form.

5. Save and then close the application_script.js file.

6 Use an XML editor to open the toolbars.xml file.

For more information, see "Files in the Customization Package" on page 434.

7 Locate the following object:

```
'<custom_ui for="Microsoft.Outlook.Contact">'
```

8 Add the following code to the object you located in Step 7:

```
<button id="email_to_team_contact"label="#btn_email_to_contact_team"
size="normal"image="orcl_email_to_team:16" getVisible="get_visible"
getEnabled="get_enabled" onAction="button_on_action"/>

<button id="meeting_with_team_contact"label="#btn_meeting_with_contact_team"
size="normal" image="orcl_meeting_with_team:16" getVisible="get_visible"
getEnabled="get_enabled" onAction="button_on_action"/>
```

9 Save and then close the toolbars.xml file.

10 Open the application_script.js file and then locate the following code:

```
var account_team_options
```

11 Add the following code immediately before the code you located in Step 10:

```
var contact_team_options = helpers.merge_contexts(team_options,
{"type_dependence": { "Contact": "enabled" } });
action_manager.add_action("email_to_team_contact",email_to_team,contact_team_op
tions);
action_manager.add_action("meeting_with_team_contact", meeting_with_team,
contact_team_options);
```

12 Save and test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Controlling the New Button in the Sales Book

This topic describes how to configure CRM Desktop to prevent a user from creating a new object from the Sales Book, such as an account. For example, you can configure it to make the Account form read-only, but if the user opens the Sales Book to pick an Account, then this user can click New to create a new Account. The example in this topic describes how to configure CRM Desktop to make an account a read-only object but still allow the user to associate an account with a contact.

To control the new button in the Sales Book

1 Use an XML editor to open the lookup_view_def.xml file.

2 Locate the following code:

```
<lookup_view_def key="lookup:accounts">
  <display name="#obj_account_plural" />
  <filter dasl="([http://schemas.microsoft.com/mapi/proptag/0x001A001E] >=
  'IPM.Contact.SBL.Account' AND [http://schemas.microsoft.com/mapi/proptag/
```

```
0x001A001E] <= 'IPM.Contact.SBL.Account')" />
  <view id="accounts:salesbook" />
  <quick_lookup dasl_format="[http://schemas.microsoft.com/mapi/proptag/
0x3A11001F] = '%s'" />
  <type id="Account" />
</lookup_view_def>
```

3 Remove the Account Id from the type id tag, which is indicated in Step 2 in bold. For example:

```
<type id="" />
```

4 Save your changes and reload the client.

Controlling the Search in Siebel Button That Does Online Lookup

The Online Lookup feature allows the user to connect to the Siebel Server, search for data, and then link this data to an object in the client. For example, assume the user must add a contact to a calendar appointment but this contact does not reside in the set of contacts that CRM Desktop synchronized to the client. The user can use Online Lookup to search for this contact on the Siebel Server and then add it to the calendar appointment in the client. The user clicks the Search in Siebel button in the CRM Desktop - SalesBook dialog box to start Online Lookup. For information about controlling how CRM Desktop pins objects that the Online Lookup feature locates, see Controlling How Siebel CRM Desktop Pins Objects on page 201.

The example in this topic adds Online Lookup for employee records.

To control the Search in Siebel button that does Online Lookup

1 Add the data source that Online Lookup must use:

a Use an XML editor to open the data_sources.xml file.

b Locate the connector element.

c Add the following data source definition for employees to the element that you located in Step b:

```
<data_source name="Employee" scope="General">
  <type>Employee</type>
  <columns>
    <column sort_order="1" sort_mode="asc">First Name</column>
    <column sort_order="0" sort_mode="asc">Last Name</column>
    <column sort_order="2" sort_mode="asc">Position</column>
  </columns>
</data_source>
```

You can specify a set of columns in the columns section, including a sort mode of asc or desc and the sort order. For more information, see "Setting the Scope for Online Lookup" on page 200.

2 Add the view definition that Online Lookup must use:

a Use an XML editor to open the views.xml file.

b Add the following definition for the employee view. Column headings contain the names of the resources that the package_res.xml file contains:

```
<view id="employees:online_salesbook">
  <image_list>
      <res_id type="normal">type_image:User:16</res_id>
  </image_list>
  <columns>
    <column width="150" sort="asc">
      <heading type="string">head_first_name</heading>
      <field>First Name</field>
    </column>
    <column width="150">
      <heading type="string">head_last_name</heading>
      <field>Last Name</field>
    </column>
    <column width="200">
      <heading type="string">obj_position</heading>
      <field>Position</field>
    </column>
  </columns></view>
```

3 Specify parameters for the object selector:

a Use a JavaScript editor to open the business_logic.js file.

b Locate the selectors section that starts with the following comment:

```
// Selectors options
```

c Locate the following selector for the Employee object:

```
scheme.objects.get_object("Employee").selectors_options = {
  "source": {
    "caption": "obj_employee_plural",
    "view_id": "employees:salesbook",
    "search_by": ["First Name", "Last Name"],
    "display_format": ":[:(Login Name):]",
    "data_source_type": "database",
    "allow_new": false
  }
};
```

d Add the following online parameter that enables Online Lookup. The view_id parameter references the view name that you created in Step 2:

```
scheme.objects.get_object("Employee").selectors_options = {
  "source": {
  "caption": "obj_employee_plural",
  "view_id": "employees:salesbook",
  "search_by": ["First Name", "Last Name"],
  "display_format": ":[:(Login Name):]",
  "data_source_type": "database",
  "allow_new": false,
  "online": {
    "view_id": "employees:online_salesbook",
```

```
        "like_template": "*{keyword}*"
      }
    }
    };
```

4 Register the MVG control on the form that must include Online Lookup. In this example, you add Online Lookup for the Employee MVG control on the Activity form:

   a Use a JavaScript editor to open the forms.js file.

   b Locate the following function:

```
activity_form_base(ctx)
```

   c Locate the following code in the function that you located in Step b:

```
// Employee MVG
register_mvg_dialog(ctx, "Employee", "ActionToEmployee", "btn_mvgEmployee", {
"use_autocomplete_list": false });
```

   For more information, see “Registering MVG Controls” on page 176.

   d Add the following online_sb parameter to the code that you located in Step c:

```
// Employee MVG
register_mvg_dialog(ctx, "Employee", "ActionToEmployee", "btn_mvgEmployee", {
"use_autocomplete_list": false,
"online_sb": new mvg_dialogs.online_sales_book()
});
```

5 Save all the JavaScript and XML files you modified and then restart the client.

6 Test your work:

   a Open any calendar appointment in the client.

   b Click the button for the Employee field.

   c In the Activity Employees dialog box, click the SalesBook control.

   d In the CRM Desktop - SalesBook dialog box, click Search in Siebel.

   e In the CRM Desktop - online SalesBook dialog box, enter a wildcard search in the text window.

   For example, enter the following wildcard search:

```
J*
```

   f Make sure CRM Desktop populates the Siebel CRM - - online SalesBook dialog box with all employees that reside on the Siebel Server that match your wildcard search.

Setting the Scope for Online Lookup

The scope attribute specifies the view mode that CRM Desktop uses to search records on the Siebel Server. It can include one of the following values:

- General.
- Deduplication.

- Not specified. CRM Desktop sets the scope to General.

CRM Desktop uses a different view mode to search for records in the employee example according to the following scope:

- **General.** CRM Desktop uses the Organization view mode.
- **Deduplication.** CRM Desktop uses the All view mode.

These values correspond to the following view mode that the siebel_meta_info.xml file specifies for the Employee object type:

```
<object TypeId='Employee' Label='Employee' LabelPlural='Employees'
EnableGetIDsBatching='true' IntObjName='CRMDesktopEmployeeIO'
SiebMsgXmlElemName='Employee'
SiebMsgXmlCollectionElemName='ListOfCRMDesktopEmployeeIO'
>
<viewmodes General="Organization" Dedup="All" />
```

Controlling How Siebel CRM Desktop Pins Objects

CRM Desktop uses a push pin icon to indicate the records that the Online Lookup feature brings into the client. CRM Desktop pins these items for one week, by default. It removes these pinned items from the client after one week, including the record and any links to the record. You can change this default value.For more information "Controlling the Search in Siebel Button That Does Online Lookup" on page 198.

Controlling How Long Siebel CRM Desktop Pins Objects

This topic describes how to change the default value that determines how long CRM Desktop pins an object.

To control how long Siebel CRM Desktop pins an object

1. Use a JavaScript editor to open the mvg_dialogs.js file.
2. Locate the following code:

```
this.default_options = {
  "online_sb_xml": online_sb_xml,
  "online_sb_template_params": {
    "caption": "#ol_lookup-online_caption"
  },
  "remote_connector_timeout": 0,
  "remote_connector_max_rows_number": 100
};
```

3. Modify the code you located in Step 2 to the following code. Bold font indicates the modification:

```
this.default_options = {
  "online_sb_xml": online_sb_xml,
  "online_sb_template_params": {
    "caption": "#ol_lookup-online_caption"
```

```
  },
  "remote_connector_timeout": 0,
  "remote_connector_max_rows_number": 100
  "pinned_object_lifetime": pin_time
};
```

where:

- *pin_time* specifies the number of seconds to pin an item. Note the following:
  - The default value is 604800, which is seven days.
  - One day is 86400.
  - You can specify -1 (negative one) to pin objects forever.

4 Locate the following code in the select_items function:

```
synchronizer.pin_object(view_dsc.link_to, remote_id);
```

5 Replace the code you located in Step 4 with the following code:

```
synchronizer.pin_remote(view_dsc.link_to, remote_id,
options.pinned_object_lifetime);
```

Controlling the Pin Period for Contacts in the Activity Form

The example in this topic modifies code so that CRM Desktop permanently pins contacts in the Activity form.

To control the pin period for contacts in the Activity form

1 Use a JavaScript editor to open the forms.js file.

2 Locate the following code:

```
register_mvg_dialog(ctx, "Contact", "ActionToContact","btn_mvgContact", {
"use_on_afrer_saved": ctx.parent_ctx != null &&
ctx.parent_ctx.form.ol_2003_virtual_form, "online_sb": new
mvg_dialogs.online_sales_book(), "use_autocomplete_list": false });
```

To change the pin period for a specific object, you must locate the code for the form that you must modify. It is recommended that you locate the mvg_dialogs.online_sales_book call. For example, the code in this step resides on the Activity form and it sets up the contact association. For an example of how locating this code might vary, see "Controlling the Pin Period for Contacts in the Opportunity Form" on page 203. For more information about register_mvg_dialog, see "Registering MVG Controls" on page 176.

3 Modify the code you located in Step 2 to the following code. Bold font indicates the modification:

```
register_mvg_dialog(ctx, "Contact", "ActionToContact", "btn_mvgContact", {
"use_on_afrer_saved": ctx.parent_ctx != null &&
ctx.parent_ctx.form.ol_2003_virtual_form, "online_sb": new
mvg_dialogs.online_sales_book(), "use_autocomplete_list": false,
"pinned_object_lifetime": -1 });
```

Controlling the Pin Period for Contacts in the Opportunity Form

The example in this topic modifies code so that CRM Desktop permanently pins contacts in the Opportunity form.

To control the pin period for contacts in the Opportunity form

1 Use a JavaScript editor to open the forms.js file.

2 Locate the following code:

```
var contact_options = { "link_to": "Contact","tag": "mvg","autocomplete":
ctx.form.add_contact_subform.contact_id,"btn_show":
ctx.form.add_contact_subform.btn_contact_select,"btn_add":
ctx.form.btn_contact_add,"related_selector": new
mvg_dialogs.custom_sales_book(),"sb_custom_view": true,"online_sb": new
mvg_dialogs.online_sales_book(),"remote_connector_timeout": 15000}
```

In this example, CRM Desktop sets up the options in the Opportunity form as a separate object that it sends to the register function. This configuration is different from the configuration described in "Controlling the Pin Period for Contacts in the Activity Form" on page 202.

3 Modify the code you located in Step 2 to the following code. Bold font indicates the modification:

```
var contact_options = { "link_to": "Contact","tag": "mvg","autocomplete":
ctx.form.add_contact_subform.contact_id,"btn_show":
ctx.form.add_contact_subform.btn_contact_select,"btn_add":
ctx.form.btn_contact_add,"related_selector": new
mvg_dialogs.custom_sales_book(),"sb_custom_view": true,"online_sb": new
mvg_dialogs.online_sales_book(),"remote_connector_timeout": 15000,
"pinned_object_lifetime": -1}
```

Controlling How Siebel CRM Desktop Sorts Records in Comboboxes

This topic describes how to control how Siebel CRM Desktop sorts records in a combobox. For more information, see "Combobox Control of the Forms File" on page 477.

To control how Siebel CRM Desktop sorts records in comboboxes

1 Use an XML editor to open the forms_xx.xml file.

2 Use the following code to add the order_by clause to a combobox control:

```
<combobox id="combobox_name" tab_order="tab_order">
  <items format=":[:(field_name_to_display):]" value_column="value_column"
has_null_item="null_item_value">
    <source type="auto" name="type_name">
    </source>
    <order_by>
      <order ascend="true">sort_field</order>
    </order_by>
```

```
    </items>
    <field>display_field</field>
  </combobox>
```

where:

- *combobox_name* identifies the name of combobox control.
- *value_column* identifies the field name from which to save the value.
- *null_item_value* is set to one of the following values:
  - **true.** Display null items.
  - **false.** Do not display null items.
- *type_name* identifies the object type name from the siebel_basic_mapping.xml file.
- *sort_field* identifies the field name on which CRM Desktop does the sort.
- *display_field* identifies the name of the field that CRM Desktop displays in the client that displays the sort results.

For example:

```
<combobox id="Type" tab_order="9">
  <items format=":[:(Label):]" value_column="Value" has_null_item="true">
    <source type="auto" name="AccountTypePicklist"></source>
    <order_by>
      <order ascend="true">SortOrder</order>
    </order_by>
  </items>
  <field>Type</field>
</combobox>
```

Controlling How Siebel CRM Desktop Handles Data That Is Not Directly Visible

You can use the viewmodes element to configure Siebel CRM Desktop to display or hide data that is not directly visible. *Data that is not directly visible* is a type of data that the client does not receive during synchronization, but instead gets through an association with another Siebel CRM object type. For example, filter settings might prevent CRM Desktop from synchronizing some accounts to the client, but the synchronized contacts that reference these account might contain this account data. In this situation, CRM Desktop displays this account information in the Contact form, but not in an Account view.

Beginning with Siebel CRM Desktop version 3.7, you can configure a view mode according to the type of operation that CRM Desktop performs. You can use one of the following values for the viewmodes element:

- **General.** Specifies a value for the General viewmode. If this value exists, then it overrides a value that the ViewMode attribute specifies.
- **Dedup.** Specifies a value for the Deduplication view mode. For more information, see Resolving Synchronization Conflicts on page 154.

- **QBID.** Specifies a value for the Query-By-Id view mode.
- **Not specified.** If you do not specify the viewmodes element, then CRM Desktop does the following:
  - Uses the default value for the deduplication viewmode that it gets from the value that you set for the General view mode
  - Uses All as the default value for the Query-By-Id view mode.

CRM Desktop uses a master filter only on query change requests when it uses the General view mode.

Using Query By Id to Hide Data That is Not Directly Visible

The example in this topic configures CRM Desktop to hide all data that is not directly visible for contacts. It prevents CRM Desktop from getting data from the Siebel Server and storing it in the client.

To hide data that is not directly visible for contacts

1. Use an XML editor open the siebel_meta_info.xml file.
2. Define the following view modes for the contact object type:

   ```
   General="Sales Rep" Dedup="All" QBID="Sales Rep"/>
   ```

3. Define the following view modes for the account object type:

   ```
   General="Sales Rep" Dedup="All" QBID="Sales Rep"/>
   ```

4. Define the following view modes for the opportunity object type:

   ```
   General="Sales Rep" Dedup="All" QBID="Sales Rep"/>
   ```

5. Define the following view modes for Contact.Account and Account.Contact:

   ```
   General="Sales Rep" Dedup="All" QBID="Sales Rep"/>
   ```

6. Define the following view modes for Contact.Opportunity and Opportunity.Contact:

   ```
   General="Sales Rep" Dedup="All" QBID="Sales Rep"/>
   ```

7. Make sure the user does not modify the default filter preset that restricts the number of objects that CRM Desktop synchronizes from the Siebel Server.

Using Query By Id to Synchronize Only My Accounts and Activities

1. Use an XML editor open the siebel_meta_info.xml file.
2. Locate the following code:

   ```
   Account
   ```

```
<object TypeId='Account' Label='#obj_account' LabelPlural='#obj_account_plural'
UpsertBusObjCacheSize='0' EnableGetIDsBatching='true'
IntObjName='CRMDesktopAccountIO' SiebMsgXmlElemName='Account'
SiebMsgXmlCollectionElemName='ListOfCRMDesktopAccountIO' >

<viewmodes General="Sales Rep" Dedup="All"/>

Account.Action

<object TypeId='Account.Action' Label='Activity' LabelPlural='Activities'
EnableGetIDsBatching='true' IntObjName='CRMDesktopAccountIO'
SiebMsgXmlElemName='Action' SiebMsgXmlCollectionElemName='ListOfAction' >

<viewmodes General="All" Dedup="All" />
```

3 Modify the code you located in Step 2 to the following code. Bolded font indicates the modifications you must make:

```
Account

<object TypeId='Account' Label='#obj_account' LabelPlural='#obj_account_plural'
UpsertBusObjCacheSize='0' EnableGetIDsBatching='true'
IntObjName='CRMDesktopAccountIO' SiebMsgXmlElemName='Account'
SiebMsgXmlCollectionElemName='ListOfCRMDesktopAccountIO' >

<viewmodes General="Sales Rep" Dedup="All" QBID="Sales Rep"/>

Account.Action

<object TypeId='Account.Action' Label='Activity' LabelPlural='Activities'
EnableGetIDsBatching='true' IntObjName='CRMDesktopAccountIO'
SiebMsgXmlElemName='Action' SiebMsgXmlCollectionElemName='ListOfAction' >

<viewmodes General="Personal" Dedup="All" QBID="Personal"/>
```

Using Filters to Hide Data That Is Not Directly Visible

This topic describes how to hide data that is not directly visible for accounts from a custom Microsoft Outlook view that Siebel CRM Desktop uses. It describes how to prevent CRM Desktop from storing data in the client. For information about how to prevent CRM Desktop from getting data from the Siebel Server and storing it in the client, see "Using Query By Id to Hide Data That is Not Directly Visible" on page 205.

To use filters to hide data that is not directly visible

1 Use an XML editor to open the siebel_basic_mapping.xml file.

2 Locate the code described in "Account Code" on page 207.

3 Examine the following items in the code you located in Step 2:

- The view id is `all_accounts`.
- The following attribute in the definition for the User1 field instructs CRM Desktop to remove data that is not directly visible for an account from the view:

```
true
```

4 Use an XML editor to open the views.xml file and then locate the following view:

```
<str key="all_accounts">
```

5 Verify that the following code in the CDATA section is set to `true`:

```
<filter>&amp;quot;http://schemas.microsoft.com/exchange/extensionattribute1&amp;quot;
&amp;lt;&amp;gt; 'true'</filter>
```

6 Use an XML editor open the connector_configuration.xml file, and then make sure the state_field for the Account type uses the following value:

```
<type id="Account" state_field="ObjectState">
```

7 Verify that CRM Desktop applies the DASL filter:

a Open the CRM Desktop client.

b Navigate to the Accounts list.

c Right-click in the Accounts list and then choose Advanced View Settings.

d In the Advanced View Settings: Siebel Accounts dialog box, click Filter.

e In the Filter dialog box, click the SQL tab.

f Verify that the Find Items That Match These Criteria window includes the value that you set in Step 2. CRM Desktop displays this value in the following format:

```
http://schemas.microsoft.com/exchange/extensionattribute1 <> 'true'
```

If you copy this view, then CRM Desktop also copies this filter. This configuration allows you to create other views that you can use with Siebel CRM data.

Account Code

The following code specifies account information in the siebel_basic_mapping.xml file:

```
<type id="Account" display_name="#obj_account_plural" folder_type="10">
  <form message_class="IPM.Contact.SBL.Account" icon="type_image:Account:16"
large_icon="type_image:Account:32" display_name="#obj_account">SBL Account</form>
  <alt_messageclasses>
  </alt_messageclasses>
  <custom_views default_name="#view_siebel_accounts">
    <view id="all_accounts" name="#view_siebel_accounts"></view>
  </custom_views>
  <field id="ObjectState">
    <reader>
      <mapi_user>
        <user_field id="sbl ObjectState" ol_field_type="3"/>
        <convertor>
          <integer/>
        </convertor>
      </mapi_user>
    </reader>
    <writer>
```

```
      <multiwriter>
        <outlook_user>
          <user_field id="sbl ObjectState" ol_field_type="3"/>
          <convertor>
            <integer/>
          </convertor>
        </outlook_user>
        <outlook_std>
          <outlook_field id="User1"/>
          <convertor>
            <bitmask2string>
              <rule mask="134217728" result="134217728" value=""/>
              <rule mask="1" result="1" value="true"/>
              <rule mask="1073741824" result="1073741824" value="true"/>
            </bitmask2string>
          </convertor>
        </outlook_std>
        <outlook_user>
          <user_field id="sbl IndirectlyVisible" ol_field_type="6"/>
          <convertor>
            <bitmask2bool>
              <condition mask="1" result="1" eq="true"/>
            </bitmask2bool>
          </convertor>
        </outlook_user>
      </multiwriter>
    </writer>
  </field>
```

Controlling How Siebel CRM Desktop Adds Deleted Items to the Exclusion List

Starting with Siebel CRM Desktop version 3.7, if the user does not possess permission to delete a record, and if the user attempts to delete a record, then Siebel CRM Desktop automatically adds this record to the Exclusion List. This topic describes how to modify this behavior. You can configure Siebel CRM Desktop so that it does not automatically add this record to the Exclusion list. For more information about the Exclusion list, see "Controlling the Synchronization Exceptions Button In the Filter Records Tab" on page 132.

To control how Siebel CRM Desktop adds deleted items to the Exclusion list

1. Use a JavaScript editor to open the actions.js file.
2. Locate the following function:

   ```
   siebel_item_delete(ctx)
   ```

3. Locate the following code that resides in the function that you located in Step 2:

```
var default_options = {
"confirmation": true,
"exclude_allowed": false,
"silent_exclude": true,
"exclude_supported": function() { return true; }
};
```

This code determines Exclusion list functionality.

4 (Optional) Configure CRM Desktop to prompt the user to add a record to the Exclusions List. You use the following code:

```
"silent_exclude": false
```

If you do this configuration, and if the user attempts to delete a record, then CRM Desktop displays a Delete Confirmation dialog box that includes one of the following messages according to the permissions that the user possesses:

User Does Not Possess Permission to Delete Siebel Records in Outlook	User Does Possess Permission to Delete Siebel Records in Outlook
CRM Desktop displays a message that is similar to the following: You do not have permission to delete Siebel records in Outlook. However, you can add this record to the Exclusions List. If you do this, then Siebel CRM does not download the record to Outlook, but it does keep a copy of it on the Siebel Server. Do you want to add this record to the Exclusions List?	CRM Desktop displays a message that is similar to the following: Are you sure you want to delete records from Siebel and Outlook? Click Yes to delete records from Siebel and Outlook. Click No to delete records only from Outlook, and to not synchronize future updates for these records from Siebel to Outlook.

5 (Optional) Configure CRM Desktop to prevent the user from adding the record to the Exclusions list. You use the following code:

```
"exclude_supported": function() { return false; }
```

If you do this configuration, and if the user attempts to delete a record, and if the user does not possess permissions to delete Siebel records in Outlook, then CRM Desktop displays a Delete Confirmation dialog box that includes a message that is similar to the following. The user can only click OK to close the dialog box. CRM Desktop does not delete the record or add it to the Exclusions list:

```
You do not have permissions to delete Siebel records in Outlook.
```

Preventing Users from Deleting Records

This topic describes how to configure Siebel CRM Desktop to prevent the user from deleting records in the client. You can also configure it to allow the user to delete a record in the client and then confirm this deletion during synchronization. For more information, see "Configuring Siebel CRM Desktop to Disregard Erroneous Data That Users Modify" on page 142 and "Controlling How Siebel CRM Desktop Deletes Records During Synchronization" on page 151.

To prevent users from deleting records

1. In Siebel Tools, make sure the integration component object type is displayed.

   For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2. In the Object Explorer, click Integration Object.
3. In the Integration Objects list, query the Name property for CRMDesktopContactIO, and then make sure the Object Locked property contains a check mark.
4. In the Object Explorer, expand the Integration Object tree and then click Integration Component.
5. In the Object Explorer, expand the Integration Component tree and then click Integration Component User Prop.
6. In the Integration Component User Prop list, add a new record with the following values.

Property	Value
NoDelete	Y

7. Repeat Step 2 through Step 6 for every CRMDesktop integration object.
8. Compile your changes.
9. Log in to the client.
10. Delete a contact.
11. Perform a synchronization.
12. Make sure CRM Desktop displays a message that is similar to the following:

```
EAI Adapter call failed with error: No deletes are allowed in Integration
Component Action_Contact (SBhL-EAI-04183)
```

13. Republish the customization package.

    For more information, see "Republishing Customization Packages" on page 80.

Preventing Users from Deleting Records According to Conditions

This topic describes how to configure Siebel CRM Desktop to prevent the user from deleting records in the client according to a condition. For example, not allowing the user to delete an opportunity if the Status is Pending. The delete button in the client is a native Outlook button that you cannot disable. Instead, you can modify the code that runs if the user clicks Delete.

Preventing Users from Deleting Contacts According to Conditions

The example in this topic prevents the user from deleting any opportunity that includes a Status of Pending. You can configure Siebel CRM Desktop to handle all Siebel CRM objects that it stores in the Outlook Contact objectin the same way. This configuration applies to contacts, accounts, opportunities, and any other custom Siebel CRM object, such as service requests or call reports.The following variable provides access to the record data for these objects:

item

This example configures CRM Desktop to determine if an item of the correct type exists, such as a contact, account, or opportunity. It then determines if this object meets a condition. If the condition exists, then it cancels the delete.

To prevent users from deleting contacts according to conditions

1 Locate the following code:

```
var action = "cancel";
if (can_delete_all)
  {
    if (exclude_allowed)
    {
      switch

       (ctx.ui.message_box(0,ctx.session.res_string("msg_delete_exclude_confirmatio
       n_message"),ctx.session.res_string("msg_delete_exclude_confirmation_caption"
       ), 0x123))

     {
```

CRM Desktop routes every delete to the siebel_item_delete function in the actions.js file. This function sets up the default delete behavior. You can modify the delete behavior in the following function:

```
this.execute
```

This function includes a section that starts with the following code:

```
var action = "cancel";
if (can_delete_all)
{
  if (exclude_allowed)
  {
```

```
switch
(ctx.ui.message_box(0,ctx.session.res_string("msg_delete_exclude_confirmatio
n_message"),ctx.session.res_string("msg_delete_exclude_confirmation_caption"
), 0x123))
```

This code displays delete confirmation to the user. You place the code that you modify before the siebel_item_delete function to prevent CRM Desktop from displaying multiple dialog boxes.

For more information, see "Customizing Form Functions" on page 167.

2 Add the following code before the code that you located in Step 1:

```
if (item[“type_id”] == “Opportunity” && item[“Status”] ==“Pending”)

{
// show a message to the users
ctx.ui.message_box(0,ctx.session.res_string("msg_cant_delete_item"),ctx.session
.res_string("msg_cant_delete_item_caption"), 0x40);// cancel the delete event
action_ctx.cancel_action = true;// stop processing this function (otherwise the
item might still get deleted)
return;
}

End If
```

Preventing Users from Deleting Calendar Items and Activities According to Conditions

The configuration that CRM Desktop uses for activities and calendar items is different than the configuration it uses for contacts. Instead, the activity or calendar item uses the Event object in Outlook. This configuration allows the user to access an activity or calendar item in different ways.

For example, assume the user accesses the contact form. This form includes one activity, named Test Delete. Assume the user opens the activity form for Test Delete from the contact form and finds that Test Delete includes the following values:

- Birthday Call in the Type field.
- 6/30/2012 in the Start field.

At this point, CRM Desktop behavior is similar to other Siebel objects. To prevent the user from deleting an activity with a type of Birthday Call, you can add the following code, which is similar to how you handle delete prevention for most objects:

```
if (item["type_id"] == "Action" && item["Type"] == "Birthday Call")
{
ctx.ui.message_box(0,ctx.session.res_string("msg_cant_delete_item"),ctx.session.re
s_string("msg_cant_delete_item_caption"), 0x40);
action_ctx.cancel_action = true;
return;
}
```

However, the user can access this same activity from the Calendar. The user can drill down on the entry for this activity that CRM Desktop displays in the day for 6/30/2012 in the calendar. CRM Desktop then displays the activity form for this activity. This activity form is similar to a typical Outlook calendar appointment. The delete handler represents the activity with the typical following Outlook type, so the item variable now includes only the standard Outlook field:

```
type_id='Event'
```

You cannot use only the Siebel type field. Instead, if CRM Desktop receives an event through the event handler, then it gets the same object but as type_id='Action'. You then use the same code that you use in "Preventing Users from Deleting Contacts According to Conditions" on page 211 to do the remaining conditional examination. You use the following code.

To prevent users from deleting calendar items and activities according to conditions

- Use the following code:

```
if (item["type_id"] == "Event")
{
// This is a standard calendar appointment, so we need to retrieve the Activity
// Use the Id to retrieve the right object
// Get id from the active object
var intId = item["id"];
if (intId != null)
{
// The id fields is an object and need to be converted
intId = ctx.session.hexstring_to_id(ctx.session.id_to_hexstring(intId).substr(0,
200));
// Setup a search spec to query for the Activity
var filter = ctx.session.create_expression("PIMObjectId", "eq", intId);
// Run the query
var oAction = ctx.session.find_item("Action",filter);
// If we have found the Activity then we can do the similar thing again.
if (oAction && oAction["Type"] == "Birthday Call")
{
ctx.ui.message_box(0,ctx.session.res_string("msg_cant_delete_item"),ctx.session.r
s_string("msg_cant_delete_item_caption"), 0x40);action_ctx.cancel_action = true;
return;
}
}
}
```

Preventing Users from Creating New Objects

Sometimes, business requirements dictate that a user should not be able to create a particular top-level object, such as an Opportunity. CRM Desktop cannot prevent users from attempting to create a top-level object using Microsoft Outlook's native features, such as the Ctrl-N shortcut or selecting File then New. However, CRM Desktop can ensure that the new form is read-only and prevent saving the form, thus preventing the creation of the new object.

In the example in this topic, the Opportunity form is configured to be read-only, preventing a user from creating a new object in the client. If you use this technique, then the user can still modify existing Opportunity.

To prevent a user from creating an object

1 Use a JavaScript editor to open the forms.js file.

For more information, see "JavaScript Files in the Customization Package" on page 437.

2 Locate thespecific type declaration functionthat is associated with the form you must modify.

For example, to make the Opportunity form read-only, locate the following function:

```
// OPPORTUNITY FORM SCRIPTS //
function opportunity_form(ctx)
{
}
```

For more information, see "Customizing Form Functions" on page 167.

3 Add the following code:

```
{…
    if (!sb_helpers.check_first_sync(ctx)){…}:
    if (helpers.is_new_item(ctx.form.item)) {
    form_helpers.enable_form(ctx.form, false);
…
}
```

4 Save and then close the forms.js file

5 Republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Making Forms Read-Only

You can configure Siebel CRM Desktop to make a form in the client read-only. The user can view values in a read-only form but cannot modify these values.

To make forms read-only

1 Open the form handler of the object that you must make read-only.

For more information, see "Customizing Form Handlers" on page 166.

2 Add the following code to the form handler that you located in Step 1:

```
form_helpers.enable_form(ctx.form, editable);
```

where:

- ctx.form identifies the form that CRM Desktop must make read-only.

- *editable* is one of the following values:
  - **true.** Makes the form readable and writable.
  - **false.** Makes the form read-only.

For example:

```
form_helpers.enable_form(ctx.form, true);
```

3 Add the following code to make the view readable:

```
ctx.form.view_name.enabled = true;
```

For example:

```
ctx.form.contacts_view.enabled = true;
```

If you make a form read-only in Step 2, then it is recommended that you enable the view that the form uses so that the user can scroll down through the form, can scroll to the side of the form, and can drill down on a record.

Making Top-Level Objects in Forms Read-Only

This topic describes how to make the top-level objects that reside in a form read-only. For example, you can use security logic to modify access in a script without displaying the form where this object resides, or you can disable this form even if the user possesses the permissions to modify an object where a script does the modifications.

To make top-level objects in forms read-only

- Locate, and then modify the following code that resides in the form handler for the top-level object:

```
form_helpers.enable_form(ctx.form, modify_access);
```

where:

- *modify_access* contains one of the following values:
  - **true.** The user can modify values in the form.
  - **false.** The user cannot modify values in the form.

The following code sets the value for the modify_access variable:

```
var modify_access = ctx.security_descriptor.modify_access();
```

Allowing Users to Add New Records in Read-Only Forms

This topic describes how to allow the user to add a new record in a read-only form. The example in this topic configures a form that allows the user to create new accounts but not to change existing accounts. If you use this configuration, then the user cannot use the scrollbar and cannot drill down on a record.

To allow users to add new records in read-only forms

1 Use a JavaScript editor to open the forms.js file.

2 Add the following code to the account_form function:

```
ctx.form.enabled = (ctx.item_ex.get_id() == null);
```

This code allows the user to create a new account and to edit account information until the user saves the account. If the user saves the account, then the user cannot edit the account.

When CRM Desktop saves an item it sets an Id for that item. This code determines if CRM Desktop has set an Id for the current item. It returns one of the following values:

- **True.** An Id is not set for the item. This value indicates that the user can edit the form.
- **False**. An Id is set for the item. This value indicates that the user cannot edit the form.

3 Add the following code to make the view readable:

```
ctx.form.view_name.enabled = true;
```

For example:

```
ctx.form.accounts_view.enabled = true;
```

accounts_view is a view control that displays accounts from the forms_xx.xml file. If you make a form read-only in Step 2, then it is recommended that you make the view that the form uses readable so that the user can scroll down through the form, can scroll to the side of the form, and can drill down on a record.

4 (Optional) To allow the user to edit information for a new account that the user has saved but not synchronized, you can change the code that you added in Step 2 to the following:

```
form_helpers.enable_form(ctx.form, !ctx.security_descriptor.is_synced());
```

This code determines if CRM Desktop has synchronized the current object and then does the following:

- If the current object is not synchronized, then it allows the user to edit the object.
- If the current object is synchronized, then it does not allow the user to edit the object.

5 Test your modifications, and then republish the customization package.

For more information, see “Republishing Customization Packages” on page 80.

Controlling Access to Object Types

This topic describes how to control access to object types. It includes the following topics:

Making Object Types Read-Only

This topic describes how to make an object type read-only so that the user can view it but not modify it. The example in this topic makes the account object type read-only.

To make object types read-only

1 Use a JavaScript editor to open the security_utils.js file.

2 Add the object type that you must make read-only to the array of security descriptors that CRM Desktop uses:

a Locate the following function:

```
siebel_security_manager()
```

b Locate the following code that resides in the function that you located in Step a:

```
factories = {
  "Action": security_manager.factory_from_constructor(activity_security),
  ...
  "Account": security_manager.factory_from_constructor(account_security),
  "Opportunity":
  security_manager.factory_from_constructor(opportunity_security),
};
```

This code defines the array of security descriptors. A *security descriptor* is a type of object that includes a set of methods, including methods that CRM Desktop uses to determine access. Each object in CRM Desktop includes a security descriptor.

c Add the following code to the factories object that you located in Step b:

```
"custom_object":security_manager.factory_from_constructor(function_name)
```

where:

- *custom_object* identifies the object type of the custom object.
- *function_name* identifies the name of the function that CRM Desktop uses to construct the security descriptor. For more information, see "Function That CRM Desktop Uses to Construct Security Descriptors" on page 219.

For example, add the following code:

```
"Account":security_manager.factory_from_constructor(account_security)
```

3 Specify the function that determines if the user possesses the permission to modify the form:

a Locate the function that you specified in Step c on page 217.

b Add the following code to the function that you located in Step a.

```
base_security.call(this, ctx, item_ex, {
"modify_filter": modify_check_function
});
function custom_object_security(ctx, item_ex)
{
```

```
base_security.call(this, ctx, item_ex, {
"modify_filter": function
});
```

where:

- *custom_object_security* specifies the name of the function that CRM Desktop calls to determine if the user possesses the permission to modify the form. For more information about custom_object_security, see "Function That CRM Desktop Uses to Construct Security Descriptors" on page 219.

For example, you can use the following code for the account object:

```
base_security.call(this, ctx, item_ex, {
"modify_filter": modify_check_function
});
function account_security(ctx, item_ex)
{
base_security.call(this, ctx, item_ex, {
"modify_filter": function
});
```

4 Define the function that you specify in Step 3. Use the following code:

```
function modify_check_function(ctx, sd, item_ex, value)
{
  value.and_value = false;
  value.or_value = false;
}
```

where:

- modify_check_function must be the function that you specify in Step 3.
- sd provides access to the access method that the security descriptor uses, such as modify_access, delete_access, or link_access. For more information, see "Setting the Security Descriptor" on page 220.
- and_value and or_value must be set to false to make an object type read-only. These parameters determine the result of the following calculation:

  ```
  resulting_value = (value || or_value) && and_value;
  ```

  Note the following:

  - If CRM Desktop calls an access checking method, such as modify_access or delete_access, then the access checking method returns a value in resulting_value.
  - If or_value and and_value are true, then resulting_value is true.
  - If or_value or and_value is false, then resulting_value is false.
  - The value parameter is predefined. You cannot modify it. You can modify or_value and and_value.

You can define this function anywhere in the security_utils.js file. It recommended that you define it near the custom_object_security function.

5 (Optional) Make the object type read-only depending on a condition.

You can configure CRM Desktop to examine an object, and then set and_value and or_value to false or true, depending on the results of this examination. For example, the following code determines if a custom field of an object includes a value, where the value is the check mark in a checkbox. If it does, then it sets `Can modify` to false, and then sets the values for and_value and or_value depending on the value of `Can modify`:

```
function modify_check_function(ctx, sd, item_ex, value)
{
  var modification_allowed = item_ex.get_property("Can modify").checked;
  value.and_value = modification_allowed;
  value.or_value = modification_allowed;
}
```

Function That CRM Desktop Uses to Construct Security Descriptors

The function that Siebel CRM Desktop uses to construct security descriptors resides in the security_utils.js file. For example, CRM Desktop uses the following function to construct the security descriptor for the account object type:

```
function account_security(ctx, item_ex)
{
base_security.call(this, ctx, item_ex, {
"modify_filter": account_read_only,
"link_filter": account_link_filter,
"delete_filter": function(ctx, sd, item_ex, value) {value.or_value = true;
value.and_value = true}
});
}
```

The base_security function constructs the default security descriptor. This descriptor includes the predefined methods that set access. CRM Desktop uses the base_security function during a call from the security descriptor function of any object, and it uses the values that this function contains. The security descriptor uses the following constructor function, and CRM Desktop uses the default values from this function for all access methods. So, it is not necessary for you to write code in every function that CRM Desktop uses to construct a descriptor:

```
function custom_object_security(ctx, item_ex)
{
base_security.call(this, ctx, item_ex);
}
```

where:

- *custom_object_security* specifies the name of the function that CRM Desktop uses to construct the object descriptor. For example, CRM Desktop uses the account_security function to construct the object descriptor for an account.
- `this, ctx, item_ex` are the default set of parameters that CRM Desktop uses. You can add more options to define the default access methods and the values that they return.

Setting the Security Descriptor

sd (security descriptor) is the security_descriptor object that CRM Desktop uses for the object where you are modifying access. You can use it to examine other types of permissions. For example, the following code prevents the user from deleting or modifying an object. If the delete_access method that the security descriptor references returns a value of false, then CRM Desktop sets and_value and or_value to false:

```
function modify_check_function(ctx, sd, item_ex, value)
{
if (sd.delete_access())
  {
  value.and_value = false;
  value.or_value = false;
  }
}
```

Preventing Users From Deleting Object Types

This topic describes how to configure Siebel CRM Desktop so that users cannot delete object types.

To prevent users from deleting object types

- Do all the steps described in "Controlling Access to Object Types" on page 216, with the following differences:

  a. Replace the code you use in Step 3 on page 217 with the following code:

     ```
     base_security.call(this, ctx, item_ex, {"delete_filter":
     delete_check_function});
     ```

     This code uses the delete_filter parameter when Siebel CRM Desktop calls the base_security function.

  b. In Step 4 on page 218, replace modify_check_function with delete_check_function.

  Note that you can configure CRM Desktop to control access to object types and prevent users from deleting object types. To do this, do the work described in "Controlling Access to Object Types" on page 216, create a copy of this work, and then modify this copy to prevent users from deleting object types.

Preventing Users From Removing Links Between Object Types

This topic describes how to configure Siebel CRM Desktop so that the user cannot remove a link that occurs between object types, such as removing an industry link from an account.

To prevent users from removing links between object types

1. Do all the steps described in "Controlling Access to Object Types" on page 216, with the following differences:

   a. Replace the code you use in Step 3 on page 217 with the following code:

```
base_security.call(this, ctx, item_ex, {"delete_filter":
link_check_function});
```

This code uses the delete_filter parameter when Siebel CRM Desktop calls the base_security function.

b In Step 4 on page 218, use the following code:

```
if (link.link_to == "object_type")
  {
  access.create = true;
  access.remove = false;
}
```

where:

- `link` is a required parameter that contains information about the link that CRM Desktop examines. This information identifies the object that provides the link source, the object that provides the link destination, the tag that CRM Desktop uses for the link, and other information.
- *object_type* identifies the type of object where CRM Desktop applies the logic that you specify in this `if` statement.
- `access.create` is an optional parameter that you can set to one of the following values:
  - **true.** Allows the user to create a link between two object types.
  - **false.** Prohibits the user from creating a link between two object types.
- `access.remove` is an optional parameter that you can set to one of the following values:
  - **true.** Allows the user to remove a link between two object types.
  - **false.** Prohibits the user from removing a link between two object types.

For example, the following code allows the user to create a link between an industry and the object where you are defining link access, but not to remove this link:

```
if (link.link_to == "Industry")
{
access.create = true;
access.remove = false;
}
```

CRM Desktop uses access.create and access.remove to create or remove a link instead of the and_value and or_value described in "Controlling Access to Object Types" on page 216. It does this because and_value and or_value only allow you to specify a single true or false value that determines if the user can modify or delete an object. A link allows you to set a true or false value that determines if the user can create a link, and a separate true or false value that determines if the user can remove a link. For example, CRM Desktop creates a link if the user uses the autocomplete control to pick an account in an opportunity form, and it deletes this link if the user subsequently deletes this account.

CRM Desktop enables or disables buttons in the MVG dialog box depending on the access that you specify. For example, if you disallow the user from adding new associations, then CRM Desktop disables the Add button in the MVG dialog box.

Note that you can configure CRM Desktop to control access to object types and prevent users from removing links between object types. To do this, do the work described in "Controlling Access to Object Types" on page 216, create a copy of this work, and then modify this copy to prevent users from removing links between object types.

2 (Optional) Use the following code to specify the type of link:

```
if (link.link_to == "Industry" && link.tag == "link_type")
  {
  access.create = true;
  access.remove = false;
}
```

where:

- *link_type* specifies the type of link. For more information, see "Types of Links You Can Specify" on page 222.

Types of Links You Can Specify

You can use one of the following values when you specify the type of link in the `if (link.link_to` statement:

- **direct.** CRM Desktop applies the logic that you specify in this `if` statement only on direct links. A *direct link* is a type of link that possesses a one-to-one relationship between one object type and another object type. A link between one account and one opportunity is an example of a direct link. The function that resides in the business_logic.js file determines the type of link. The add_direct_link function specifies a direct link.
- **mvg.** CRM Desktop applies the logic that you specify in this `if` statement only on MVG links. An *MVG link* is a type of link that possesses a one-to-many relationship between one object type and another object type. A link between one opportunity and many contacts is an example of an MVG link. The function that resides in the business_logic.js file determines the type of link. The add_mvg_link function specifies a direct link.

For example, the following code allows the user to create a direct link between an industry and the object where you are customizing link access, but not to remove this link:

```
if (link.link_to == "Industry" && link.tag == "direct")
  {
  access.create = true;
  access.remove = false;
}
```

Localizing Strings

To localize strings, you add a resource string to a resource file and then reference that string from other XML files.

To localize strings

1 Add a new resource string for the custom label and attribute name or warning message that you must localize. Add this resource string in the following files:

- Use the package_res.xml file for a default resource.
- Use the package_res.*xx_YY*.xml file for a specific locale.

where:

- *xx_YY* is the language you use in your implementation.

For example, for Portuguese Brazilian you use package_res.pt_BR.xml.

The following standards determine the locale naming convention:

- **xx.** The ISO 639-1 standard for the language.
- **YY.** The ISO 3166-1 standard for the country. This standard supports dialects and language adoptions for specific countries.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Specify a localizable string. You add the following code:

```
<str key="string_id">localizable_string</str>
```

where:

- *string_id* is the Id of the localizable string. The double quotes are required.
- *localizable_string* is the localizable string.

3 Use the localizable string Id in every location where CRM Desktop must display the string.

You must use different formats to specify the string in different types of files. Use values from the following table.

File Type	Description

Any XML file except for the views.xml file.	Use the following format:
	#string_id
	For example:
	`<cell size="22">`
	`<static id="account_label" tab_order="6">`
	`<text>#lbl_account</text>`
	`</static>`
	`</cell>`

The views.xmlfile.	Use the following format:
	$string_id$
	For example:
	`<str key="all_accounts">`
	`<![CDATA[`
	`<?xml version="1.0"?> <view type="table">`
	`<viewname>$view_siebel_accounts$</`
	`viewname> ..... </view>`
	`]]>`
	`</str`

Any JavaScript file.	Use the following format:
	session.res_string("string_id")
	For example:
	`ui.message_box(0, session.res_string("msg_general_error"),`
	`session.res_string("msg_general_error_caption"), 0x40);`

4 Add the XML files to the customization package.

5 Republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Localizing the Forms Files

To customize the behavior of the forms_xx.xml file that comes predefined with Siebel CRM Desktop, you can use a forms file that is specific to a language. For example, for JPN (Japanese), you use forms_12.ja_JP.xml.

To localize the forms files

1. In Windows Explorer, navigate to the directory that contains the forms_xx.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Right-click the forms_xx.xml file and then click Copy.

3. Rename this copy to indicate that it is specific to a language.

   For example, you can rename the file to forms_12.ja_JP.xml.

4. Use an XML editor to open the file you renamed in Step 3, and then make any changes that are required to support the language.

   You can use this file to change the layout of the form, such as adding new fields. For example, in Japanese, you might use three different fields for the Account Name:

   - One field for kana that contains the pronunciation of the Account Name
   - One field for kanji that contains the native spelling of the Account Name
   - One field for the English representation of the Account Name

   If you make these changes, then make sure you also do the other configuration that the changes require, such as defining field mappings and modifying the form layout.

5. (Optional) Change text strings. Repeat Step 2 and Step 3 but create a copy of the package_res.xml file, rename it, and then edit the text strings.

   For example, you can rename the file to package_res.ja_JP.xml.

Validating the Data That Users Enter

This topic describes how to validate the data that the user enters in Siebel CRM Desktop. It includes the following topics:

A *validator* is a type of form handler that you can specify to validate the information that a user enters. You typically specify a validator in the following section of the forms.js file:

```
//FORM VALIDATION
```

For more information, see "Validation Rules You Can Configure for Custom Forms" on page 162.

Preparing to Use Validation

To use validation, you must make sure the form_validator object is defined.

To prepare to use validation

1 Use a JavaScript editor to open the forms.js file.

2 Make sure the form_validator object is defined.

To use validation, the forms.js file must include the following code near the start of the JavaScript function for that form:

```
var validator = new
form_helpers.form_validator(ctx.session, ctx.form);
```

This code creates a form_validator object that you can use through the validator variable.

Making Sure Users Enter Information in a Field

You can use validation to make sure the user enters information in a field.

To make sure the user enters information in a field

1 Make sure the form_validator object is defined.

For more information, see "Preparing to Use Validation" on page 226.

2 Add the following code to the forms.js file:

```
validator.validate_empty_field("field_name","controlId","string_key",boolean_highlight);
```

where:

- *field_name* is the name of the field you must examine.
- *controlId* is the name of the control in the form that Siebel CRM Desktop uses to display the field.
- *string_key* is the string key in the resource file that contains the text for the message that CRM Desktop displays in the client if the validation fails.
- *boolean_highlight* is an optional parameter that determines if CRM Desktop displays a highlighting box around the control in the client to indicate that the field is required. The default value is true.

For example, the following code makes sure the user enters information in the Opportunity Name field:

```
validator.validate_empty_field("Name","opportunity","msg_opportunity_name_validation");
```

3 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Making Sure Users Enter Unique Values

You can use validation to make sure the value that the user enters in a field is unique.

To make sure the user enters unique values

1 Make sure the form_validator object is defined.

For more information, see "Preparing to Use Validation" on page 226.

2 Add the following code to the forms.js file:

```
validator.validate_unique_fields("fields_to_check","fields_to_highlight","string_key",boolean_skip_empty_fields);
```

where:

- *fields_to_check* is an array of fields. This array must be unique.
- *fields_to_highlight* is an array of control identifiers that Siebel CRM Desktop highlights in the client if the validation fails.
- *string_key* is the string key in the resource file that contains the text for the message that CRM Desktop displays in the client if the validation fails.
- *skip_empty_fields* is an optional parameter that determines if CRM Desktop ignores empty fields when it compares records. You can set it to one of the following values:
  - **true.** Ignore empty fields.
- **false.** do not ignore empty fields.

For example, the following code makes sure the user enters an opportunity name that is unique for a given account:

```
validator.validate_unique_fields(["Name", "AccountId"],
["opportunity","account_id"],"msg_opportunity_unique");
```

where:

- ["Name", "Account Id"] identifies the fields that, when combined, must be unique. In this example, the Name field plus the Account Id field constitutes this unique combination.
- ["opportunity","account_id"] identifies the form controls that CRM Desktop highlights in the client if the validation fails. In this example, it highlights the control for the opportunity name and the control for the account name.
- msg_opportunity_unique identifies the string key in the resource file that contains the text for the message that CRM Desktop displays in the client if the validation fails.

3 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Making Sure Users Do Not Exceed the Maximum Number of Characters

You can use validation to make sure the user does not enter more than a maximum number of characters in a field.

To make sure users do not exceed the maximum number of characters

1 Make sure the form_validator object is defined.

For more information, see "Preparing to Use Validation" on page 226.

2 Add the following code to the forms.js file:

```
validator.validate_field_length("field_name","control_Id","maximum_length","string_key");
```

where:

- *field_name* identifies the name of the field that Siebel CRM Desktop must validate.
- *control_Id* identifies the control that CRM Desktop uses for the field that it must validate.
- *maximum_length* specifies the maximum length for the field in characters.
- *string_key* is the string key in the resource file that contains the text for the message that CRM Desktop displays in the client if the validation fails.

For example, the following code makes sure the user does not enter more than 1500 characters in the Comment field of an Activity:

```
validator.validate_field_length("Comment","description",1500,"msg_activity_comments_validation");
```

3 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Creating Custom Validations

You can create a custom validation.

To create custom validations

1 Make sure the form_validator object is defined.

For more information, see "Preparing to Use Validation" on page 226.

2 Add the following code to the forms.js file:

```
validator.add_custom("function_name","fields_to_highlight","string_key");
```

where:

- *function_name* is the name of a function that does the validation. CRM Desktop calls this function before it saves the record. This function gets the ctx object as input and allows it to access data and form items. This function must return one of the following values:
  - **true.** The validation is successful.
  - **false.** The validation is not successful.
- *fields_to highlight* is an array of control identifiers that CRM Desktop highlights in the client if the validation fails.
- *string_key* is the string key in the resource file that contains the text for the message that CRM Desktop displays in the client if the validation fails.

3 Test your changes and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80.

Example of Creating a Custom Validation

If the user enters a new opportunity, then the following code makes sure the close date that the user enters occurs later than the current date:

```
validator.add_custom(validate_close_date,["close_date"],"msg_opportunity_close_dat
e_validation");
```

To do the validation, this code calls the following validate_close_date function:

```
function validate_close_date()
{
  var close_date = new Date(ctx.form.item.snapshot['Primary Revenue Close Date']);
  var original_close_date = new Date(ctx.form.item['Primary Revenue Close Date']);
  var today = new Date();
  var utc_today = new
Date(today.getUTCFullYear(),today.getUTCMonth(),today.getUTCDate());
  if (close_date != null)
  {
  return close_date < utc_today ? ((original_close_date + 0) == (close_date + 0) ?
true :
false ) : true;
  }
  else true;
}
```

For more information, see "Customizing Form Functions" on page 167.Table 16 describes the parts of the validate_close_date function.

Table 16. Parts of the Validate Close Date Function

Code	Description
`var close_date = new Date(ctx.form.item.snapshot['Primary Revenue Close Date']);`	This code creates the close_date variable and sets the value for this variable to the following value: new This code does the following: 1 References the ctx object. 2 References the form. 3 Accesses the item. In this example, this item is the current record. 4 Examines the current state of the record. 5 Retrieves the Primary Revenue Close Date field.
`var original_close_date = new Date(ctx.form.item['Primary Revenue Close Date']);`	This code defines the original_close_date variable. Siebel CRM Desktop enters into this variable the value that the field contains when the user opens the form. To do this, it references the following items: ■ Ctx object ■ Form ■ Item object

Table 16. Parts of the Validate Close Date Function

Code	Description

`var today = new Date();`
 	
`var utc_today = new Date(today.getUTCFullYear(),`
 	
`today.getUTCMonth(), today.getUTCDate());`
	This code creates the following variable and enters the current date into this variable:
	
	today
	
	To support the UTC (Coordinated Universal Time) format that Siebel CRM uses, this code uses the following variable to create a UTC date:
	
	today

`if (close_date != null)`
`{`
`  return close_date < utc_today ? (`
`(original_close_date + 0) == (close_date + 0) ? true : false ) : true;`
`}`
`else true;`
	This code does the following validation:
	
	■ Determines if the user completed the field in the client. If the user did not complete the field, then this code returns the following value:
	
	true
	
	■ If the close_date occurs before the current date, then this code determines if the close date in the snapshot is different from the close date that the user set when the user opened the record:
	
	■ If these close dates are different, then this situation indicates that the user updated the close date. In this situation the close date must occur after today and the code returns the following value:
	
	false
	
	■ If these close dates are not different, then this situation indicates that the user did not change the date and this code returns the following value:
	
	true

Process of Adding Custom Objects

To add a custom object in Siebel CRM Desktop, you do the following:

1 Creating the Custom Object on page 232

2 Defining Synchronization for Custom Objects on page 238

3 Adding Custom Views in Microsoft Outlook on page 239

4 Defining the User Interface on page 239

5 Defining Validation Rules on page 246

6 Defining Validation Rules for a Phone Number on page 247

7 Adding Custom Logic on page 249

8 Defining the Toolbar on page 250

9 Defining the Logic for Custom Forms on page 251

The example in this topic adds an Activity object to Microsoft Outlook. In Siebel CRM, this object is named Action. To add this object to Outlook, you modify the siebel_basic_mapping.xml file. For more information about:

- Overview of XML files that you modify in this example, see "Overview of Customizing Siebel CRM Desktop" on page 159.
- Details about tags in XML files that you modify in this example, see Appendix C, "XML Files Reference"

Creating the Custom Object

This task is a step in "Process of Adding Custom Objects" on page 231.

In this topic, to add a new object to Microsoft Outlook, you describe the structure of the object and then create mappings between fields, lists, and so on. You make these customizations in the siebel_basic_mapping.xml file.

To create the custom object

1 Use an XML editor to open the siebel_basic_mapping.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Specify the name of the custom object. You add the following code to the siebel_basic_mapping.xml file:

```
<type id="Action" display_name="#obj_activity_plural" folder_type="10">
  <form message_class="IPM.Contact.SBL.Activity" icon="type_image:Event:16"
    large_icon="type_image:Event:32" display_name="Activity">SBL Activity</form>
</type>
```

This example code defines SBL Activity as the form to display for this object. You specify the form layout later.Note the following:

- You add the type tag to describe the new object.
- To specify the folder name for the object in Outlook, you can use the display_name attribute of the type tag.

To specify the native Outlook object that is the base for the custom object folder type, you can use the folder_type attribute.

3 Create a set of fields for the custom object.

For more information, see "Creating a Set of Fields for the Custom Object" on page 233.

4 Specify intersection objects for many-to-many relationships.

For more information, see "Specifying the Many-To-Many Relationships" on page 235.

5 Specify the lists.

For more information, see "Specifying the List" on page 236.

Creating a Set of Fields for the Custom Object

This topic describes how to create a set of fields for the custom object.

The siebel_basic_mapping.xml file describes each of the fields for the custom object. Siebel CRM Desktop maps each field to a custom field, except for the following fields:

- CRM Desktop maps the Description field to the Last Name field. This field is a native field in Outlook.
- CRM Desktop maps the Comment field to the Body field because it does not support the textarea field. Therefore, it uses the native Outlook control that displays the value for the Body field.

For more information, see ""Adding Custom Fields" on page 182.

To create a set of fields for the custom object

1 Use an XML editor to open the siebel_basic_mapping.xml file.

2 Add the following code to the type tag:

```
<type id="ChannelPartner" hidden_folder="true" folder_type="10"
display_name="CHPT">
  <form message_class="IPM.Contact.SBL.Channel_Partner" display_name="Channel
    Partner" icon="type_image:User:16"></form>
    <field id="Name">
     <reader>
       <mapi_std>
        <mapi_tag id="0x3A110000"></mapi_tag>
          <convertor><string/></convertor>
        </mapi_std>
      </reader>
      <writer>
        <Outlook_std>
          <Outlook_field id="LastName"></Outlook_field>
          <convertor><string/></convertor>
        </Outlook_std>
      </writer>
      <reader>
        <mapi_std>
          <mapi_tag id="0x3A060000"></mapi_tag>
          <convertor><string/></convertor>
        </mapi_std>
      </reader>
      <writer>
        <Outlook_std>
          <Outlook_field id="FirstName"></Outlook_field>
          <convertor><string/></convertor>
        </Outlook_std>
      </writer>
```

```
        </field>
          <field id="Location">
        <reader>
          <mapi_user>
            <user_field id="sbl Location" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </mapi_user>
        </reader>
        <writer>
          <Outlook_user>
            <user_field id="sbl Location" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </Outlook_user>
        </writer>
      </field>
</type>
```

Fields That Siebel CRM Desktop Uses for the Custom Object

Table 17 describes the fields you create for this example.

Table 17. Fields That Siebel CRM Desktop Uses for the Custom Object

Field Label	Field Name	Field Type
Description	Description	Text
Type	Type	Picklist
Priority	Priority	Picklist
Owner	Primary Owner Id	Lookup
Account	Account Id	Lookup
Opportunity	Opportunity Id	Lookup
Contacts	No field on this object	MVG
Employee	No field on this object	MVG
Planned Start	Planned	datetime
Planned Completion	Planned Completion	datetime
Due	Due	datetime
Status	Status	Picklist
Comments	Comment	Textarea

Specifying the Many-To-Many Relationships

You do not specify many-to-many relationships in Step 3 on page 232. A many-to-many relationship exists between contacts and activities, and between employees and activities. You must specify more objects that contain links to activity and contact, or activity and employee. The remaining description for a many-to-many relationship is the same as for other objects where you specify the object name and object fields. This object can also include a field that indicates if this intersection record is a primary record or not a primary record.

You must specify the Microsoft Outlook folder that stores these objects. For this example, it is not desirable to display Opportunity and Channel Partner objects to the user. You use a hidden folder in Outlook that you specify in the code as the following:

```
predefined_folder="1"
```

To specify the many-to-many relationships

1. Use an XML editor to open the siebel_basic_mapping.xml.
2. CRM Desktop does not require a form to display these objects, so you do not define a form. Instead, you add the following code:

```
<type id="Opportunity.Channel_Partner.Association" hidden_folder="true"
folder_type="10" display_name="OCHP">
<form message_class="IPM.Contact.SBL.OpportunityChannel_PartnerAssociation"
display_name="OpportunityChannel_PartnerAssociation"
icon="type_image:Generic:16"></form>
  <field id="OpportunityId">
    <reader>
      <mapi_user>
        <user_field id="sbl OpportunityId" ol_field_type="1"></user_field>
        <convertor><binary_hexstring/></convertor>
        </mapi_user>
    </reader>
    <writer>
      <Outlook_user>
        <user_field id="sbl OpportunityId" ol_field_type="1"></user_field>
        <convertor><binary_hexstring/></convertor>
      </Outlook_user>
    </writer>
    </field>
      <field id="ChannelPartnerId" ver="2">
    <reader>
      <mapi_user>
        <user_field id="sbl ChannelPartnerId" ol_field_type="1"></user_field>
      <convertor><binary_hexstring/></convertor>
      </mapi_user>
    </reader>
    <writer>
      <multiwriter>
        <Outlook_user>
          <user_field id="sbl ChannelPartnerId" ol_field_type="1"></user_field>
      <convertor><binary_hexstring/></convertor>
        </Outlook_user>
```

```
            <link_fields>
              <field from="Name" to="PartnerName"></field>
              <field from="Location" to="PartnerLocation"></field>
            </link_fields>
          </multiwriter>
        </writer>
        </field>
          <field id="PartnerName">
        <reader>
          <mapi_user>
            <user_field id="sbl PartnerName" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </mapi_user>
        </reader>
        <writer>
          <Outlook_user>
            <user_field id="sbl PartnerName" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </Outlook_user>
        </writer>
        </field>
          <field id="PartnerLocation">
        <reader>
          <mapi_user>
            <user_field id="sbl PartnerLocation" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </mapi_user>
        </reader>
        <writer>
          <Outlook_user>
            <user_field id="sbl PartnerLocation" ol_field_type="1"></user_field>
            <convertor><string/></convertor>
          </Outlook_user>
        </writer>
      </field>
    </type>
```

Specifying the List

The custom object stores items that the user chooses in a list. You describe the list field in the same way as you describe a string field. You must describe the object that stores all list values. Each list uses a separate object to store the values for the list. You must build the IDs for these objects according to the following rules:

- Object name and field
- Name and list

You must make sure the Type list on the Activity object includes the ID of the ActionTypePicklist object. To specify a list object, you must specify the following set of standard fields:

- Label
- Value (string)

- SortOrder (integer)
- IsDefault (Boolean)

To specify the list

1. Use an XML editor to open the siebel_basic_mapping.xml file.
2. To create a drop-down list, you add the following code to the activity object:

```
<type id="ActionTypePicklist" predefined_folder="1">
  <form message_class="IPM.Contact.SBL.ActionTypePicklist"></form>
  <field id="Label">
    <reader class="mapi_user">
      <user_field id="sbl picklistLabel" ol_field_type="1"></user_field>
      <convertor class="string"></convertor>
    </reader>
    <writer class="Microsoft Outlook_user">
      <user_field id="sbl picklistLabel" ol_field_type="1"></user_field>
      <convertor class="string"></convertor>
    </writer>
  </field>
  <field id="Value">
    <reader class="mapi_user">
      <user_field id="sbl picklistValue" ol_field_type="1"></user_field>
      <convertor class="string"></convertor>
    </reader>
    <writer class="Microsoft Outlook_user">
      <user_field id="sbl picklistValue" ol_field_type="1"></user_field>
      <convertor class="string"></convertor>
    </writer>
  </field>
  <field id="SortOrder">
    <reader class="mapi_user">
      <user_field id="sbl SortOrder" ol_field_type="3"></user_field>
      <convertor class="integer"></convertor>
    </reader>
    <writer class="Microsoft Outlook_user">
      <user_field id="sbl SortOrder" ol_field_type="3"></user_field>
      <convertor class="integer"></convertor>
    </writer>
  </field>
  <field id="IsDefault">
    <reader class="mapi_user">
      <user_field id="sbl IsDefault" ol_field_type="6"></user_field>
      <convertor class="bool"></convertor>
    </reader>
    <writer class="Microsoft Outlook_user">
      <user_field id="sbl IsDefault" ol_field_type="6"></user_field>
      <convertor class="bool"></convertor>
    </writer>
  </field>
</type>
```

Defining Synchronization for Custom Objects

This task is a step in "Process of Adding Custom Objects" on page 231.

In this topic, you create a custom object so that Siebel CRM Desktop can synchronize it with a Siebel CRM object.

To define synchronization for a custom object

1 Use an XML editor open the connector_configuration.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Create the Links section.

The Links section includes a set of fields that reference other objects. You must create these references to allow the Synchronization Engine to synchronize objects in the correct order and to download the related items. You add the following XML code to create the links:

```
<type id="Action">
  <view label="Activity" label_plural="Activities"
small_icon="type_image:Event:16" normal_icon="type_image:Event:24"
large_icon="type_image:Event:48"></view>
  <synchronizer name_format=":[:(Description):]">
    <links>
      <link>Account Id</link>
      <link>Opportunity Id</link>
      <link>Primary Owner Id</link>
    </links>
  </synchronizer>
</type>
```

3 Create the deduplication keys.

The business environment for this example requires that activities are the same if their descriptions are the same, so you create the Description natural key. You add the following XML code to the synchronizer tag:

```
<natural_keys>
  <natural_key>
    <field>Description</field>
  </natural_key>
</natural_keys>
```

For more information, see Resolving Synchronization Conflicts on page 154.

4 Add the following descriptions to the connector_configuration.xml file for the intersection records that you defined for contacts and employee in Step 4 on page 232:

```
<type id="Action.Employee.Association">
  <view label="Activity Employee" label_plural="Activity Employees"
small_icon="type_image:Generic:16" normal_icon="type_image:Generic:24"
large_icon="type_image:Generic:48" suppress_sync_ui="true"></view>
  <synchronizer name_format=":[:(UserName) :]">
    <links>
      <link>EmployeeId</link>
```

```
      <link>ActionId</link>
    </links>
  </synchronizer>
</type>
<type id="Action.Contact.Association">
  <view label="Activity Contact" label_plural="Activity Contacts "
small_icon="type_image:Generic:16" normal_icon="type_image:Generic:24"
large_icon="type_image:Generic:48" suppress_sync_ui="true"></view>
  <synchronizer name_format=":[:(ContactName) :]">
    <links>
      <link>ActionId</link>
      <link>ContactId</link>
    </links>
  </synchronizer>
</type>
```

Adding Custom Views in Microsoft Outlook

This task is a step in "Process of Adding Custom Objects" on page 231.

This topic describes how to add a custom view in Microsoft Outlook.

To add custom views in Microsoft Outlook

1 Use an XML editor to open the siebel_basic_mapping.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2 Add the following custom_views section to the Activity type tag:

```
<type id="Action" display_name="#obj_activity_plural" folder_type="10">
  <form (…) >SBL Activity</form>
  <custom_views default_name="Siebel Activities">
    <view id="all_activities" name="Siebel Activities"></view>
    <view id="all_activities_by_owner" name="Siebel Activities by Primary
Owner"></view>
    <view id="all_activities_by_priority" name="Siebel Activities by
Priority"></view>
  </custom_views>
```

3 Make sure the views.xml file describes the views that you specify in Step 2.

Defining the User Interface

This task is a step in "Process of Adding Custom Objects" on page 231.

In this topic, you define the form that Siebel CRM Desktop uses to display the custom object. You configure the custom object to use the custom SBL Activity form that resides in the siebel_basic_mapping.xml file that you modified in "Creating the Custom Object" on page 232.

Figure 10 illustrates the layout of the Activity Form that you create.

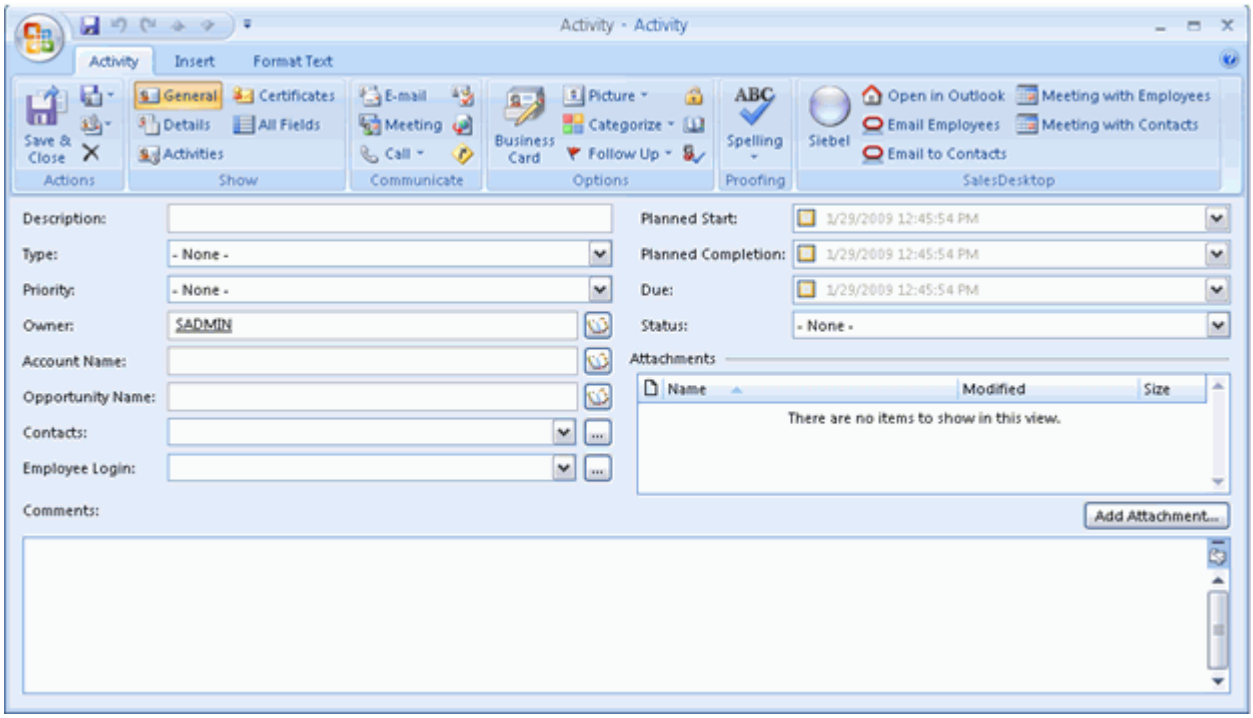


Figure 10. Layout of the Activity Form

A set of *cells* describes the form layout. A cell can be empty or it can include a control or a *stack*. A single stack can include numerous cells. The form is divided into the following parts:

- One part includes visible controls.
- One part includes hidden controls.

This example uses this configuration because the example uses native Outlook forms as a base for custom forms. Although you can modify the position of a native Outlook control, you cannot remove it entirely from the form. You move the unused controls that are native to Outlook to a location where the user cannot view them. This location is typically a very small cell.

This form is a prototype that helps you visualize the cells, stacks of cells, and the order that you use with the cells and stacks. This visualization helps to reduce the wide range of combinations of cells and stacks that you can describe down to only the cells and stacks that you can support. You can add new fields, remove fields, reorder fields, and apply any other changes during development and testing.

To define the user interface

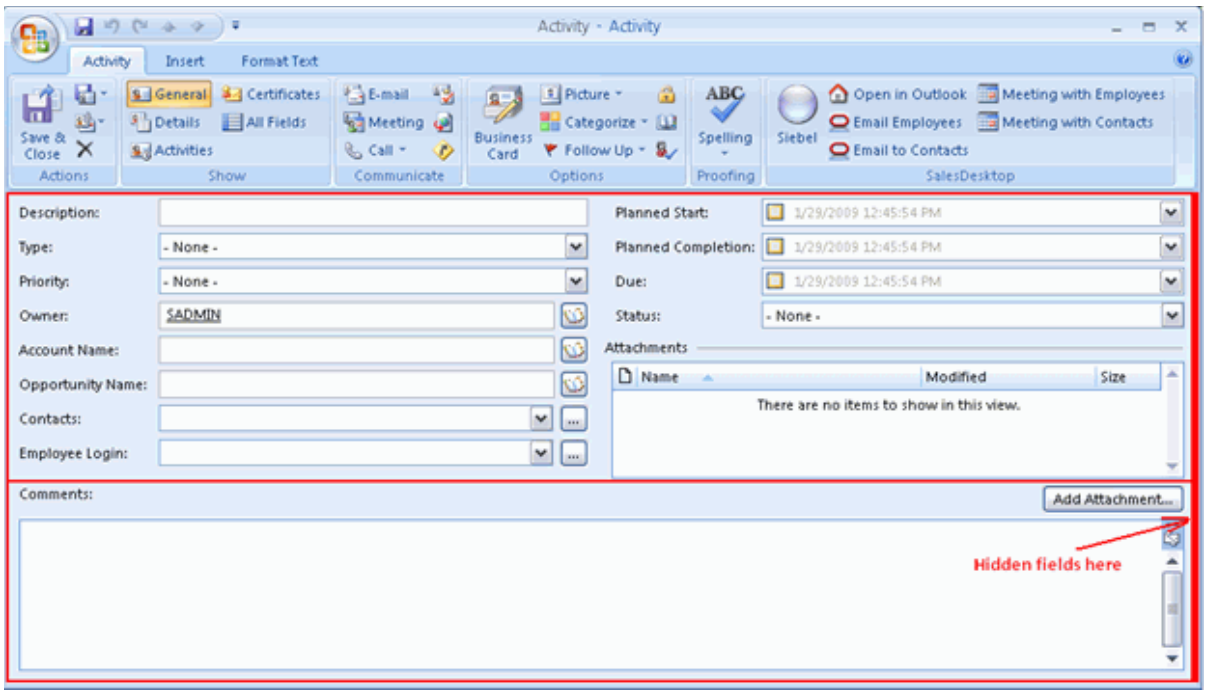
1. Use an XML editor to open the forms_12.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2 Divide the form into a visible section and a hidden section. You add the following code to the forms_12.xml file:

```
<form id="SBL Activity">
  <page id="General" tag="0x10A6" min_height="335" min_width="520">
    <cell>
      <stack layout="horz" padding="10">
      <!-- visible section -->
        <cell>
        <!-visible fields here -->
        </cell>
      <!-- hidden section -->
        <cell size="1">
        <!-hidden fields here -->
        </cell>
      </stack>
    </cell>
  </page>
</form>
```

3 Divide the visible section into a start section and an end section, as illustrated in the following figure:

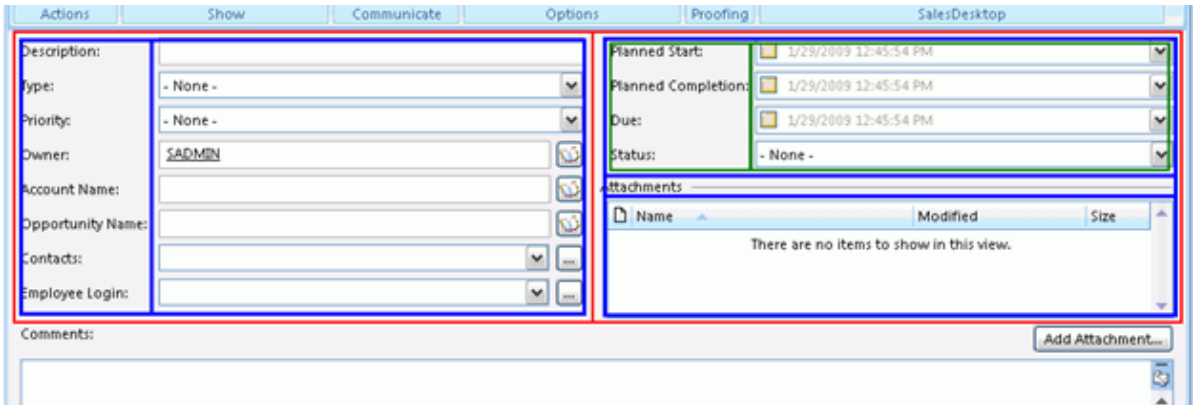


Although not required, this step helps to support the current layout of the form. Add the following code:

```
<form id="SBL Activity">
  (……)
    <!-- visible section -->
```

```
        <cell>
          <stack layout="vert" padding="5" spacing="5">
            <!-top section-->
            <cell size="220">
              <!-top section fields here-->
            </cell>
            <!-bottom section-->
            <cell>
              <!-bottom section fields here-->
            </cell>
          </stack>
        </cell>
    (..............)
</form>
```

4 Divide the start section into two parts that include a left column and a right column of fields, as illustrated in the following figure:



Add the following code:

```
<-- top section -->
<cell size=220>
  <stack layout="horz" spacing="5">
    <cell>
      <stack layout="horz" spacing="3">
      <!-- left side captions -->
        <cell size="105">
        </cell>
      <!-- left side fields -->
        <cell>                          </cell>
      </stack>
    </cell>
    <cell>
      <stack layout="vert" spacing="5">
        <cell size="105">
          <stack layout="horz" spacing="3">
          <!-- left side captions -->
            <cell size="110">
            </cell>
```

```
            <!-- left side fields -->
              <cell>
              </cell>
            </stack>
          </cell>
          <cell size="13">
            <!-attachments caption and separator here -->
          </cell>
          <cell>
            <!-attachments view here -->
          </cell>
        </stack>
      </cell>
    </stack>
  </cell>
```

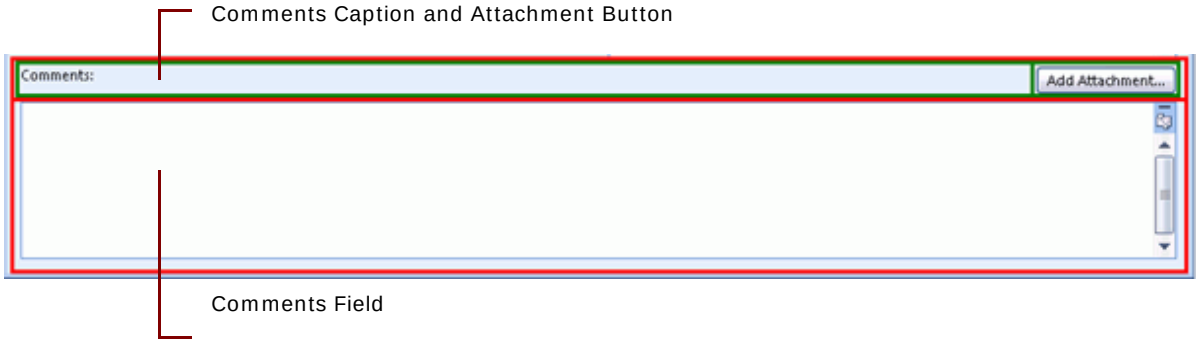
5 Divide the left column into cells where you can place captions for fields, as illustrated in the following figure:

For more information, see “XML Code That Creates Cells” on page 245.

6 Divide the right column into cells where you can place fields.

The code that you use to complete this step is similar to the code you use in Step 5. You can copy and modify the code that is described in “XML Code That Creates Cells” on page 245.

7 Define the Comments section of the Activity form, as illustrated in the following figure:



The Comments section is divided into the following parts:

- Comments caption and the Add Attachment button
- Comments field

To hold the caption and the button, the starting section includes two parts. Another section describes the Add attachment button. It is not described in the section that it relates to according to the button logic. This is not important in this step because you describe the form layout, not the button logic. For more information, see "Defining the Logic for Custom Forms" on page 251.

Add the following code:

```
<!-- visible section -->
  <cell>
    <stack layout="vert" padding="5" spacing="5">
    <!-top section-->
      <cell size="220">
      <!-top section fields here-->
      </cell>
      <!-bottom section-->
      <cell size="22">
        <stack layout="horz">
          <cell>
            <control id="0x20022" class="static" tab_order="30">
              <text>Comments:</text>
            </control>
          </cell>
          <cell size="110" attraction="far">
            <control class="button" id="0x20059" tab_order="29"
on_click="LinkAttachment()">
                                          <text>Add attachment…t</text>
                                        </control>
          </cell>
          <cell size="5" attraction="far"></cell>
        </stack>
      </cell>
      <cell>
        <control id="0x103f" tab_order="31"></control>
```

```
        </cell>
      </stack>
    </cell>
```

8 Make sure the description you create supports the version of Microsoft Outlook that your users use.

For more information, see "Correct Usage of the Forms File and Object ID" on page 246.

XML Code That Creates Cells

To create part of the layout of the form, you add the following code to the forms_12.xml file:

```
<stack layout="horz" spacing="3">
<!-- left side captions -->
  <cell size="105">
    <stack spacing="5" layout="vert" padding="4">
        <cell size="22">
          <control id="0x20002" class="static" tab_order="1">
            <text>Description:</text>
          </control>
        </cell>
        <cell size="22">
          <control id="0x20004" class="static" tab_order="3">
            <text>Type:</text>
          </control>
        </cell>
        <cell size="22">
          <control id="0x20008" class="static" tab_order="5">
            <text>Priority:</text>
          </control>
        </cell>
        <cell size="22">
          <control id="0x20010" class="static" tab_order="7">
            <text>Owner:</text>
          </control>
        </cell>
        <cell size="22">
          <control id="0x20024" class="static" tab_order="9">
            <text>Account:</text>
          </control>
        </cell>
        (……..)
      </stack>
    </cell>
<!-- left side fields -->
  <cell>
    <stack layout="vert" spacing="5">
      <cell size="22">
        <control class="edit" id="0x20003" tab_order="2">
          <field value="string">Description</field>
        </control>
      </cell>
      <cell size="22">
```

```
        <control class="combobox" id="Type" tab_order="4">
          <source type="ActionTypePicklist" field="Value" format=":[:(Label):]"></
source>
          <field>Type</field>
        </control>
      </cell>
      <cell size="22">
        <control class="combobox" id="0x20009" tab_order="6">
          <source type="ActionPriorityPicklist" field="Value"
format=":[:(Label):]"></source>
          <field>Priority</field>
        </control>
      </cell>
      <cell size="22">
        <control class="lookup" id="Owner" tab_order="8">
          <source type="Employee" format=":[:(Login Name):]"
resource_id="lookup:employees"></source>
          <field>Primary Owner Id</field>
        </control>
      </cell>
      <cell size="22">
        <control id="0x20025" tab_order="10" class="lookup">
          <source type="Account" format=":[:(Name):]"
resource_id="lookup:accounts"></source>
          <field>Account Id</field>
        </control>
      </cell>
      (……..)
    </stack>
  </cell>
</stack>
```

Correct Usage of the Forms File and Object ID

Microsoft Outlook 2003, Microsoft Outlook 2007, and Microsoft Outlook 2010 describe forms differently. These versions use different IDs for native Outlook controls. You must create a different description of the form for each version that your implementation uses. Each form uses the same layout description but the native Outlook control ID is different.

For example, assume you create code for Microsoft Outlook 2007. For this code to work correctly with Microsoft Outlook 2003, you must replace object IDs that are native in Microsoft Outlook 2007 with object IDs that are native in Microsoft Outlook 2003. In the example in this topic, you must replace the IDs that are in the hidden section and in the ID of the body control.

The code in this topic supports Microsoft Outlook 2007, so you must add it to the forms_12.xml file. If you use Microsoft Outlook 2003, then you use the forms_11.xml file and you use Object IDs that are native to Microsoft Outlook 2003.

Defining Validation Rules

This task is a step in "Process of Adding Custom Objects" on page 231.

In this topic, you add data validation rules to make sure the user enters information in the form correctly. For this example, assume an activity includes the following business requirements:

- The Owner field must contain an owner.
- The Planned Start field must contain a date.
- The date in the Planned Complete field must not occur earlier than the date in the Planned Start field.

In this topic, you use the forms_12.xml file. For more information, see "Correct Usage of the Forms File and Object ID" on page 246.

To define validation rules

- Create a validation_rules section for the form description. You add the following code to the forms_12.xml file:

```
<form id="SBL Activity">
  <validation_rules>
    <rule message="Owner is required.">
      <expression>
        <![CDATA[item["Primary Owner Id"] != null]]>
      </expression>
      <asserted_control id="Owner"></asserted_control>
    </rule>
    <rule message="Planned Start is required" expression="Planned != null">
      <asserted_control id="0x20017"></asserted_control>
    </rule>
    <rule message="#msg_activity_planned_complete_validation">
      <expression>
        <![CDATA[
          item['Planned Completion']!=null ? item['Planned Completion'] >=
          item['Planned'] : true;
        ]]>
      </expression>
      <asserted_control id="PlannedCompletion"></asserted_control>
    </rule>
  </validation_rules>
```

Defining Validation Rules for a Phone Number

This task is a step in "Process of Adding Custom Objects" on page 231.

To define validation rules for the phone number in a contact or account, you add the following regular expression to the forms_xx.xml file:

```
var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-9]{4}/);
item["Work Phone #"] != '' ? (item["Work Phone #"].match(phonetest) != null ? true
: false) : true;
```

where:

- **RegExp.** Describes the regular expression. The part of the phone number that the user enters must match the pattern that you define in the RegExp expression. If the number contains fewer than 10 digits, then Siebel CRM Desktop creates a validation failure message.
- **match(phonetest).** Causes CRM Desktop to search for the substring. The real value must not be the same as the pattern.

In this example, the following phone numbers are valid:

- 1111111111
- (999)-888-4444
- 222.888.9999

The following phone numbers are not valid:

- 1111111111 - x123
- phone: 1111111111
- +1 555.666.8888
- 11111111
- 222.aaa.8888

To define another validation, you can specify another regular expression in the rule tag of the forms_xx.xml file.

This number format is specific to telephone networks in the United States. For a different country, you must use the numbering format that is specific to that country.

To define validation rules for a phone number

1. Use an XML editor to open the forms_xx.xml file.

   For more information, see "Files That the Customization Package Contains" on page 434.

2. Add the following code to the forms_xx.xml file:

```
<rule message="#msg_business_phone_validation">
  <expression>
    <![CDATA[
var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-
9]{4}/);
item["Work Phone #"] != '' ? (item["Work Phone #"].match(phonetest)
!= null ? true : false) : true;
    ]]>
  </expression>
</rule>
<rule message="#msg_mobile_phone_validation">
  <expression>
    <![CDATA[
var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-
9]{4}/);
item["Cellular Phone #"] != '' ? (item["Cellular Phone
#"].match(phonetest) != null ? true : false) : true;
```

```
    ]]>
  </expression>
</rule>
<rule message="#msg_home_phone_validation">
  <expression>
    <![CDATA[
var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-
9]{4}/);
item["Home Phone #"] != '' ? (item["Home Phone #"].match(phonetest)
!= null ? true : false) : true;
    ]]>
  </expression>
</rule>
<rule message="#msg_fax_phone_validation">
  <expression>
    <![CDATA[
var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-
9]{4}/);
item["Fax Phone #"] != '' ? (item["Fax Phone #"].match(phonetest) !=
null ? true : false) : true;
    ]]>
  </expression>
</rule>
```

Adding Custom Logic

This task is a step in "Process of Adding Custom Objects" on page 231.

To improve usability, sometimes you must add custom logic to Siebel CRM Desktop that allows the user to manipulate Outlook data. For example:

- Automatically include the current user Id in the Owner field.
- Cause the first letter of each word in the description of an activity to automatically capitalize.
- Change the state of some controls. For example, to inform the user that the control is required.

CRM Desktop uses JavaScriptto customize Outlook data. It runs this JavaScript on events, so you define the events that are involved and you specify the code that this script calls when an event occurs. To do this, you add the following attributes to the form tag:

```
<form id="SBL Activity" on_open="form_open()" on_saving="form_saving()">
```

CRM Desktop does the following work:

- If the user opens the form, then it calls the form_open function.
- If the user saves the form, then it calls the form_saving function.

You define these functions in the forms_12.xml file. For more information, see "Correct Usage of the Forms File and Object ID" on page 246.

To add custom logic

1 Use an XML editor to open the forms_12.xml file.

For more information, see "Files That the Customization Package Contains" on page 434.

2 Add the custom logic for this example. You add the following code:

```
<form id="SBL Activity"on_open="form_open()" on_saving="form_saving()"">
  <validation_rules>
    (……..)
  </validation_rules>
  <script>
    <![CDATA[
    function form_open()
    {
      //this makes a border for these controls to inform users that
      //these fields are required
      form.Type.required = true;
      form["0x20017"].required = true;

      //this code pre-fill Owner field by Current user Id
      var defaults = find_item("::Defaults", create_criteria("or"));
      if (form.item.snapshot["Primary Owner Id"] != null)
        form.item["Primary Owner Id"] = defaults.CurrentUser;
    }
    function form_saving()
    {
      //this makes all first letters capitalized
      var fields = form.item.snapshot;
      form.item.Description = fields. Description.replace(/\b[a-z]/
g,function(w){return w.toUpperCase()});
    }
    ]]>
  </script>
```

For more information, see "Customizing Form Functions" on page 167.

Defining the Toolbar

This task is a step in "Process of Adding Custom Objects" on page 231.

For this example, it is desirable to implement some actions for the custom object on a toolbar. An action can be simple, such as attaching a note to the custom object. An action can be more complicated, such as sending an email to all contacts that are related to the custom object. In this example, you add the following buttons to the toolbar:

- Add Open in Siebel CRM
- Add attachment to the toolbar of the form
- Add New Activity to the Siebel CRM Desktop toolbar

To implement these changes, you modify the toolbars.xml file. This file typically already includes definitions for the CRM Desktop toolbar and the form toolbars, so you create only the custom buttons.

To define the toolbar

1 Encode the icon that represents the new button:

   a Create a PNG file that includes the icon that represents the button in the client.

   b Encode the PNG file for Base64.

      CRM Desktop stores the icon in the resource file in a Base64 encoded string, so you must encode the PNG file. You can use any standard Base64 encoder, such as base64.exe.

   c Remove the line breaks from the contents of the output file you created in Step b.

   d Add the contents of the output file you created in Step b to the package_res.xml file.

2 Create a new button for the CRM Desktop toolbar. You add the following code to the toolbars.xml file:

```
<toolbar caption="Siebel CRM Desktop" for="explorer">
  (…….)
  <button name="New Activity" small_image="type_image:Activity:16">
    <action class="create_item" item_type="Action"/>
  </button>
  (…….)
</toolbar>
```

3 Create a new button for the form toolbar. You add the following code to the toolbars.xml file. Note that this code uses the term *inspector* to reference the form:

```
<toolbar caption="Siebel CRM Desktop" for="inspector">
  (…….)
    <button name="Attach File" small_image="attach_btn_img">
    <action class="create_attachment" accept_type="Action">
      <attachment type="Attachment" name_field="Name" body_field="Body"
linking_field="ParentId"/>
    </action>
  </button>
  (…….)
</toolbar>
```

Defining the Logic for Custom Forms

This task is a step in "Process of Adding Custom Objects" on page 231.

In this topic, you define logic for the custom form.

To define the logic for custom forms

1 Use a JavaScript editor to open the forms.js file.

For more information, see "JavaScript Files in the Customization Package" on page 437.

2 Define your custom logic, as necessary.

A wide variety of possible customizations exist. For an overview of these customizations, see "Customizing Form Handlers" on page 166. For details, see other customization topics in this book, such as "Customizing Field Behavior" on page 177 and "Customizing UI Behavior" on page 194.

3 Register any form controls you added.

For more information, see "Registering Form Controls" on page 172.

Adding Custom Dialog Boxes

This topic describes how to add a custom dialog box. It is recommended that you define custom dialog boxes in the dialogs tag in the dialogs.xml file. This tag includes a unique dialog identifier, id, that Siebel CRM Desktop uses as the starting point for creating a dialog box. This tag includes the following sections:

- Script section where you can write the scripts that define the dialog box logic. It is recommended that you use this script section to define the dialog box logic so that the predefined forms.js file can reference the script.
- Page section where you define the dialog box layout.

The example in this topic describes how to create a dialog box that allows the user to specify child competitor information for a parent opportunity. It assumes a top-level asset object type exists, and that the basic_mapping.xml file defines the parent and child Opportunity.Competitor object.

Table 18. describes the fields of the Opportunity.Competitor object that the custom dialog box you create must display. This dialog box must include a list for each field, and it must allow the user to choose a value from each of these lists.

Table 18. Fields That the Custom Dialog Box Displays

Fields	Type Of Control	Source
CompetitorName	autocomplete	Account object type.
RelationshipRole	combobox	CompetitorRelationshipRolePicklist of the opportunity object type.
ReverseRelationshipRole	combobox	CompetitorReverseRelationshipRolePicklist of the opportunity object type.

This example links the child competitor object with an account object, so it uses an autocomplete list for the CompetitorName field to make sure the user chooses a valid link.

The combobox controls in the RelationshipRole and ReverseRelationshipRole fields make sure the user does not create a relationship between the child competitor object and a picklist value.

You configure CRM Desktop to use the account object type to populate values for the autocomplete list, and picklists to populate values for the combobox controls. You define these picklists in the basic_mapping.xml file.

For information about how to add a picklist, see Chapter 10, "Customizing Picklists."

To add custom dialog boxes

1 Use a JavaScript editor to open the business_logic.js file.

2 Add a direct link:

```
add_direct_link("Opportunity.Competitor", "Account", "CompetitorId", true,
null, false, "ObjectStatus");
```

This code adds a direct link between the Opportunity.Competitor child object and the account top-level object. It supports the list of accounts that CRM Desktop displays in the autocomplete list. For more information, see "Types of Links You Can Specify" on page 222.

3 Specify the dialog box layout.

For more information, see "Specifying the Layout of the Dialog Box" on page 256.

4 Register the controls that you added in Step 3. For more information, see "Registering Form Controls" on page 172:

a Locate the following function in the business_logic.js file:

```
function od_competitor_dlg(ctx)
{
...
}
```

This function handles the custom dialog events.

b Add the following code to the function that you located in Step a:

```
register_autocomplete_control(ctx, link_to, autocomplete_ctrl,
show_salesbook_btn)
```

where:

- `ctx` indicates to use these parameter without modification.
- `link_to` identifies the object type that CRM Desktop uses for the autocomplete list.
- `autocomplete_ctrl` specifies the Id of the autcomplete control that you defined in Step 3.
- `show_salesbook_btn` specifies the Id of the button control that you defined in Step 3. If the user clicks this button, then CRM Desktop displays the dialog box.

5 Add the trigger that displays the dialog box:

a Locate the following code:

```
with (scheme.triggers)
{
  ...
  //add trigger here
}
```

b Add the following code to the code that you located in Step a:

```
add_simple_trigger(form_helpers.create_dialog_show("Competitor Dlg", {
"form_handler": include.forms.od_competitor_dlg }), null,
"Opportunity.Competitor", null, "show");
```

For more information, see "Customizing Triggers" on page 169.

6 Save, and then close the business_logic.js file.

7 Create the function that handles the dialog box:

a Use a JavaScript editor to open the forms.js file.

b Add the following code:

```
function od_competitor_dlg(ctx)
{
  var form = ctx.form;
  var events = ctx.events;
// enable autocomplete for this item;
ctx.form_links_manager.init_new();
if (ctx.item_ex.get_id() == null)
  ctx.form.item = ctx.form.item;
}
```

You can add this code anywhere in the forms.js file. It is recommended that you add it immediately after the form handler function of the object where CRM Desktop displays this dialog box.

c Save, and then close the forms.js file.

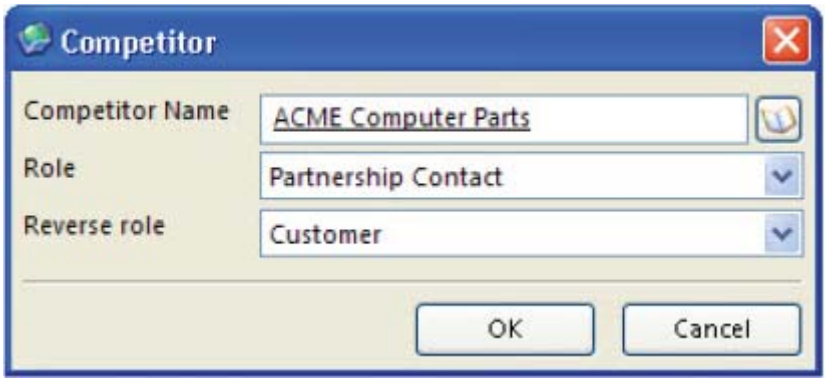
For more information about creating a function that handles a form, see "Customizing Form Handlers" on page 166.

8 Verify your work:

a Log into the client.

b Navigate to the Opportunity screen, and then click in the Competitor field.

c Verify that the client displays a dialog box that is similar to the following:



9 (Optional) Add another field to the dialog box:

a Use a JavaScript editor to open the forms_xx.xml file.

b Add the following code immediately under the code you added in Step 3:

```
<cell size="22">
  <static id="lbl_start_date" tab_order="101">
    <text>#lbl_start_date</text>
    </static>
</cell>
...
<cell size="22">
  <datetime id="startdate" tab_order="100" type="date" store_time="false">
    <field>StartDate</field>
  </datetime>
</cell>
```

where:

- `type="date"` configures CRM Desktop to display only the date without the time in the datetime contol.

c Locate the following code that you added in Step 3:

```
<appearance height="130" width="350"position="parent_center">
```

d Replace the code you located in Step c with the following code:

```
<appearance height="160" width="350" position="parent_center"></appearance>
```

This code increases the height of the dialog box to accommodate the new control.

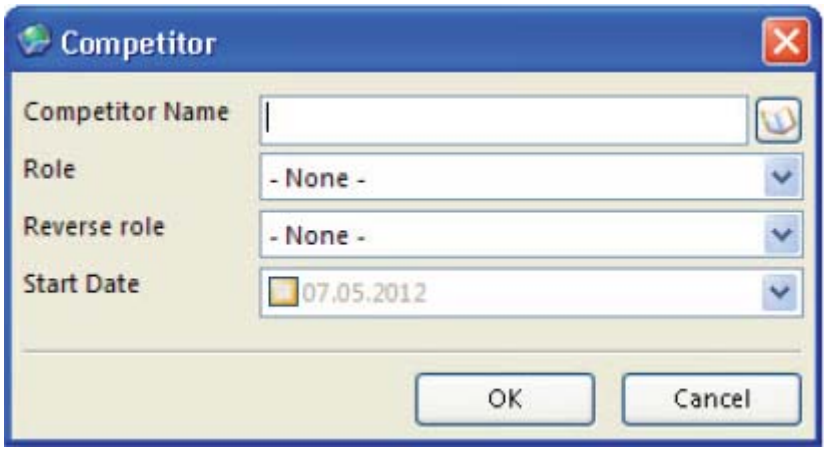
e Save the forms_xx.xml file, and then republish the customization package.

For more information, see "Republishing Customization Packages" on page 80

f Log in to the client, and then navigate to the Opportunity screen.

g Click the Competitor field.

h Verify that the client displays a dialog box that is similar to the following that includes the Start Date field:



Specifying the Layout of the Dialog Box

Figure 11 illustrates the layout of the dialog box that a page layout publisher displays when you create the layout for your custom dialog box. Each blue box represents a horizontal layout, each red box represents a vertical layout, and each light-shaded box represents a cell.

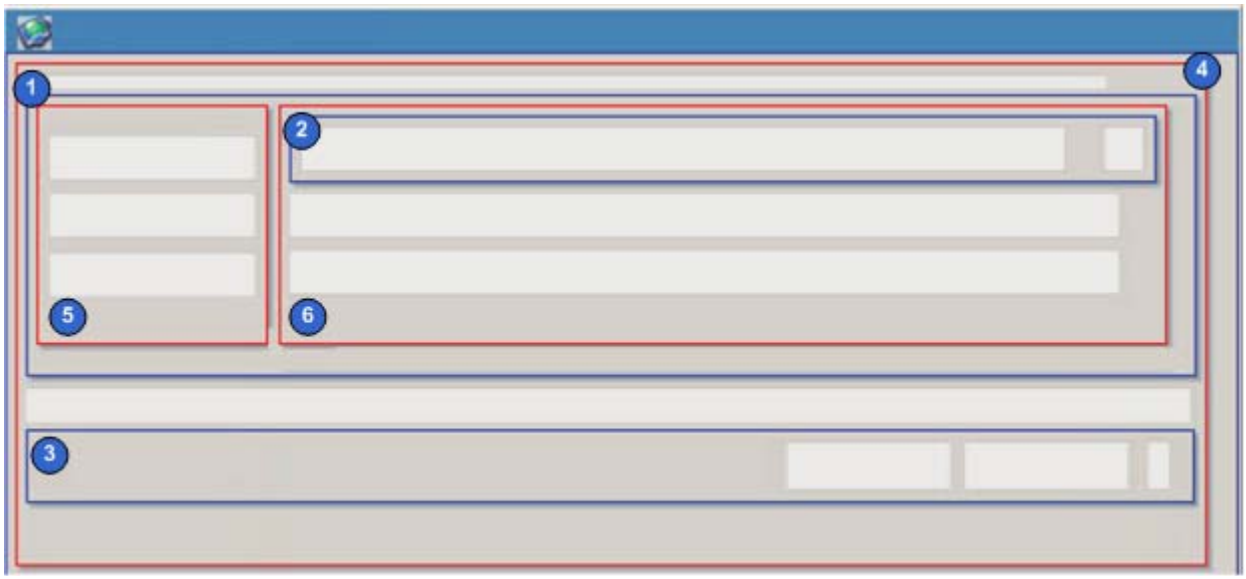


Figure 11. Layout of the Dialog Box

Explanation of Callouts

The Layout of the custom dialog box includes the following items:

1. A horizontal block that contains the controls.
2. A horizontal block that contains the autocomplete control and the button that CRM Desktop uses for the autocomplete control.
3. A horizontal block that contains the OK and Cancel buttons. This block sets the horizontal position for these buttons.
4. A vertical block that creates the start padding.
5. A vertical block that contains the controls for fields.
6. A vertical block that contains the controls for captions.

   You place the vertical blocks that contain controls for fields and captions in the first horizontal block.

For more information, see "Guidelines for Sizing Controls in Cells" on page 259.

To specify the layout of the dialog box

1. Use an XML editor to open the dialogs.xml file.
2. Add the following code. You enclose each block in a cell, and you include at least one cell that contains each control. This code starts with the first horizontal block. It also adds controls for the combobox, autocomplete list, buttons, button labels, and edge:

```
<script><script>
<dialog id="Competitor Dlg">
  <layout sizable="false" id="General" caption="#dlg_competitor_caption"
    small_icon="app_small_icon">
      <appearance height="130" width="350"position="parent_center">
      </appearance>
      <cell>
        <stack layout="horz" padding="5">
          <cell>
            <stack layout="vert" padding="5">
              <cell size="5"/>
              <cell>
                <stack layout="horz" spacing="3">
                  <cell size="100">
                    <stack layout="vert" spacing="3">
                      <cell size="22">
                        <static id="lbl_competitor_name" tab_order="1">
                          <text>#head_competitor_name</text>
                        </static>
                      </cell>
                      <cell size="22">
                        <static id="lbl_relationship_role" tab_order="2">
                          <text>#lbl_relationship_role</text>
                        </static>
                      </cell>
                      <cell size="22">
                        <static id="lbl_reverse_relationship_role" tab_order="3">
```

```
                <text>#lbl_reverse_relationship_role</text>
              </static>
            </cell>
          </stack>
        </cell>
        <cell>
          <stack layout="vert" spacing="3">
            <cell size="22">
              <stack layout="horz" spacing="2">
                <cell>
                  <autocomplete id="CompetitorId" tab_order="4">
                    <field>CompetitorId</field>
                      <source type="Account"format=":[:(AccountName):]">
                    </source>
                  </autocomplete>
                </cell>
                <cell size ="22" attraction="far">
                  <button id="btn_AccountId" image="lookup_button"
                  tab_order="5">
                    <text>...</text>
                  </button>
                </cell>
              </stack>
            </cell>
            <cell size="22">
              <combobox id="RelationshipRole" tab_order="6">
                <field>RelationshipRole</field>
                <items format=":[:(Label):]" value_column="Value"
                has_null_item="true">
                 <source type="auto"
                  name="Opportunity.CompetitorRelationshipRolePicklist">
                </source>
                <order_by>
                  <order ascend="true">SortOrder</order>
                </order_by>
              </items>
            </combobox>
          </cell>
            <cell size="22">
                <combobox id="ReverseRelationshipRole" tab_order="7">
              <field>ReverseRelationshipRole</field>
                <items format=":[:(Label):]" value_column="Value"
                has_null_item="true">
                <source type="auto"
           name="Opportunity.CompetitorReverseRelationshipRolePicklist">
              </source>
            <order_by>
                  <order ascend="true">SortOrder</order>
              </order_by>
            </items>
          </combobox>
        </cell>
      </stack>
    </cell>
```

```
          </stack>
        </cell>
        <cell size="15">
          <edge id="close_border"/>
        </cell>
        <cell size="25">
          <stack layout="horz">
            <cell>
            </cell>
            <cell size="80">
              <button id="btn_ok" tab_order="15">
                <text>#btn_ok</text>
              </button>
            </cell>
            <cell size="10"></cell>
            <cell size="80">
              <button id="btn_cancel" tab_order="16">
                <text>#btn_cancel</text>
              </button>
            </cell>
          </stack>
        </cell>
      </stack>
     </cell>
    </stack>
   </cell>
  </layout>
</dialog>
```

Guidelines for Sizing Controls in Cells

You can apply the following guidelines when you size controls in cells:

- You must place each control in a cell. The size of the cell that contains the control determines the size of the control.
- If you place a cell in a horizontal block, then you can set the horizontal size of this cell.
- If you place a cell in a vertical block, then you can set the vertical size of this cell.
- You cannot set the vertical size of a cell that you place in a horizontal block, and you cannot set the horizontal size of a cell that you place in a vertical block. A cell includes only one size attribute. CRM Desktop examines the parent layout type of the cell to determine how the size relates to the vertical or the horizontal orientation.
- If you do not set a size for a control, then CRM Desktop stretches the control so that it consumes all the available area in the cell. If you set the size for only some cells, then CRM Desktop sizes the cell according to the dimensions you specify, and expands each other cell that you do not specify size so that it consumes all the available area in the cell where it resides.
- In this example, you add a size for some cells to more accurately position the control that each cell contains.

Removing Customizations

You can remove the currently installed customization for a user and install a customization that is currently published for this user on the Siebel Server. Removing a customization allows you to fix an error that might occur as a result of this customization, such as a corrupt folder structure. It also allows Microsoft Outlook to synchronize with the Siebel Server.

To remove customizations

1 (Optional) Manually synchronize your data with the Siebel Server.

   Removing customizations uninstalls the customization and deletes all data that is not currently synchronized with the Siebel Server. To save unsynchronized data, you must manually synchronize your data with the Siebel Server before you remove the customization.

2 In the CRM Desktop client, open the CRM Desktop - Options dialog box.

3 Click the General tab.

4 In the Customization section, click Remove.

   CRM Desktop displays a dialog box that includes a message that is similar to the following:

   ```
   All data that was not synchronized to Siebel will be lost. Are you sure you want to continue?
   ```

   If you click Yes, then CRM Desktop does the following:

   - Uninstalls the current customization.
   - Deletes all data that is not currently synchronized with the Siebel Server.
   - Displays the First Run Assistant and then guides you to download the customization package from the Siebel Server.

Removing Child Objects

This topic describes an example of how to remove a child object. It describes how to remove the child notes section from the parent contact form. You comment out the object that Siebel CRM Desktop must remove in the various XML files instead of removing the object from the customization package.

To remove child objects

1 Remove the objects from the client:

   For more information about the objects that this step removes, see "Example of Removing Child Objects" on page 266:

   a Use an XML editor to open the forms_xx.xml file.

   b Locate, and then comment the following code:

```
<!-- notes view -->
.
.
.
<!-- end notes view -->
```

To comment this code, you remove the intervening comment characters. For example:

```
<!-- notes view
.
.
.
end notes view -->
```

For more information, see "Commenting in XML" on page 267.

c Use a JavaScript editor to open the forms.js file.

d Locate, and then comment the following code:

```
register_view_control_with_button(ctx, "Contact.Contact_Note","notes_view",
"btn_add_note", "btn_remove_note", null, {"created_from_ctx_type": "link_me",
"custom_view_ctrl": true });
```

To comment this code, you add the following characters at the beginning and end of the code segment:

```
/*
  .
  .
  .
/
```

For example:

```
/*
register_view_control_with_button(ctx, "Contact.Contact_Note","notes_view",
"btn_add_note", "btn_remove_note", null, {"created_from_ctx_type": "link_me",
"custom_view_ctrl": true });
/
```

For more information, see "Registering View Controls" on page 175 and "Commenting in JavaScript" on page 268.

2 Make sure CRM Desktop does not synchronize data for the objects that uses in the notes section of the view:

a Use an XML editor to open the siebel_basic_mapping.xml file.

b Locate, and then comment the following code:

```
type id="Contact.Contact_Note"
icon="type_image:Contact.Contact_Note:16">
  <field id ="Contact Id">
    <type>
      <simple type="binary"/>
    </type>
  </field>
```

```
<field id ="Note Type">
  <type>
    <simple type="string"/>
  </type>
</field>
  <field id ="Note">
    <type>
      <simple type="string"/>
    </type>
  </field>
  <field id ="Created">
    <type>
      <simple type="datetime"/>
    </type>
  </field>
  <field id ="Created By">
    <type>
      <foreign_key>
        <type_id>Employee</type_id>
      </foreign_key>
    </type>
  </field>
  <field id ="Created By Name">
    <type>
      <simple type="string"/>
    </type>
  </field>
  <field id ="NoteStatus">
    <type>
      <simple type="string"/>
    </type>
  </field>
</type
```

To comment this code, you add the following characters at the beginning and end of the code segment:

```
<!--
  .
  .
  .
-->
```

For example:

```
<!-- type id="Contact.Contact_Note"
.
.
.
</type -->
```

For more information, see “Commenting in XML” on page 267.

c Locate, and then comment the following code:

```
type id="Contact.Contact_NoteNote TypePicklist"
icon="type_image:Contact.Contact_NoteNote TypePicklist:16">
  <field id ="Label">
    <type>
      <simple type="string"/>
    </type>
  </field>
  <field id ="Value">
    <type>
      <simple type="string"/>
    </type>
  </field>
  <field id ="SortOrder">
    <type>
      <simple type="integer"/>
    </type>
  </field>
  <field id ="IsDefault">
    <type>
      <simple type="boolean"/>
    </type>
  </field>
</type
```

This step prevents CRM Desktop from processing data for the picklist object that it uses for the note object type.

d Use an XML editor to open the siebel_meta_info.xml file.

e Locate, and then comment the following code:

```
object TypeId='Contact.Contact_Note'
Label='#obj_conatct_note' LabelPlural='#obj_conatct_note_plural'
EnableGetIDsBatching='true'IntObjName='CRMDesktopContactIO'SiebMsgXmlElemNam
e='ContactNote'SiebMsgXmlCollectionElemName='ListOfContactNote' >
<viewmodes General="All" Dedup="All" />
<open_with_url_tmpl>
<![CDATA[:[:(protocol):]://:[:(hostname):]::[:(port):]/sales/_:[:(lan
g):]?SWECmd=GotoView&SWEView=Contact+Note+View&SWERF=1&
SWEHo=:[:(hostname):]&SWEBU=1&SWEApplet0=Contact+Form+Applet&SWERowId0=:[:(p
arent_id):]&SWEApplet1=Contact+Note+Applet&SWERowId1=:[:(own_id):]]]>
  </open_with_url_tmpl>
  <extra_command_options>
    <option Name='PrimaryKey1M' Value='Id' />
    <option Name='ForeignKey1M' Value='Contact Id'/>
    <option Name='Cardinality' Value='1M' />
    <option Name='ServerServiceVersion' Value='2'/>
  </extra_command_options>
<field Name='Conflict Id' Label='Conflict Id' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes'IOElemName='ConflictId' />
<field Name='Contact Id' Label='Contact Id' DataType='DTYPE_ID'
IsNullable='no' IsFilterable='no'IsRefObjId='yes' RefObjTypeId='Contact'
RefObjIsParent='yes'IsPartOfUserKey='yes' IOElemName='ContactId' />
<field Name='Created'Label='#fld_contact_contact_note@created'
DataType='DTYPE_DATETIME' IsPartOfUserKey='yes'IOElemName='Created' />
```

```
<field Name='Created By' Label='Created By' DataType='DTYPE_ID'
IsFilterable='no' IsRefObjId='yes' RefObjTypeId='Employee'
IOElemName='CreatedBy' />
<field Name='Created By Name'
Label='#fld_contact_contact_note@created_by_name' DataType='DTYPE_TEXT'
IOElemName='CreatedByName' />
<field Name='Created Date' Label='Created Date' DataType='DTYPE_UTCDATETIME'
IsHidden='yes' IOElemName='CreatedDate' />
<field Name='DS Updated' Label='DS Updated' DataType='DTYPE_DATETIME'
IsFilterable='no' IsHidden='yes' IsTimestamp='yes' IOElemName='DBLastUpd' />
<field Name='Id' Label='Id' IsPrimaryKey='yes' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes' IsPartOfUserKey='yes' IOElemName='Id' />
<field Name='Mod Id' Label='Mod Id' DataType='DTYPE_ID' IsFilterable='no'
IsHidden='yes' IOElemName='ModId' />
<field Name='Note' Label='#fld_contact_contact_note@note'
DataType='DTYPE_NOTE' IOElemName='Note' />
<field Name='Note Type' Label='#fld_contact_contact_note@note_type'
DataType='DTYPE_TEXT' HasPicklist='yes' PicklistIsStatic='yes'
PicklistCollectionType='FINS_NOTE_ALERT_MLOV'
PicklistTypeId='List_Of_Values' IOElemName='NoteType' />
<field Name='Private' Label='Private' DataType='DTYPE_BOOL' IsHidden='yes'
IOElemName='Private' />
<field Name='Updated' Label='Updated' DataType='DTYPE_DATETIME'
IsHidden='yes' IOElemName='Updated' />
<field Name='Updated By' Label='Updated By' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes' IOElemName='UpdatedBy' />
</object>
```

This step prevents CRM Desktop from processing data for the note object.

3 Modify the connector configuration:

a Use an XML editor to open the connector_configuration.xml file.

b Locate, and then comment the following code:

```
type id="Contact.Contact_Note">
<view
label="#obj_conatct_note"label_plural="#obj_conatct_note_plural"small_icon="
type_image:Note:16"normal_icon="type_image:Note:24"large_icon="type_image:No
te:48" suppress_on_top="true">
</view>
<synchronizer name_format=":[:(Note):]"threshold="10">
  <links>
    <link required="true"owned="true">Contact Id</link>
  <link>Created By</link>
  </links>
    </synchronizer>
  </type
```

This step removes the definition of the note object.

c Locate, and then comment the following code:

```
type id="Contact.Contact_Note">
  <group link="or">
    <collection container_type="Contact" foreign_key="Contact Id">
      <primary_restriction>
        <group link="and">
        </group>
      </primary_restriction>
      <container_restriction>
        <group link="and">
        </group>
      </container_restriction>
    </collection>
  </group>
</type
```

This step removes the definition of the note object from the filter presets.

4 Modify the business logic:

a Use an XML editor to open the business_logic.js file.

b Locate, and then comment the following code:

```
add_direct_link("Contact.Contact_Note", "Contact", "ContactId", true, null,
false, "NoteStatus", null, true);
```

To comment this code, you can add two forward slashes at the beginning of the code. For example:

```
//add_direct_link("Contact.Contact_Note", "Contact", "ContactId", true, null,
false, "NoteStatus", null, true);
```

This step removes the definition that CRM Desktop uses for the link between the Contact object and the Contact_Note object.

c Locate, and then comment the following code:

```
add_simple_trigger(form_helpers.create_dialog_show("Note Dlg",{
"form_handler": forms.notes_dialog, "dialog_template_params":
{"picklist_type": "Contact.Contact_NoteNote TypePicklist" } }),null,
"Contact.Contact_Note", null, "show");
```

This step removes the trigger that CRM Desktop uses to open the Note dialog box.

d Locate, and then comment the following code:

```
scheme.objects.get_object("Contact.Contact_Note").get_field("Note
Type")["initial_value_res"]
="lang_contact_note_initial_type"scheme.objects.get_object("Contact.Contact_
Note").get_field("Created")["initial_value_fn"] =
get_current_date;scheme.objects.get_object("Contact.Contact_Note").get_field
```

```
("Created By Name")["initial_value_fn"] =
get_current_user_login;scheme.objects.get_object("Contact.Contact_Note")["in
itial_links_fn"] = prefill_owner;
```

To comment this code, you can add forward slashes and asterisks. For more information, see "Commenting in JavaScript" on page 268. This step removes the definitions that CRM Desktop uses for predefault values.

Example of Removing Child Objects

Figure 12 illustrates a portion of the contact form that Siebel CRM Desktop displays before you remove the notes section.

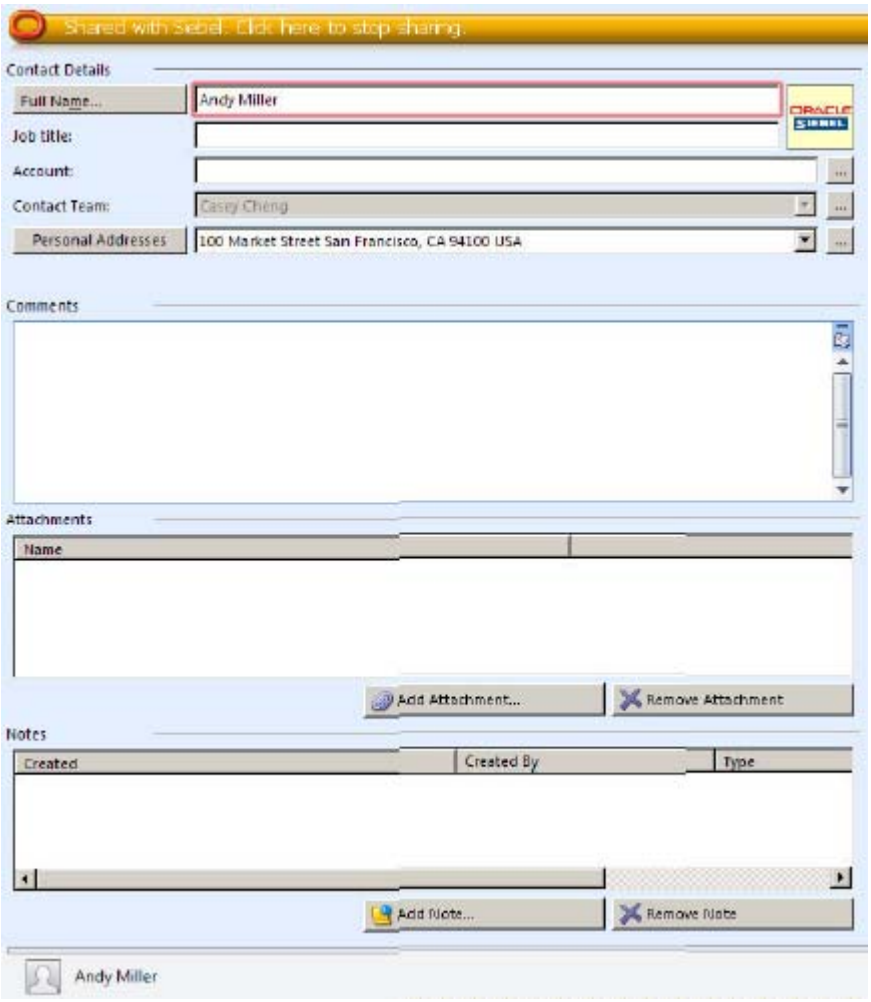


Figure 12. Notes Section of the Contact Form

Figure 13 illustrates a portion of the contact form that Siebel CRM Desktop displays after you remove the notes section.

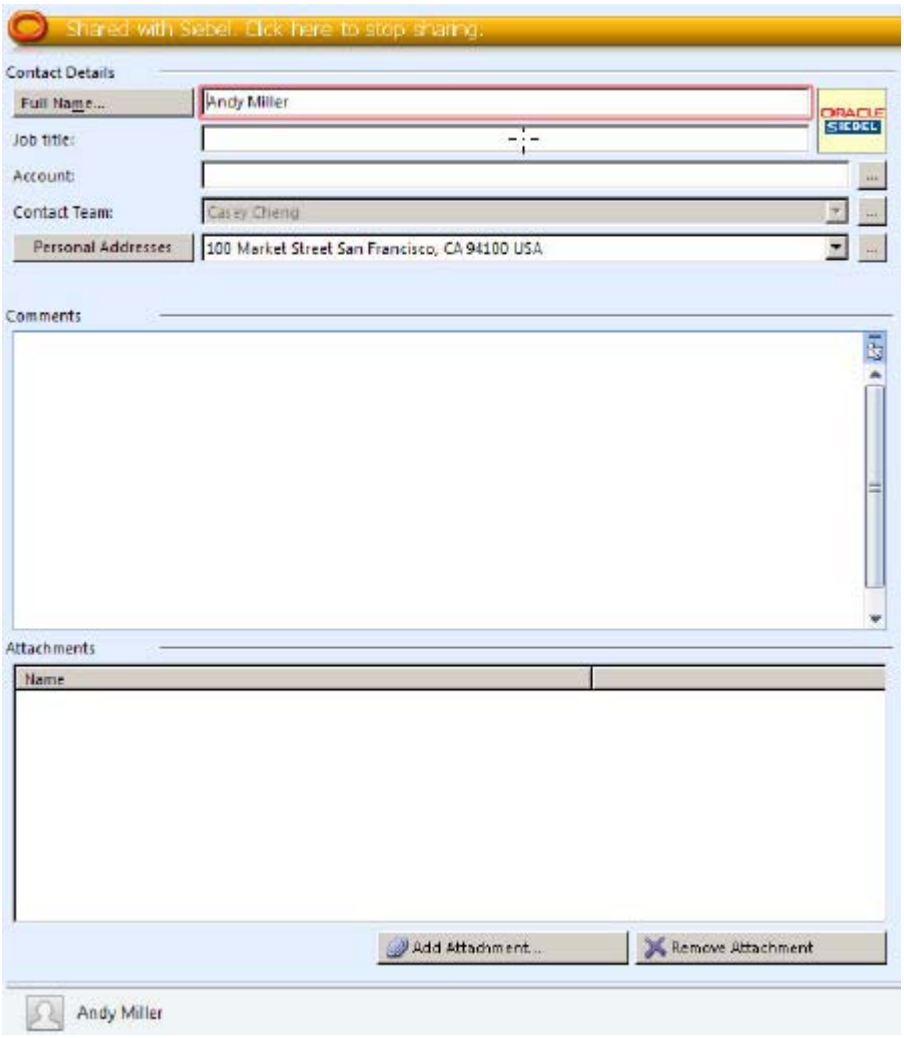


Figure 13. Contact Form That Siebel CRM Desktop Displays After You Remove the Notes Section

Commenting in XML

You can use the following syntax to add a comment In XML or HTML:

```
<!—This is a comment -->
```

An XML comment begins with the following code:

```
<!--
```

An XML comment ends with the following code:

```
-->
```

For example, consider the following code:

```
<object><field></field><field></field></object>
```

To comment this code, you use the following syntax:

```
<!-- object><field></field><field></field></object -->
```

Commenting in JavaScript

You can comment code in Javascript in two different ways. A typical XML comment begins with the following code:

```
/*
```

A typical XML comment ends with the following code:

```
*/
```

For example:

```
/* This is a comment */
```

You can also add a set of forward slashes (//) to comment an entire line of code. All code that occurs after these slashes on the line that contains the slashes is part of the comment. For example:

```
// This is a comment
```

Assume you must comment all of the following code:

```
var a = 0;
var b;
a = a + 1;
b = a;
```

The following syntax uses the slash and asterisk to comment all of the code:

```
/*
var a = 0;
var b;
a = a + 1;
b = a;
*/
```

The following syntax uses a set of forward slashes to comment all of the code:

```
// var a = 0;
// var b;
// a = a + 1;
// b = a;
```

In general, it is recommended that you use the slash and asterisk to comment multiple lines of code.

Troubleshooting Problems That Occur When You Customize Siebel CRM Desktop

To resolve a problem that occurs when you customize Siebel CRM Desktop, look for it in the list of symptoms or error messages in Table 19.

Table 19. Problems That Occur When You Customize Siebel CRM Desktop

Symptom or Error Message	Solution
After you add a new field to a CRM Desktop form, CRM Desktop displays an error that is similar to the following: `Updating error of object name object on storage {CEDC3D49-DDBB-93A2-4992-E5B2CAE62932}: object_locked (Conversion of input Message to Property Set failed with the error : Cannot convert XML Hierarchy to Integration Object Hierarchy. (SBL-EAI-04111)` This error occurs if the definition of an integration object in the Siebel Runtime Repository does not match the definition of the custom CRM Desktop form.	■ Deploy your changes to the Siebel Runtime Repository.

Debugging in Siebel CRM Desktop

In order to debug customized script in Siebel CRM Desktop, you must have a third-party debugging tool, such as the Microsoft Just-In-Time Debugger provided by Microsoft Visual Studio. For more information about Visual Studio and the Just-In-Time Debugger, see the documentation at the Microsoft TechNet Web site.

You can enable the Microsoft Just-In-Time Debugger by setting the following Windows Registry key:

```
[HKEY_CURRENT_USER\Software\Microsoft\Windows Script\Settings]
"JITDebug"=dword:00000001
```

After setting the Windows Registry key, you must restart Microsoft Outlook. If a script error is encountered after the debugger is enabled, the user will be prompted to activate the debugger. You can force the debugger to be invoked by adding "debugger" at the desired location in the custom script and republishing the metadata package.

NOTE: Oracle does not recommend enabling the Just-in Time debugger in production environments. The JIT debugger is only intended for use in development environments.

10 Customizing Picklists

This chapter describes how to customize picklists. It includes the following topics:

Overview of Customizing Picklists

You can customize the following types of picklists in Siebel CRM Desktop:

- **Static.** Values in this picklist are constant. CRM Desktop gets these values from a simple list of values. For example, the Account Status picklist includes static values, such as Candidate, Active, or Inactive. CRM Desktop typically displays a static picklist as a dropdown list that includes static values.
- **Dynamic.** Values in this picklist vary. CRM Desktop gets these values from another business component. For example, the accounts that CRM Desktop displays vary depending on if the user picks an account for a contact or picks an account for an opportunity. In this situation, the target business component might contain different values this week compared to last week because users enter new data over time. CRM Desktop typically displays a dynamic picklist as a pick applet that includes values that change.

For more information about picklists, see *Configuring Siebel Business Applications*.

Picklist Object Structure That Siebel CRM Desktop Uses

Figure 14 illustrates a hierarchy of the XML objects that Siebel CRM Desktop uses to create a picklist.

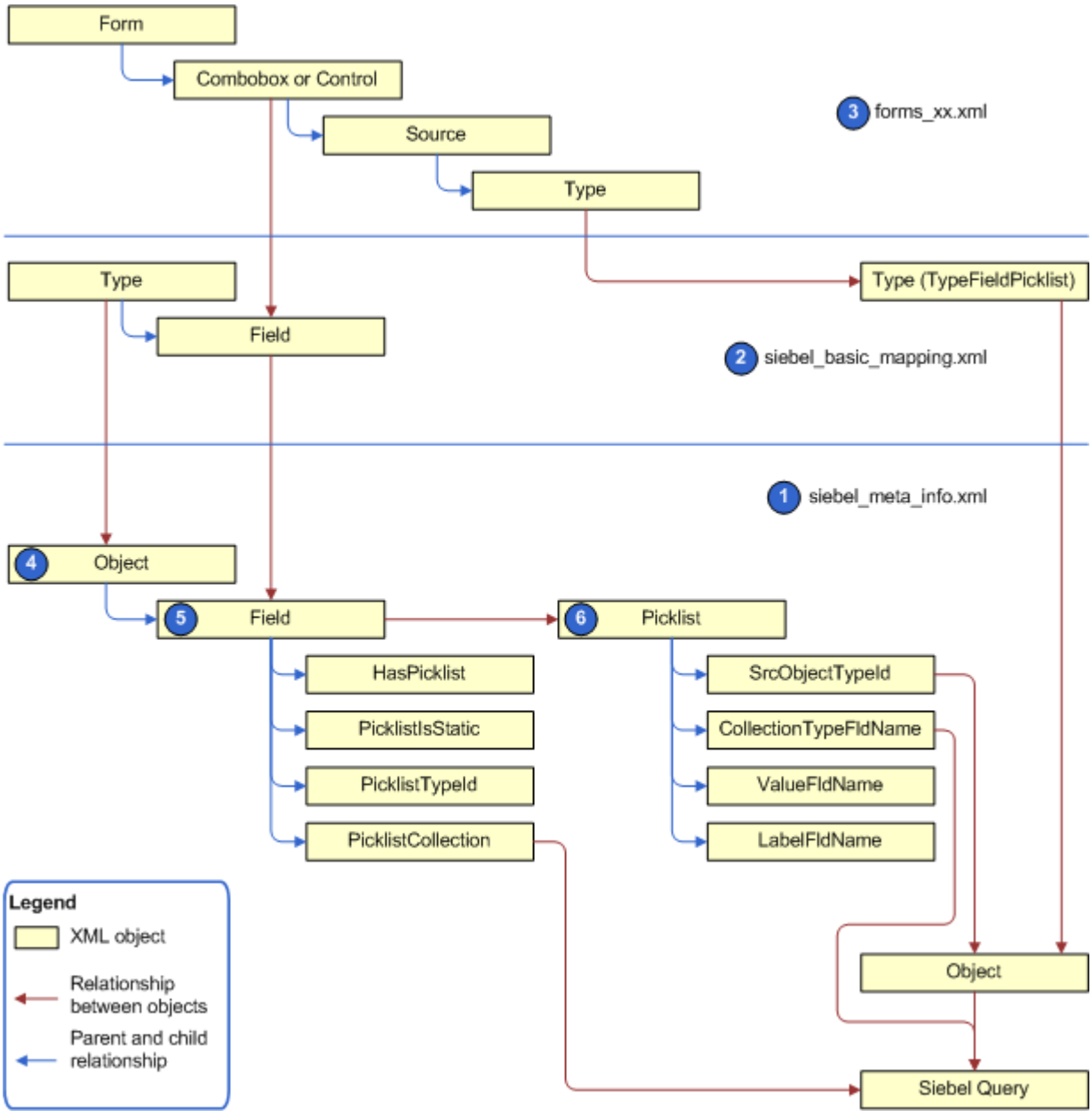


Figure 14. Picklist Object Structure That Siebel CRM Desktop Uses

Explanation of Callouts

The picklist object structure that Siebel CRM Desktop uses includes the following items:

1. **siebel_meta_info.xml.** Includes a representation of the field that CRM Desktop gets from the integration object in Siebel CRM. It defines the picklist object.
2. **siebel_basic_mapping.xml.** Maps the field that CRM Desktop gets from the integration object in Siebel CRM to the Outlook field. It includes the object that stores the picklist values in Outlook.
3. **forms_xx.xml.** Includes the definition for the user interface. You add a control for the new field and link it to the picklist data.
4. **Object.** The **CRM Desktop** representation of the integration object that it receives from Siebel CRM. For example, the Opportunity Object integration object. This object includes one or more fields.
5. **Field.** A Siebel CRM field, such as Channel. Each field includes the following attributes for the picklist:
   - **HasPicklist.** Specifies to include a picklist for the field. The value for HasPicklist must be yes.
   - **PicklistIsStatic.** Specifies to use a static picklist.
   - **PicklistTypeId.** Specifies the name of the Picklist object in CRM Desktop.
   - **PicklistCollection.** Specifies the name of the list of values in Siebel CRM that contains the values for the picklist. For example, the value of the PicklistCollection attribute for the Channel field of the Opportunity object is OPTY_CHANNEL_TYPE.
6. **Picklist.** Provides an interface to the list of values. It exposes these values and associated labels and does the translation. The Picklist object includes the following attributes:
   - **SrcObjectTypeId.** References the object in the siebel_meta_info.xml file that CRM Desktop uses to get data from Siebel CRM.
   - **CollectionTypeFldName.** Defines the field in Siebel CRM that CRM Desktop queries to use the PicklistCollection property on the object level. For example, if the PicklistCollection property on the Channel field on the Opportunity object is set to OPTY_CHANNEL_TYPE, and if the CollectionTypeFldName attribute is set to List_Of_Values, Type, then CRM Desktop uses the following query:

     ```
     [Type]='OPTY_CHANNEL_TYPE'
     ```

   - **ValueFldName.** Specifies the name of the field that CRM Desktop uses to get the values for the list of values from Siebel CRM.
   - **LabelFldName.** Specifies the name of the label for the field that the ValueFldName attribute identifies.

Modifying the Values That Predefined Static Picklists Display

This topic describes how to modify the values that a static picklist displays so that it only displays values that include a check mark in the Active property in Siebel Tools. The predefined Siebel CRM Desktop configuration displays values in this way only for the List_Of_Values picklist. Each of the following picklists displays a value even if the Active property of this value does not include a check mark:

- PickList_Generic
- PickList_Hierarchical
- PickList_Hierarchical_Child

The List_Of_Values picklist is the only object that automatically filters according to the Active property. To view code that illustrates this configuration, see "About the Predefined List of Values Object" on page 295

To modify the values that a predefined static picklist displays

1. In the client, open an activity and then click the down arrow in the Type field.
2. In the drop-down list, note that this list of values displays the following items:
   - Calendar Appointment
   - Administration
   - Adverse Event
   - Alert
   - Analyst
   - And so on

   In this example, you will modify CRM Desktop so that it does not display the Adverse Event activity type.
3. Open Siebel Tools and then display the Integration Object object type.

   For more information, see "Displaying Object Types in Siebel Tools" on page 166.
4. Add a field to the PickList Generic business component:
   a. In the Object Explorer, click Business Component.
   b. In the Business Components list, query the Name property for PickList Generic.
   c. In the Object Explorer, expand the Business Component tree and then click Field.
   d. In the Fields list, add a new field using values from the following table.

Property	Value
Name	Active
Column	ACTIVE_FLG

5. Repeat Step 4, but add the field to the PickList Hierarchical business component:
6. Add a field to the CRMDesktopPickListGenericIO integration object:
   a. In the Object Explorer, click Integration Object.
   b. In the Integration Objects list, query the Name property for CRMDesktopPickListGenericIO.

c In the Object Explorer, expand the Integration Object tree and then click Integration Object Component.

d In the Integration Components list, query the Name property for PickList Generic.

e In the Object Explorer, expand the Integration Components tree and then click Integration Component Field.

f In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	Active
Data Type	DTYPE_BOOL
External Sequence	24
External Name	Active
External Data Type	DTYPE_BOOL
XML Sequence	24
XML Tag	Active

7 Repeat Step 6, but add the field to the PickList Hierarchical integration component of the CRMDesktopPickListHierarchicalIO integration object.

8 Deploy your changes to the Siebel Runtime Repository.

9 Use an XML editor open the siebel_meta_info.xml file.

To make sure that CRM Desktop gets only the active list of values, you modify the PickList_Generic, PickList_Hierarchical, and PickList_Hierarchical_Child picklists. When you modify these picklists, you also modify the master filter so that it makes sure Active = Yes.

10 Locate the `picklist TypeId='PickList_Generic'` object and then modify it to the following code:

```
<picklist TypeId='PickList_Generic'
  SrcObjectTypeId='PickList_Generic'
  CollectionTypeFldName='Type'
  ValueFldName='Value'
  LabelFldName='Value' >
  <master_filter_expr>
    <![CDATA[
      [Active] = 'Y'
    ]]>
  </master_filter_expr>
</picklist>
```

11 Locate the `picklist TypeId='PickList_Hierarchical'` object and then modify it to the following code:

```
<picklist TypeId='PickList_Hierarchical'
  SrcObjectTypeId='PickList_Hierarchical'
  CollectionTypeFldName='Type'
```

```
  ValueFldName='Value'
  LabelFldName='Value'
  LangFldName='Language' >
  <extra_src_fldname Visible='true'>Parent</extra_src_fldname>
  <master_filter_expr>
    <![CDATA[
      [Active] = 'Y'
    ]]>
  </master_filter_expr>
</picklist>
```

12 Locate the `picklist TypeId='PickList_Hierarchical_Child'` object and then modify it to the following code:

```
<picklist TypeId='PickList_Hierarchical_Child'
  SrcObjectTypeId='PickList_Hierarchical'
  CollectionTypeFldName='Type'
  ValueFldName='Value'
  LabelFldName='Value'
  LangFldName='Language' >
  <extra_src_fldname Visible='true'>Parent</extra_src_fldname>
  <master_filter_expr>
    <![CDATA[
      [Active] = 'Y' AND NOT [Parent Id] Is Null
    ]]>
  </master_filter_expr>
</picklist>
```

13 Save the siebel_meta_info.xmlfile, upload it to the CRM Desktop Admin screen, and then add it to the active package.

14 Apply the package and then synchronize.

15 Test your changes. Repeat Step 2.

Make sure that CRM Desktop does not display the Adverse Event value in the drop-down list that the Type field uses.

Modifying the Values That Predefined Lists of Values Display

You can modify the field mapping that occurs between Siebel CRM Desktop and Outlook. The example in this topic modifies the mapping for the On Hold status. For more information, see "How Siebel CRM Desktop Maps the Status Field of an Activity" on page 446.

To modify the values that predefined lists of values display

1 Use an XML editor open the package_res.xml file.

2 Locate the following code:

```
str key="lang_action_status_on_hold" comment="Siebel SEED DATA for Action/
Status">On Hold</str>
```

3 Change the code you located in Step 2 to the following code:

```
<str key="lang_action_status_on_hold" comment="This should be taken from Siebel
SEED DATA for Action/Status">Waiting</str>
```

Process of Creating Predefined Picklists

The example in this topic adds a predefined picklist to the Contact form that allows the user to choose from a set of predefined contact methods. These methods indicate how the contact prefers to be contacted, such as through email, pager, or phone.

To add a predefined picklist, you do the following:

1 Identifying Predefined Picklist Objects in Siebel CRM on page 277.

2 Creating an Integration Object for the Contact Method Picklist on page 279

3 Extending an Integration Object for the Contact Method Picklist on page 280

4 Adding Fields to the Customization Package on page 282

5 Customizing the Physical Layout for the Pick List on page 288

6 Publishing and Testing Picklists on page 292

For more information, see "Overview of Customizing Picklists" on page 271.

Identifying Predefined Picklist Objects in Siebel CRM

This task is a step in "Process of Creating Predefined Picklists" on page 277.

In this topic, you identify the picklist objects that you use to add a picklist. These object come predefined with Siebel CRM.

To identify predefined picklist objects in Siebel CRM

1 Identify the field that Siebel CRM associates with the picklist you must add:

a Open Siebel Call Center.

b Navigate to the Contacts list and then click a name in the Last Name field.

c Click the More Info tab and then click the down arrow in the Contact Method field.

Siebel CRM displays the drop-down list for the Contact Method field. This is the drop-down list you customize in this example.

d Choose the Help menu and then click About View.

The About View dialog box lists the applets in the order that Siebel Call Center displays them.

e Note the applet name in that Siebel Call Center displays in the Contact Method drop-down list.

In this example, this is the Contact Form Applet - Child applet.

f In Siebel Tools, in the Object Explorer, click Applet.

g In the Applets list, query the Name property for the applet you noted in Step e.

This applet is Contact Form Applet - Child.

h Right-click the Contact Form Applet - Child applet, and then choose Edit Web Layout.

If Siebel Tools displays the Read-only Object dialog box, then click OK. In this task, you do not modify the form so you can use a read-only version.

i In the Applet Web Template editor, click the Contact Method control.

j In the Properties window, note the value for the following property.

Property	Value
Field	Preferred Communications

In this example, Siebel CRM associates the Preferred Communications field with the Contact Method pick list.

2 Identify the pick list:

a Open Siebel Tools.

b In the Object Explorer, click Business Component.

c In the Business Components list, query the Name property for Contact.

d In the Object Explorer, expand the Business Component tree, and then click Field.

e In the Fields list, query the Name property for Preferred Communications and then note the value for the following property.

Property	Value
PickList	Comm Media Picklist

3 Identify the business component that the pick list references:

a In the Object Explorer, click Pick List.

b In the Picklists list, query the Name property for Comm Media Picklist and then note the value for the following property.

Property	Value
Business Component	PickList Hierarchical Sub-Area

4 Identify the parent business object of the business component that the pick list references:

a In the Object Explorer, click the Flat tab and then click Business Object Component.

b In the Business Object Components list, query the Bus Comp property for PickList Hierarchical Sub-Area and then note the value for the following property.

Property	Value
Parent Business Object	CommSrv CM Channel Type PickList Administration

If your query does not return a result, then create a new business object component using values from the table in this step. If you create a new business object component, then make sure the business component where it resides is the primary business component for the business object that it references.

Creating an Integration Object for the Contact Method Picklist

This task is a step in "Process of Creating Predefined Picklists" on page 277.

In this topic, you create an integration object for the Contact Method picklist.

To create an integration object for the Contact Method picklist

1 In Siebel Tools, make sure the integration component object type is displayed.

For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2 Choose the File Menu and then click New Object.

3 Click the EAI tab, click Integration Object and then click OK.

4 In the Integration Object Builder Dialog box, choose values for the following items and then click Next.

Property	Value
Project	Choose a project. It is recommended that you create a separate project for any customization you make to Siebel CRM Desktop. For example, use a project named Siebel CRM Desktop.
Business Service	EAI Siebel Wizard

5 Choose values for the following items and then click Next.

Property	Value
Source Object	CommSrv CM Channel Type PickList Administration This value is the name of the business object.
Source Root	PickList Hierarchical Sub-Area This value is the name of the business component.
Integration Object Name	CRMDesktopPickListHierarchicalSubArea This value is the name of the integration object you are creating.

6 Accept the default values, click Next, and then click Finish.

7 In the Object Explorer, click Integration Object, query the Name property for CRMDesktopPickListHierarchicalSubArea, and then note the following property of the integration object you created in Step 6.

Property	Value
XML Tag	ListOfCrmdesktoppicklisthierarchicalsubarea

Extending an Integration Object for the Contact Method Picklist

This task is a step in "Process of Creating Predefined Picklists" on page 277.

In this topic, you extend the Contact integration component. The Contact integration component is included in multiple locations. You must extend it in each of the following integration objects:

- CRMDesktopContactIO
- CRMDesktopAccountIO
- CRMDesktopOpportunityIO

To extend an integration object for the Contact Method picklist

1 In Siebel Tools, make sure the integration component object type is displayed.

For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2 In the Object Explorer, click Integration Object.

3 In the Integration Objects list, query the Name property for CRMDesktopContactIO, and then make sure the Object Locked property contains a check mark.

4 In the Object Explorer, expand the Integration Object tree and then click Integration Component.

5 In the Integration Components list, query the Name property for Contact.

6 In the Object Explorer, expand the Integration Components tree and then click Integration Component Field.

7 In the Integration Component Fields list, add a new record with the following values.

Property	Value
Name	Preferred Communications
External Name	The value for each of these properties must match the field name on the Contact business component. In this example, Preferred Communications is the field you must reference. This is the value you noted in Step j on page 278.
Length	30 The value for this property must match the value that is set in the Length property of the Preferred Communications field.
Physical Data Type	DTYPE_TEXT
External Data Type	The value for each of these properties must match the value that is set in the Type property of the Preferred Communications field.
External Sequence	500 For more information, see "Requirements for the Sequence Property" on page 281.
XML Sequence	This value must equal the value in the External Sequence property. In this example, that value is 500.
XML Tag	PreferredCommunications The value for this property must match the field name on the Contact business component but with the spaces removed. This is the value you noted in Step j on page 278.

8 Repeat Step 3 through Step 7 for the CRMDesktopAccountIO integration object.

9 Repeat Step 3 through Step 7 for the CRMDesktopOpportunityIO integration object.

10 Compile your changes.

For more information, see *Using Siebel Tools*.

Requirements for the Sequence Property

You can enter any numeric value in the sequence property. Example properties include External Sequence and XML Sequence. This value must be unique. It must not display in the same sequence property for any other integration component in the Opportunity integration object.

Adding Fields to the Customization Package

This task is a step in "Process of Creating Predefined Picklists" on page 277.

In this topic you add a field to the customization package.

To add a field to the customization package

1 Create a working set of files for the customization package:

   a Open a Windows command line and then navigate to the directory that contains the current files of the customization package.

      For more information, see "Files That the Customization Package Contains" on page 434 and "Using the Windows Command Line to Set Optional Parameters" on page 100.

   b Create a copy of the current set of customization package files.

   c Move the original set of files to a backup directory.

      If necessary, to restore the configuration that existed before you started this customization effort, you can revert to this backup set of files.

   d Create a working set of customization package files. You rename the set of files you copied in Step b.

      For example, enter the following command:

      ```
      rename v01* v02*
      ```

      This command renames the prefix for all files in the directory that currently use v01 as the prefix. For example, it renames v01_forms_12.xml to v02_forms_12.xml. It is recommended that you use this technique to indicate that you have modified the customization package.

2 Verify that Siebel Tools added the integration object:

   a Use an XML editor open the siebel_meta_info.xml file.

      For more information, see "Files That the Customization Package Contains" on page 434.

   b Locate the PickList_Preferred_Communications object. You search for the following code:

      ```
      <object TypeId="PickList_Preferred_Communications"
      ```

   c In the header of the PickList_Preferred_Communications object, make sure the following attributes exist and with the correct value.

Attribute	Value
IntObjName	CRMDesktopPreferredCommPickList
SiebMsgXmlElemName	PicklistHierarchicalSub-Area
SiebMsgXmlCollectionElemName	ListOfCrmdesktoppreferredcommpicklist

3 Create the picklist. You add the following element to the siebel_meta_info.xml file:

```
<picklist TypeId='PickList_Preferred_Communications' CollectionTypeFld
Name='Type' SrcObjectTypeId='PickList_Preferred_Communications'
ValueFldName='Value' LabelFldName='Value' LangFldName='Language' >
  <master_filter_expr>
    <![CDATA]
      [Parent] = LookupValue ('OFFER_MEDIA', 'Package')
  ]]>
  </master_filter_expr>
</picklist>
```

For more information, see "Specifying Attributes of the Pick List Element" on page 284.

4 Add the Preferred Communications field to the Contact object:

a Locate the Contact object. You search for the following code:

```
object TypeID='Contact'
```

b Add the following code to the Contact object:

```
<field Name="Preferred Communications" Label="Preferred Communications"
DataType="DTYPE_TEXT" HasPicklist="yes" PicklistIsStatic="yes"
PicklistCollectionType="OFFER_MEDIA" PicklistTypeId="PickList Preferred
Communications" IOElemName="PreferredCommunications" />
```

5 Repeat Step 4 for each of the following objects:

- Account.Contact
- Opportunity.Contact

In this example, these objects in the siebel_meta_info.xml file must include the Preferred Communications field. You must add this field to each object.

6 Add code that creates a map for the pick list between the Siebel Server and Siebel CRM Desktop for the parent Contact object:

a Use an XML editor to open the siebel_basic_mapping.xml file.

b Create a new object type for the pick list.

For more information, see "Code That Creates a New Object Type for the Pick List" on page 285

c Locate the parent object. You search the siebel_basic_mapping.xml file for the following code:

```
<type id="Contact"
```

d Add code to the Contact object that defines a map between the Siebel Server and Siebel CRM Desktop.

For more information, see "Code That Creates a Map Between the Siebel Server and Siebel CRM Desktop" on page 286.

7 Add code that creates a map for the pick list between the Siebel Server and Siebel CRM Desktop for the child Account Contacts object:

a Choose the code from the contact object that you use to map the child account object.

For more information, see "Mapping Child Objects for a Custom Picklist" on page 286.

b Copy this code to the clipboard.

c Locate the Account Contacts child object. Search the siebel_basic_mapping.xml file for the following code:

```
type id="Account.Contact.Association"
```

d Locate the Contact field. Search in the Account.Contact.Association object for the following text:

```
field id="ContactId"
```

e Locate the matching ContactId. Search the ContactId field for the following text:

```
user_field id="sbl Contact ID"
```

f Paste the contact fields that you copied to the clipboard in Step a into the following user field:

```
sbl Contact ID
```

For more information, see "Mapping Child Objects for a Custom Picklist" on page 286.

g Map the Preferred Communications field. You add the following code immediately after the code you pasted in Step f:

```
<field from="Preferred Communications" to="ContactPreferred
Communications"></field>
```

8 Repeat Step 7 for the opportunity child object.

Specifying Attributes of the Pick List Element

If you specify a pick list element in the siebel_meta_info.xml file, then the TypeId attribute and the SrcObjectTypeId attribute of this element must match the value in the PicklistTypeId attribute. For example, assume you add the following field:

```
<field Name='Note Type' Label='#fld_account_account_note@note_type'
DataType='DTYPE_TEXT' HasPicklist='yes' PicklistIsStatic='yes'
PicklistTypeId='AccountNoteType' IOElemName='NoteType' />
```

In this example, you must set the TypeId attribute in the pick list element to AccountNoteType.

Table 20 describes values to use in the pick list element for a Siebel LOV. If you do not use a Siebel LOV, then adjust these values so they match the field names on the integration component field.

Table 20. Values to Use in the Picklist Element for a Siebel LOV

Attribute	Value
CollectionTypeFldName	Type
ValueFldName	Value
LangFldName	Language

To add more filters, you can use the master_filter_expr attribute. This attribute typically must match the value in the Search Specification property of the object definition for the pick list in Siebel Tools. In the example on Step 3 on page 280, the master_filter_expr attribute constrains the values to the correct LOV Type.

Code That Creates a New Object Type for the Pick List

To create a new object type for the pick list, you must use the following format for the type id attribute:

```
<type id="object_namefield_namePicklist" predefined_folder="1">
```

where:

- *object_name* is the name of the object type in the siebel_meta_info.xml file.
- *field_name* is the name of the field that resides in the object you define in the object_name.

For example:

```
<type id="ContactPreferred CommunicationsPicklist" predefined_folder="1">
```

To create a new object type for a pick list, add the following code to the siebel_basic_mapping.xml file:

```
<type id="ContactPreferred CommunicationsPicklist" predefined_folder="1">
  <form message_class="IPM.Contact.SBL.ContactPreferred CommunicationsPicklist"></
form>
  <field id="Label">
    <reader>
      <mapi_user><user_field id="sbl picklistLabel" ol_field_type="1"></user_field>
        <convertor><string/></convertor>
      </mapi_user>
    </reader>
    <writer>
      <Outlook_user><user_field id="sbl picklistLabel" ol_field_type="1"></
user_field>
        <convertor><string/></convertor>
      </Outlook_user>
    </writer>
    </field>
      <field id="Value">
    <reader>
      <mapi_user><user_field id="sbl picklistValue" ol_field_type="1"></user_field>
        <convertor><string/></convertor>
      </mapi_user>
    </reader>
    <writer>
      <Outlook_user><user_field id="sbl picklistValue" ol_field_type="1"></
user_field>
        <convertor><string/></convertor>
      </Outlook_user>
    </writer>
      </field>
        <field id="SortOrder">
    <reader>
      <mapi_user><user_field id="sbl SortOrder" ol_field_type="3"></user_field>
```

```
        <convertor><integer/></convertor>
      </mapi_user>
    </reader>
    <writer>
      <Outlook_user><user_field id="sbl SortOrder" ol_field_type="3"></user_field>
        <convertor><integer/></convertor>
      </Outlook_user>
    </writer>
      </field>
        <field id="IsDefault">
    <reader>
      <mapi_user><user_field id="sbl IsDefault" ol_field_type="6"></user_field>
        <convertor><bool/></convertor>
      </mapi_user>
    </reader>
    <writer>
      <Outlook_user><user_field id="sbl IsDefault" ol_field_type="6"></user_field>
        <convertor><bool/></convertor>
      </Outlook_user>
    </writer>
  </field>
</type>
```

Code That Creates a Map Between the Siebel Server and Siebel CRM Desktop

To create a map between the Siebel Server and Siebel CRM Desktop, you add the following code to the Contact object of the siebel_basic_mapping.xml file:

```
<field id="Preferred Communications">
  <reader>
    <mapi_user><user_field id="sbl Preferred Communications" ol_field_type="1"></
user_field>
    <convertor><string/></convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user><user_field id="sbl Preferred Communications"
ol_field_type="1"></user_field>
      <convertor><string/></convertor>
    </Outlook_user>
  </writer>
</field>
```

Mapping Child Objects for a Custom Picklist

It is recommended that you do not map a child object directly in the child object. Instead, you can copy values from the parent object and then paste them into the child object. This technique provides the following advantages:

- Allows Siebel CRM Desktop to copy values on the contact to the child object, such as the account or opportunity.

- If the user changes the value in a contact, then CRM Desktop automatically updates the child objects.

Example Code That Maps Child Objects for a Custom Picklist

Figure 15 illustrates the example code you use to map a child object for a custom picklist.

```
<type id="Account.Contact.Association"…>                                    (1)
…
<field id="ContactId" ver="3">                                              (2)
        <reader class="mapi_user">
(3)             <user_field id="sbl Contact ID" ol_field_type="1"></user_field>
                <convertor class="binary_hexstring"></convertor>
        </reader>
        <writer class="multiwriter">
                <writer class="outlook_user">
                        <user_field id="sbl Contact ID" ol_field_type="1"></user_field>
                        <convertor class="binary_hexstring"></convertor>
                </writer>
(4)             <writer class="link_fields">
                        <field from="DisplayName" to="ContactName"></field>
                        <field from="M/M" to="ContactM/M"></field>
                        <field from="First Name" to="ContactFirstName"></field>
                        <field from="Last Name" to="ContactLastName"></field>
                        <field from="Job Title" to="ContactJobTitle"></field>
                        <field from="Work Phone #" to="ContactWorkPhone"></field>
                        <field from="Cellular Phone #" to="ContactCellularPhone"></field>
                        <field from="Email Address" to="ContactEmailAddress"></field>
                        <field from="Status" to="ContactStatus"></field>
                        <field from="Preferred Communications" to="ContactPreferred
Communications"></field>
                </writer>
        </writer>                                                           (5)
</field>
…
</type>
```

Figure 15. Example Code That Maps Child Objects for a Custom Picklist

Explanation of Callouts

The example code to map a child object for a custom picklist includes the following items:

1 The following attribute identifies the account child object of the parent contact:

```
type id="Account.Contact.Association"
```

2 The following tag identifies the Contact field in the account object:

```
field id="ContactId"
```

3 The following attribute identifies the matching ContactId:

```
user_field id="sbl Contact ID"
```

4 You copy these fields from the parent contact object and then paste them into the account child object.

5 You add the Preferred Communications field to allow you to add it to Account Contact forms. You create this field in "Code That Creates a Map Between the Siebel Server and Siebel CRM Desktop" on page 286.

Customizing the Physical Layout for the Pick List

This task is a step in "Process of Creating Predefined Picklists" on page 277.

Figure 16 illustrates the Contact Details section of the contact form you customize in this example.

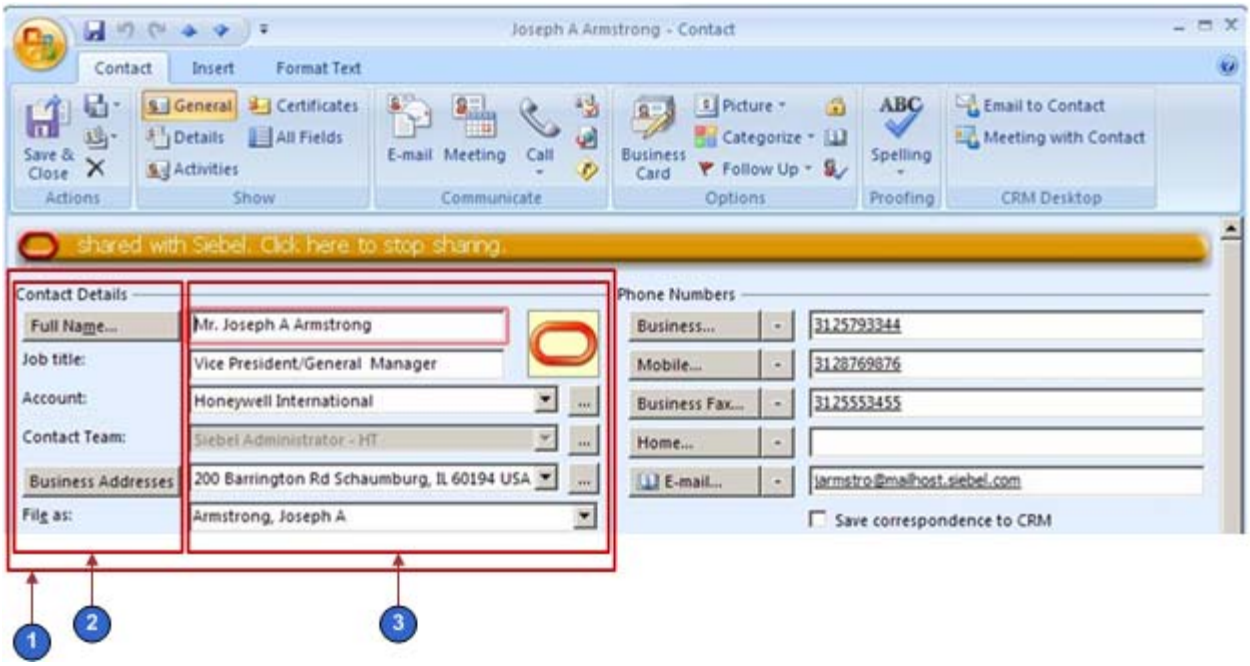


Figure 16. Contact Details Section of the Contact Form

Explanation of Callouts

The Contact form includes the following parts:

1 The following object in the forms_11.xml file identifies the Contact Details section:

```
SBL Contact
```

Siebel CRM Desktop considers this area as a single cell. This cell includes two child regions that are placed horizontally in relationship to one another.

2 The left side of the cell includes six cells that are arranged vertically.

3 The right side of the cell includes the following items:

- The Oracle oval link and the ellipses (. . .) buttons include their own cell layers.
- The starting cell includes the following items:
  - Two horizontal subcells. These subcells accommodate the Oracle oval link.
  - Two vertical subcells that accommodate the fields to the left of the Oracle oval link.
- The remaining cells following the starting cell include subcells that accommodate their picklists and ellipses (. . .) buttons.

To customize the physical layout for the pick list

1 Examine the physical location of where to place the custom field:

a Open the client and then navigate to the Contact form.

For more information, see Figure 16 on page 288.

b In the Contact Details section, locate the Business Addresses field.

In this example, you add the custom Preferred Contact Method field under the existing Business Address field.

2 Provide sufficient vertical room for the custom field and label. Increase the cell size:

a Locate the cell size attribute for the cell you must increase.

You must increase the size of the cell that contains all the other cells. In this example, examine Figure 16, and then note that you must increase the height of cells that are labeled 2 and 3. To do this, you increase the size of the cell that is labeled 1.

b Increase the cell size by 30.

For example, if the current cell size is 155, then change it to 185:

```
<cell size="185">
```

3 Add the label for the custom field:

a Use an XML editor to open the forms_11.xml file.

In this example, assume you are using Microsoft Outlook 2003. For more information, see "Customizing Forms" on page 161.

b Locate the form you must modify. Locate the following code:

```
form id="SBL Contact"
```

Each form includes the SBL prefix. The object name follows this prefix.

c Locate the following code. This code defines the label for the existing business address:

```
<cell size="22">
  <control id="dd_addresses" class="dropdown"
  caption="#head_business_addresses" tab_order="12" />
```

```
</cell>
```

Code for the label and fields is located in the following object after the validation rules:

```
SBL Contact
```

d Create the label for the new contact method. You add the following code immediately following the code you located in Step c:

```
<cell size="22">
  <static id="ContactMethod" tab_order="9">
    <text>#lbl_ContactMethod</text>
  <static>
</cell>
```

For more information, see "Code That Creates the Label for a Custom Field" on page 291.

4 Add the custom field:

a Locate the following code. This code defines the field for the existing business address:

```
<cell size="21">
    <stack layout="horz" spacing="5">
    <cell>
    <combobox id="business_address_mvg" tab_order="13">
    <field>Primary Address Id</field>
        DETAILS DELETED
      </control>
    </cell>
  </stack>
</cell>
```

To simplify this step, search for unique text, such as business_address_mvg. For brevity, *DETAILS DELETED* indicates that details for this tag are removed from this book.

For more information, see ""Adding Custom Fields" on page 182" on page 221.

b Create the field for the new contact method. You add the following code immediately following the code you located in Step a:

```
<cell size="22">
  <combobox id="cbx_ContactPreferred CommunicationsPicklist">
    <field>Preferred Communications</field>
      <source type="ContactPreferred CommunicationsPicklist" field="Value"
format=":[:(Label):]"> </source>
  </combobox>
</cell>
```

For more information, see "Code That Creates the Custom Field" on page 291, and "Combobox Control of the Forms File" on page 477.

5 Specify the custom symbolic strings:

a Use an XML editor open the package_res.xml file.

b Add the following code anywhere in the file:

```
<str key="lbl_ContactMethod">Contact Method:</str>
```

```
<str key="head_contact_method">Contact Method</str>
```

You can place these strings anywhere in the file. To assist with maintenance, it is recommended that you place them with similar strings. If necessary, to create symbolic strings that accommodate another language, you can create alternate package_resource_xml files. For more information, see "Localizing Strings" on page 223.

Code That Creates the Label for a Custom Field

To create the label for the custom contact method field, you add the following code:

```
<cell size="22">
  <static id="ContactMethod" tab_order="9">
    <text>#lbl_ContactMethod</text>
  <static>
</cell>
```

where:

- *Id* is an arbitrary, unique value for the form.
- *Tab_order* determines the order that CRM Desktop uses to position the cursor in the fields in the form when the user presses the TAB key.
- *Text* defines the text that Siebel CRM Desktop uses for the label.
- # indicates the symbolic string that the package_res.xml file defines. You can use this string for a global deployment.

Code That Creates the Custom Field

To create the custom field for the contact method, you add the following code:

```
<cell size="22">
  <combobox id="cbx_ContactPreferred CommunicationsPicklist">
    <field>Preferred Communications</field>
      <source type="ContactPreferred CommunicationsPicklist" field="Value"
format=":[:(Label):]">
      </source>
  </combobox>
</cell>
```

where:

- *Id* is an arbitrary, unique value in the form. A picklist must include the cbx prefix.
- *Tab_order* determines the order that Siebel CRM Desktop uses to place the cursor in the fields in the form when the user presses the TAB key. The value you enter here must be greater than the value you enter in the Tab_order for the label.
- *Source* determines where to get the data.
- *Type* identifies the object name. This name is defined in the siebel_basic_mapping.xml file.
- *Field* specifies the field that provides the values that the user chooses from the pick list. In this example, the Value field provides these values.

- *<Field>* identifies the field name from the object definition that populates the picklist.
- *Format* specifies how to display text in the picklist.

The format tag allows you to use a combination of static text and fields in the picklist. It uses the following format:

```
*any_static_text*:[:(field1_name):]*any_static_text*:[:(field2_name):]*any_static_text*
```

You must use the bracket, colon, and parentheses. For example:

```
Contact :[:(First Name):] :[:(Last Name):] ? Contact: John Smith
```

For more information, see ""Adding Custom Fields" on page 182" on page 221 and "Combobox Control of the Forms File" on page 477.

Publishing and Testing Picklists

This task is a step in "Process of Creating Predefined Picklists" on page 277.

In this topic, you publish and test your customization.

To publish and test a picklist

1. Publish your changes.

   For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

2. Test your changes:

   a. Open the client and then navigate to the Contact form.

   b. Verify that the form includes a label and picklist for the Contact Method field, as illustrated in the following diagram:

   c. Click the down arrow next to the Contact Method field, and then verify that Siebel CRM Desktop displays a picklist that contains the following values:

      - None
      - Chat
      - Email
      - Fax
      - Pager
      - Phone
      - Wireless Message.

   d. Choose a value in the picklist and then verify that CRM Desktop changes the value in the Contact Method field to the value you choose.

Process of Creating Custom Static Picklists

To create a custom static picklist, you do the following:

1 Modifying Siebel CRM Objects to Support Static Picklists on page 293

2 Adding Fields to the Metadata to Support Static Picklists on page 294

3 Adding Fields to the Basic Mapping to Support Static Picklists on page 295

4 Modifying the Basic Mapping to Store Values for Static Picklists on page 296

5 Modifying the Form to Support Static Picklists on page 297

6 Uploading and Testing Your Static Picklist on page 298

The example in this topic adds a static picklist to the Opportunity form in Siebel CRM Desktop. Assume that a field named JVD Simple in the Opportunity business component already exists in Siebel CRM. It references a picklist named JVD Simple Picklist. Table 21 describes the properties of the list of values in Siebel CRM that this picklist references. Although this topic uses a static picklist as an example, you can use this topic to create a static picklist or a multi-value static picklist.

Table 21. Properties of the List of Values That the JVD Simple Picklist References

Type	Display Value	Translate	Language Independent Code	Language Name
JVD_SIMPLE	Simple Value 1	Checked	Simple Value 1	English-American
JVD_SIMPLE	Simple Value 2	Checked	Simple Value 2	English-American
JVD_SIMPLE	Simple Value 3	Checked	Simple Value 3	English-American

Modifying Siebel CRM Objects to Support Static Picklists

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you modify Siebel CRM objects to support a static picklist.

To modify Siebel CRM objects to support a static picklist

1 Open Siebel Tools.

2 In the Object Explorer, click Integration Object.

3 In the Integration Objects list, query the name property for CRMDesktopOpportunityIO.

4 In the Object Explorer, expand the Integration Object tree and then click Integration Component.

5 In the Integration Components list, query the External Name Context property for Opportunity.

6 In the Object Explorer, expand the Integration Component tree and then click Integration Component Field.

7 In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Simple
Data Type	DTYPE_TEXT
Length	30
External Sequence	232
External Name	JVD Simple
External Data Type	DTYPE_TEXT
XML Sequence	232
XML Tag	JVDSimple

8 In the Object Explorer, expand the Integration Component Field tree and then click Integration Component Field User Prop.

9 In the Integration Component Field User Props list, add a new user property using values from the following table.

Property	Value
Name	PICKLIST
Value	Y

10 Deploy your changes to the Siebel Runtime Repository.

Adding Fields to the Metadata to Support Static Picklists

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you add a field to the metadata to support a static picklist.

To add a field to the metadata to support a static picklist

1 Use an XML editor open the siebel_meta_info.xml file.

2 Locate the following object:

```
TypeId="Opportunity"
```

3 Add a new field to the Opportunity object. You add the following code immediately following the code line you located in Step 2:

```
<field Name='JVD Simple' Label='JVD Simple' DataType='DTYPE_TEXT'
IsFilterable='no'HasPicklist='yes' PicklistIsStatic='yes'
PicklistCollectionType='JVD_SIMPLE'PicklistTypeId='List_Of_Values'
IsHidden='no' IOElemName='JVDSimple' />
```

where:

- **Field Name** is any name you choose. It is recommended that you use the same name that you use in Siebel CRM for this field.
- **Label** is equal to the value you provide for the Name.
- **DataType** is the data type you specify for the integration component field in Step 7 on page 294.
- **IsFilterable** is set to the default of no. IsFilterable is not relevant for this example.
- **HasPicklist** must be set to yes.
- **PicklistIsStatic** must be set to yes.
- **PicklistCollectionType** is the name of the list of values.
- **PicklistTypeId** identifies the name of the picklist type for static picklists. For more information, see "About the Predefined List of Values Object" on page 295.
- **IsHidden** must be set to no.
- **IOElemName** identifies the name you specified for the XML Tag property in Step 7 on page 294.

About the Predefined List of Values Object

The following List_of_Values picklist object comes predefined with Siebel CRM Desktop. It determines the fields to get, the field type, where to query for the language, and it makes the list of values active. It is similar to the Picklist Generic business component in Siebel CRM:

```
<picklist TypeId='List_Of_Values'
  SrcObjectTypeId='List_Of_Values'
  CollectionTypeFldName='Type'
  ValueFldName='Value'
  LabelFldName='Value'
  LangFldName='Language'>
  <master_filter_expr>
    <![CDATA[
      [Active] = 'Y'
    ]]>
  </master_filter_expr>
</picklist>
```

Adding Fields to the Basic Mapping to Support Static Picklists

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you add a field to the basic mapping to support a static picklist.

To add a field to the basic mapping to support a static picklist

1 Use an XML editor to open the siebel_basic_mapping.xml file.

2 Locate the following object:

```
id="Opportunity"
```

3 Add the following code immediately following the code line you located in Step 2:

```
<field id="JVD Simple">
  <reader>
      <mapi_user>
        <user_field id="sbl JVD Simple" ol_field_type="1"></user_field>
        <convertor>
          <string/>
        </convertor>
      </mapi_user>
    </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Simple" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

where:

- **field id** identifies the name of the integration component field you defined in Step 7 on page 294.
- **user_field id** identifies the custom field in Outlook where Siebel CRM Desktop stores the field value. It is recommended that you use sbl as the first three characters for this field Id. For example, sbl JVD Simple.

Modifying the Basic Mapping to Store Values for Static Picklists

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you modify the basic mapping to store values for a static picklist.

To modify the basic mapping to store values for a static picklist

1 Locate a predefined static list of values.

The predefined List_Of_Values object in the metadata makes sure that Siebel CRM Desktop can get the values that it displays in the dropdown list. It gets these values from Siebel CRM. It must store these values in Outlook to allow the user to work offline. To do this, it is recommended that you use one of the static list of values that comes predefined in CRM Desktop. These objects typically contain the same fields.

For example, locate the following object:

```
id='ContactStatusPicklist'
```

2 Copy the entire ContactStatusPicklist object.

3 Paste the entire ContactStatusPicklist object immediately following the object you located in Step 1.

4 Change the following values:

```
<type id="type_Id field_Id Picklist" predefined_folder="1" ver="1">
<form message_class="IPM.Contact.SBL.type_Id_field_Id"></form>
```

where:

- *type_Id* identifies the Id of the type, such as Opportunity.
- *field_Id* identifies the Id of the field you specified in Step 3 on page 296, such as JVD Simple.

In the message_class attribute, replace any spaces that exist in the type_Id and the field_Id with an underscore (_).

For example:

```
<type id="OpportunityJVD SimplePicklist" predefined_folder="1" ver="1">
<form message_class="IPM.Contact.SBL.OpportunityJVD_SimplePicklist"></form>
```

If you do not correctly identify the type Id, then CRM Desktop does not display any picklist values in the dropdown list at run time.

The code in this book does not include the entire object. It includes only the items you must change for this example.

Modifying the Form to Support Static Picklists

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you add fields to the form to support a static picklist.

To modify the form to support a static picklist

1 Use an XML editor to open the forms_xx.xml file.

For more information about the forms_xx.xml file, see "Files in the Customization Package" on page 434.

2 Locate the Opportunity form. Locate the following code:

```
<form id="SBL Opportunity">
```

3 Add the following code to the Opportunity form:

```
<combobox id="Id_name">
  <field>field_name</field>
  <source type="lov_type_Id" field="source_field_name"
    format=":[:(Label):]"></source>
</combobox>
```

where:

- *Id_name* is any text that identifies the purpose of the dropdown list.
- *field_name* identifies the field you specify in the basic mapping in Step 3 on page 296.
- *lov_type_Id* identifies the object that stores the list of values in the basic mapping that you specify in Step 4 on page 297.
- *source_field_name* identifies the field from the Picklist object that Siebel CRM Desktop displays in the client.

For example, you add the following code:

```
<combobox id="jvd_simple">
  <field>JVD Simple</field>
  <source type="OpportunityJVD SimplePicklist" field="Value"
    format=":[:(Label):]"></source>
</combobox>
```

For more information, see "Combobox Control of the Forms File" on page 477.

Uploading and Testing Your Static Picklist

This task is a step in "Process of Creating Custom Static Picklists" on page 293.

In this topic you upload and test your static picklist.

To upload and test your static picklist

1 Publish the following files that you modified:

- siebel_meta_info.xml
- siebel_basic_mapping.xml
- forms_xx.xml

For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

2 Open the client and then navigate to the Opportunity form.

3 Make sure this form includes a new field named JVD Simple.

4 Make sure the JVD Simple field includes a picklist and that this picklist includes the values listed in the Display Value column in Table 21 on page 293.

Creating Static Picklists That Use Long Values

The Display Value property of a list of values in Siebel CRM is limited to 30 characters. If you use the Display Value property as the source of the values that Siebel CRM Desktop displays in a static picklist, then each of these value cannot exceed 30 characters in length. The Description property of a list of values in Siebel CRM can contain up to 255 characters. The example in this topic describes how to use this Description property to create a list of values that includes values that are longer than 30 characters in length.

The example in this topic assumes that a field named JVD Simple in the Opportunity business component already exists in Siebel CRM and that this field references a picklist named JVD Simple Picklist. Table 21 on page 293 describes the properties of the list of values that this picklist references. In addition, Table 22 describes values for the Description property.

Table 22. Properties of the List of Values That the JVD Simple Picklist References for Long Values

Type	Display Value	Description
JVD_SIMPLE	Simple Value 1	Description for Simple Value 1
JVD_SIMPLE	Simple Value 2	Description for Simple Value 2
JVD_SIMPLE	Simple Value 3	Description for Simple Value 3

To create a static picklist that uses long values

1 Do all the work described in "Process of Creating Custom Static Picklists" on page 293 with the following modifications:

- In Step 3 on page 294 you set the following:

  ```
  PicklistTypeId='List_Of_Values_Description
  ```

- Do not do "Uploading and Testing Your Static Picklist" on page 298 at this time.

2 Use an XML editor open the siebel_meta_info.xml file.

3 Locate and then make a copy of the predefined list of values.

For more information, see "About the Predefined List of Values Object" on page 295.

4 Modify the copy you made in Step 3 to the following:

```
<picklist TypeId='List_Of_Values_Description'
  SrcObjectTypeId='List_Of_Values'
  CollectionTypeFldName='Type'
  ValueFldName='Description'
  LabelFldName='Description'
```

```
    LangFldName='Language'>
    <master_filter_expr>
      <![CDATA[
        [Active] = 'Y'
      ]]>
    </master_filter_expr>
  </picklist>
```

Note the following:

- Bold text indicates the values that you must modify for this example.
- You must change the value in the TypeId attribute to a unique value, such as List_Of_Values_Description.
- The predefined list of values includes a ValueFldName attribute that is set to Value and a LabelFldName attribute that is set to Value. You must change the values of these attributes so that they reference the Description property.

5 Do "Uploading and Testing Your Static Picklist" on page 298 with the following modification:

- Make sure the JVD Simple field includes a picklist and that this picklist includes the values listed in the Description column in Table 22 on page 299.

Process of Creating Dynamic Picklists

This topic describes how to create a dynamic picklist that uses predefined objects that already exist in Siebel CRM. You do the following work to create a dynamic picklist:

1 Modifying Siebel CRM Objects to Support Dynamic Picklists on page 300

2 Modifying the Metadata, Basic Mapping, and Forms to Support Dynamic Picklists on page 302

A dynamic picklist does not get values from the List Of Values table. It gets values from a business component that resides in Siebel CRM. The example in this topic creates a dynamic picklist that changes values according to the opportunity that the user chooses. For more information, see "Overview of Customizing Picklists" on page 271.

Modifying Siebel CRM Objects to Support Dynamic Picklists

This task is a step in "Process of Creating Dynamic Picklists" on page 300.

In this topic you modify Siebel CRM objects to support a dynamic picklist.

To modify Siebel CRM objects to support a dynamic picklist

1 Open Siebel Tools.

2 In the Object Explorer, click Business Component.

3 In the Business Components list, query the name property for Opportunity.

4 In the Object Explorer, expand the Business Component tree and then click Business Component Field.

5 In the Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Industry
Join	S_INDUST
Column	NAME
Type	DTYPE_TEXT
Picklist	PickList Industry

6 In the Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Industry Id
Join	S_OPTY_X
Column	ATTRIB_03
Type	DTYPE_TEXT

7 Extend the integration object with the new industry field. Do "Modifying Siebel CRM Objects to Support Static Picklists" on page 293 with the following modifications:

- In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Industry
Data Type	DTYPE_TEXT
Length	50
External Sequence	237
External Name	JVD Industry
External Data Type	DTYPE_TEXT
XML Sequence	237
XML Tag	JVDIndustry

- Do not add an integration component field user property.

Modifying the Metadata, Basic Mapping, and Forms to Support Dynamic Picklists

This task is a step in "Process of Creating Dynamic Picklists" on page 300.

In this topic you add fields to the metadata, basic mapping, and forms to support a dynamic picklist.

To modify the metadata, basic mapping, and forms to support a dynamic picklist

1 Add the picklist field to the metadata. Do "Adding Fields to the Metadata to Support Static Picklists" on page 294 with the following modifications:

   a Add the following code:

```
<field Name='JVD Industry' Label='JVD Industry' DataType='DTYPE_TEXT'
HasPicklist='yes'PicklistIsStatic='yes'PicklistTypeId='PickList_Industry'
IOElemName='JVDIndustry'/>
```

   Note that this code does not include the CollectionType attribute because a dynamic picklist does not use the list of values that a static picklist uses. It also includes the PickList_Industry picklist type.

   b Create the new PickList_Industry picklist type. You add the following code immediately after the code you added in Step a:

```
<picklist TypeId='PickList_Industry'
  SrcObjectTypeId='Industry' ValueFldName='Name'
LabelFldName='Name' />
```

   This picklist definition is not as complex as the picklist definition you use to create a static picklist. A dynamic picklist does not use a list of values and does not require the CollectionTypeFldName attribute. This example uses the predefined Industry object to get the picklist values.

2 Add the JVD Industry field to the basic mapping. Do "Adding Fields to the Basic Mapping to Support Static Picklists" on page 295, but use the following code:

```
<field id="JVD Industry">
  <reader>
    <mapi_user>
      <user_field id="sbl JVD Industry" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Industry" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
```

```
      </Outlook_user>
    </writer>
  </field>
```

Bold text identifies the code that is different for a dynamic picklist.

3 Create the type that stores the list of values. Do "Modifying the Basic Mapping to Store Values for Static Picklists" on page 296, but use the following code:

```
<type id="OpportunityJVD IndustryPicklist"
predefined_folder="1" ver="1">
<form message_class="IPM.Contact.SBL.OpportunityJVD_IndustryPicklist"></form>
```

Bold text identifies the code that is different for a dynamic picklist. For more information about doing this step, see Step 4 on page 297.

4 Add the field to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<combobox id="jvd_industry">
  <field>JVD Industry</field>
  <source type="OpportunityJVD IndustryPicklist" field="Value"
    format=":[:(Label):]"></source>
</combobox>
```

Bold text identifies the code that is different for a dynamic picklist. For more information, see "Combobox Control of the Forms File" on page 477.

5 Upload and publish your work. Do "Uploading and Testing Your Static Picklist" on page 298.

6 Test your work:

a Open the client and then navigate to the opportunity form.

b Make sure the form displays the Industry picklist.

c Choose the Industry picklist and then make sure it displays the appropriate values.

The picklist must display all the industries that are appropriate for the opportunity that Siebel CRM Desktop currently displays. For example:

- Banks
- Basic Materials
- Beef Cattle Feedlots
- Beef Cattle except feedlots
- Berry Crops
- And so on

Process of Creating Dynamic Picklists That Use Custom Objects

This topic describes how to create a dynamic picklist that uses custom objects that you define. You do the following work to add a dynamic picklist that uses custom objects:

1. Modifying the Business Component on page 304
2. Creating an Integration Object on page 305
3. Modifying Siebel CRM Desktop to Support the New Integration Object on page 308
4. Modifying the Remaining Siebel CRM Desktop Objects on page 310

For more information, see "Overview of Customizing Picklists" on page 271.

Modifying the Business Component

This task is a step in "Process of Creating Dynamic Picklists That Use Custom Objects" on page 304.

In this topic you modify the Opportunity business component to support a dynamic picklist that uses custom objects.

To modify the business component

1. Open Siebel Tools.
2. In the Object Explorer, click Business Component.
3. In the Business Components list, query the name property for Opportunity.
4. In the Object Explorer, expand the Business Component tree and then click Business Component Field.
5. In the Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Fulfillment Center
Join	S_ORG_FUL
Column	NAME
Type	DTYPE_TEXT
Picklist	eAuto PickList Business Rule

6 In the Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Fulfillment Center Id
Join	S_OPTY_X
Column	ATTRIB_06
Type	DTYPE_TEXT

Creating an Integration Object

This task is a step in "Process of Creating Dynamic Picklists That Use Custom Objects" on page 304.

The Fulfillment Center is not available as an object in predefined Siebel CRM Desktop. To make it available, you create an integration object.

To create an integration object

1 In Siebel Tools, choose the File menu and then click New Object.

2 In the New Object Wizards dialog box, click the EAI tab, click Integration Object, and then click OK.

3 In the Integration Object Builder dialog box, set values using information from the following table and then click Next.

Property	Value
Project	Choose the project you use for this development effort.
Business Service	EAI Siebel Wizard

4 In the next screen of the Integration Object Builder dialog box, set values using information from the following table and then click Next.

Property	Value
Source Object	Fulfillment Center
Source Root	Fulfillment Center
Integration Object Name	CRMDesktopFulfillmentCenterIO

5 In the Integration Object Builder - Choose Integration Components dialog box, do the following:

a Expand the Fulfillment Center tree.

b Remove the check mark from the Fulfillment Center_Position check box.

c Click Next and then click Finish.

Siebel Tools creates and then displays the new integration object.

6 (Optional) Make the XML Tag property consistent with the other integration objects that Siebel CRM Desktop uses. You change properties using values from the following table.

Property	Value
XML Tag	ListOfCRMDesktopFulfillmentCenterIO

7 Make fields that Siebel CRM Desktop does not require inactive:

a In the Object Explorer, expand the Integration Object tree, expand the Integration Component Tree, and then click Integration Component Field.

b In the Integration Component Fields list, set the Inactive property to True for each of the following fields:

- Description
- Main Fax Number
- Main Phone Number
- Primary Position Id
- UIActive
- UISelected
- operation
- searchspec

The Integration Object Builder wizard creates an integration component that includes all fields that the business component includes, by default. You can remove the fields that Siebel CRM Desktop does not require to make web service calls more efficient.

8 In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	DS Updated
Data Type	DTYPE_DATETIME
Length	30
External Sequence	10
External Name	DS Updated
External Data Type	DTYPE_DATETIME
XML Sequence	10
XML Tag	DBLastUpd

Each integration component that Siebel CRM Desktop uses must include the DS Updated field.

9 In the Object Explorer, click Integration Component Key.

Although the Integration Object Builder wizard creates a key for the integration component that it creates, Siebel CRM Desktop requires more keys to support the synchronization process.

10 Create the modification key:

a In the Integration Component Keys list, add a new key using values from the following table.

Property	Value
Name	Modification Key
Key Sequence Number	1
Key Type	Modification Key

b In the Object Explorer, expand the Integration Component Key and then choose Integration Component Key Field.

c In the Integration Component Key Fields list, create two new fields using values from the following table.

Name	Field Name
DBLastUpd	DS Updated
Mod Id	Mod Id

11 Create the primary key:

a In the Integration Component Keys list, add a new key using values from the following table.

Property	Value
Name	Primary Key
Key Sequence Number	1
Key Type	User Key

b In the Integration Component Key Fields list, add a new field using values from the following table.

Name	Field Name
Id	Id

12 Create the status key:

a In the Integration Component Keys list, add a new key using values from the following table.

Property	Value
Name	Status Key
Key Sequence Number	1
Key Type	Status Key

b In the Integration Component Key Fields list, add four new fields using values from the following table.

Name	Field Name
DBLastUpd	DS Updated
Id	Id
Mod Id	Mod Id
Name	Name

13 Deploy your changes to the Siebel Runtime Repository.

Modifying Siebel CRM Desktop to Support the New Integration Object

This task is a step in "Process of Creating Dynamic Picklists That Use Custom Objects" on page 304.

If you expose a new object to Siebel CRM Desktop, then you must create a new type in the connector_configuration.xml file. This file defines the objects that Siebel CRM Desktop synchronizes. For more information, see "Files in the Customization Package" on page 434.

To modify Siebel CRM Desktop to support the new integration object

1 Use an XML editor open the connector_configuration.xml file and then add the following code:

```
<type id="Fulfillment Center">
  <view label="Fulfillment Center"
  label_plural="Fulfillment Centers" small_icon="type_image:Generic:16"
  normal_icon="type_image:Generic:24"
  large_icon="type_image:Generic:48"
  suppress_sync_ui="true"></view>
  <synchronizer name_format=":[:(Name):]"
    frequency="604800" threshold="0">
    <links></links>
  </synchronizer>
</type>
```

2 Create a new metadata type to support the new integration object. You use an XML editor open the siebel_meta_info.xml file and then add the following code:

```
<object TypeId='Fulfillment Center' Label='Fulfillment Center'
LabelPlural='Fulfillment Centers' ViewMode='All' EnableGetIDsBatching='true'
IntObjName='CRMDesktopFulfillmentCenterIO'
SiebMsgXmlElemName='FulfillmentCenter'
SiebMsgXmlCollectionElemName='ListOfCRMDesktopFul fillmentCenterIO' >

<prohibit_operation AnyMod='yes' ErrMsg='#mod_operation_is_prohibited_err_msg'/>

<extra_command_options>
  <option Name='PrimaryKey1M' Value='Id' />
  <option Name='ForeignKey1M' Value='Id' />
  <option Name='Cardinality' Value='1M' />
  <option Name='ServerServiceVersion'Value='2' />
</extra_command_options>

<field Name='Conflict Id' Label='Conflict Id' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes'IOElemName='ConflictId' />

<field Name='Created' Label='Created' DataType='DTYPE_DATETIME'
IOElemName='Created' />

<field Name='Created By' Label='Created By' DataType='DTYPE_ID' IsFilterable='no'
IsHidden='yes'IOElemName='CreatedBy' />

<field Name='DS Updated' Label='DS Updated' DataType='DTYPE_DATETIME'
IsFilterable='no' IsHidden='yes' IsTimestamp='yes' IOElemName='DBLastUpd' />

<field Name='Id' Label='Id' IsPrimaryKey='yes' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes' IsPartOfUserKey='yes' IOElemName='Id' />

<field Name='Mod Id' Label='Mod Id' DataType='DTYPE_ID' IsFilterable='no'
```

```
IsHidden='yes' IOElemName='ModId' />

<field Name='Name' Label='Name' DataType='DTYPE_TEXT' IsPartOfUserKey='yes'
IOElemName='Name' />

<field Name='Updated' Label='Updated' DataType='DTYPE_DATETIME' IsHidden='yes'
IOElemName='Updated' />

<field Name='Updated By' Label='Updated By' DataType='DTYPE_ID' IsFilterable='no'
IsHidden='yes' IOElemName='UpdatedBy' />
</object>
```

3 Create the basic mapping object to support the new integration object. You use an XML editor to open the siebel_basic_mapping.xml file and then add the following code:

```
<type id="Fulfillment Center" hidden_folder="true" folder_type="10"
display_name="Fulfillment Center">
  <form message_class="IPM.Contact.SBL.Fulfillment_Center"
    display_name="Fulfillment Center"
    icon="type_image:Generic:16"></form>
  <field id="Name">
    <reader>
      <mapi_std>
        <mapi_tag id="0x3A110000"></mapi_tag>
        <convertor>
          <string/>
        </convertor>
      </mapi_std>
    </reader>
    <writer>
      <Outlook_std>
        <Outlook_field id="LastName"></Outlook_field>
        <convertor>
          <string/>
        </convertor>
      </Outlook_std>
    </writer>
  </field>
</type>
```

When Siebel CRM Desktop synchronizes data it uses this code to determine where to store mapping data in Outlook.

Modifying the Remaining Siebel CRM Desktop Objects

This task is a step in "Process of Creating Dynamic Picklists That Use Custom Objects" on page 304.

The remaining tasks you must perform are nearly identical to the tasks you perform to add a dynamic picklist. The main difference is you must use the JVD Fulfillment Center field name. For more information, see "Process of Creating Dynamic Picklists" on page 300.

To modify the remaining Siebel CRM Desktop objects

1 Extend the integration object with the new industry field. Do "Modifying Siebel CRM Objects to Support Static Picklists" on page 293 with the following modifications:

- In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Fulfillment Center
Data Type	DTYPE_TEXT
Length	50
External Sequence	237
External Name	JVD Fulfillment Center
External Data Type	DTYPE_TEXT
XML Sequence	237
XML Tag	JVDFulfillmentCenter

- Do not add an integration component field user property.

2 Add the picklist field to the metadata. Do "Adding Fields to the Metadata to Support Static Picklists" on page 294 with the following modifications:

a Add the following code:

```
<field Name='JVD Fulfillment Center' Label='JVD Fulfillment Center'
DataType='DTYPE_TEXT'
HasPicklist='yes'PicklistIsStatic='yes'PicklistTypeId='PickList_FulfillmentC
enter' IOElemName='JVDFulfillmentCenter'/>
```

b Create the new PickList_Industry picklist type. You add the following code immediately after the code you added in Step a:

```
<picklist TypeId='PickList__FulfillmentCenter'
  SrcObjectTypeId='Fulfillment Center' ValueFldName='Name'
LabelFldName='Name' />
```

3 Add the JVD Industry field to the basic mapping. Do "Adding Fields to the Basic Mapping to Support Static Picklists" on page 295, but use the following code:

```
<field id="JVD Fulfillment Center">
  <reader>
    <mapi_user>
      <user_field id="sbl JVD Fulfillment Center" ol_field_type="1"></
user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
```

```
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Fulfillment Center" ol_field_type="1"></
user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

Bold text identifies the code that is different for a dynamic picklist.

4 Create the type that stores the list of values. Do "Modifying the Basic Mapping to Store Values for Static Picklists" on page 296, but use the following code:

```
<type id="OpportunityJVD FulfillmentCenterPicklist"
predefined_folder="1" ver="1">
<form
message_class="IPM.Contact.SBL.OpportunityJVD_FulfillmentCenterPicklist"></
form>
```

Bold text identifies the code that is different for a dynamic picklist. For more information about doing this step, see Step 4 on page 297.

5 Add the field to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<combobox id="jvd_fulfillmentcenter">
  <field>JVD Fulfillment Center</field>
  <source type="OpportunityJVD FulfillmentCenter" field="Value"
    format=":[:(Label):]"></source>
</combobox>
```

Bold text identifies the code that is different for a dynamic picklist. For more information, see "Combobox Control of the Forms File" on page 477.

6 Upload and publish your work. Do "Uploading and Testing Your Static Picklist" on page 298.

7 Test your work:

a Open the client and then navigate to the opportunity form.

b Make sure the form displays the Fulfillment Center picklist.

c Choose the Fulfillment Center picklist and then make sure it displays the appropriate values.

The picklist must display all the fulfillment centers that are appropriate for the opportunity that Siebel CRM Desktop currently displays. For example:

- Concord Fulfillment Center
- Copy Center
- Marketing Department
- And so on

Process of Creating Dynamic Picklists That Use a SalesBook Control

This topic describes how to create a dynamic picklist that uses a salesbook control. You do the following work to add a dynamic picklist that uses a salesbook control:

1. Modifying Siebel CRM Objects to Support a Dynamic Picklist That Uses a SalesBook Control on page 313
2. Modifying the Metadata on page 314
3. Modifying the Basic Mapping and Connector Configuration on page 315
4. Defining the View on page 317
5. Modifying the Business Logic and Testing Your Work on page 319
6. (Optional) Defining Multiple Linked Fields on page 320

For more information, see "Overview of Customizing Picklists" on page 271.

Modifying Siebel CRM Objects to Support a Dynamic Picklist That Uses a SalesBook Control

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

In this topic you modify Siebel CRM objects to support a dynamic picklist that uses a salesbook control.

To modify Siebel CRM objects to support a dynamic picklist that uses a salesbook control

1. Do all the work described in "Process of Creating Dynamic Picklists That Use Custom Objects" on page 304.

   To add a dynamic picklist that uses a salesbook control, you reuse a number of the objects that you configure when you create a dynamic picklist that uses custom objects.

2. Activate the following integration component fields that you deactivated in Step 7 on page 306:
   - Description
   - Main Fax Number
   - Main Phone Number
3. Set the Data Type property for the following integration component fields to DTYPE_TEXT:
   - Main Fax Number
   - Main Phone Number

   If the Data Type property for these fields is not DTYPE_TEXT, then Siebel CRM Desktop cannot synchronize data for these fields.

4 Add a new integration component field to the CRMDesktopFulfillmentCenterIO integration object using values from the following table.

Property	Value
Name	JVD Fulfillment Center Id
Data Type	DTYPE_ID
External Sequence	239
External Name	JVD Fulfillment Center Id
External Data Type	DTYPE_ID
XML Sequence	239
XML Tag	JVDFulfillmentCenterId

5 Deploy your changes to the Siebel Runtime Repository.

Modifying the Metadata

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

In this topic you modify the metadata.

To modify the metadata

1 Open the siebel_meta_info.xml file and then add the following new fields to the Fulfillment Center type:

```
<field Name='Description' Label='Description' DataType='DTYPE_TEXT'
IsPartOfUserKey='no' IOElemName='Description' />

<field Name='Main Fax Number' Label='Main Fax Number' DataType='DTYPE_PHONE'
IsPartOfUserKey='no' IOElemName='MainFaxNumber' />

<field Name='Main Phone Number' Label='Main Phone Number' DataType='DTYPE_PHONE'
IsPartOfUserKey='no' IOElemName='MainPhoneNumber' />
```

For more information about adding fields to the metadata, see "Adding Fields to the Metadata to Support Static Picklists" on page 294.

2 Remove the JVD Fulfillment Center field definition that you added in Step 2 on page 311.

This fields represents data that Siebel CRM gets through a join. The Fulfillment Center object includes this data so it is not necessary to synchronize it from the opportunity object.

3 Add a field definition for the JVD Fulfillment Center Id field. You add the following code:

```
<field Name='JVD Fulfillment Center Id' Label='JVD Fulfillment Center Id'
DataType='DTYPE_ID' IsFilterable='no' IsRefObjId='yes' RefObjTypeId='Fulfillment
Center' RefExposedToUI='no' IOElemName='JVDFulfillmentCenterId'/>
```

where:

- **IsRefObjId** specifies that CRM Desktop uses this field to create a relation with another object in CRM Desktop.
- **RefObjTypeId** specifies the type of object that includes the relation. For example, Fulfillment Center.
- **RefExposedToUI** instructs CRM Desktop to display the related object in the client as a separate object.

Modifying the Basic Mapping and Connector Configuration

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

In this topic you modify the basic mapping and connector configuration.

To modify the basic mapping and connector configuration

1 Allow Siebel CRM Desktop to store values for the new fields you added in Step 1 on page 314. You add the following code to the Fulfillment Center object in the siebel_basic_mapping.xml file. This code maps telephone number fields to existing Outlook fields and creates a custom field for the Description field:

```
<field id="Main Phone Number">
  <reader>
    <mapi_std>
      <mapi_tag id="0x3A080000"></mapi_tag>
      <convertor>
        <string/>
      </convertor>
    </mapi_std>
  </reader>
  <writer>
    <Outlook_std>
    <Outlook_field id="BusinessTelephoneNumber"></Outlook_field>
    <convertor>
      <string/>
    </convertor>
    </Outlook_std>
  </writer>
</field>

<field id="Main Fax Number">
  <reader>
    <mapi_std>
      <mapi_tag id="0x3A240000"></mapi_tag>
```

```
        <convertor>
          <string/>
        </convertor>
      </mapi_std>
    </reader>
    <writer>
      <Outlook_std>
      <Outlook_field id="BusinessFaxNumber"></Outlook_field>
      <convertor>
        <string/>
      </convertor>
      </Outlook_std>
    </writer>
  </field>

  <field id="Description">
    <reader>
      <mapi_user>
        <user_field id="sbl Description" ol_field_type="1"></user_field>
        <convertor>
          <string/>
        </convertor>
      </mapi_user>
    </reader>
    <writer>
      <Outlook_user>
        <user_field id="sbl Description" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
      </Outlook_std>
    </writer>
  </field>
```

2 Locate the following type definition for the opportunity object:

```
<type id="Opportunity"
```

3 Add the following code to the object you located in Step 2:

```
<field id="JVD Fulfillment Center Id">
  <reader>
    <mapi_user>
      <user_field id="sbl JVD Fulfillment Center ID" ol_field_type="1"></
      user_field>
      <convertor>
        <binary_hexstring/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <multiwriter>
      <Outlook_user>
        <user_field id="sbl JVD Fulfillment Center ID" ol_field_type="1"></
```

```
          user_field>
          <convertor>
            <binary_hexstring/>
          </convertor>
        </Outlook_user>
        <link_fields>
          <field from="Name" to="JVD Fulfillment Center"></field>
        </link_fields>
      </multiwriter>
    </writer>
  </field>
```

The writer statement in this code includes a link field that allows CRM Desktop to get the value for the JVD Fulfillment Center field in the Opportunity object from the Name field of the Fulfillment Center object. This is similar to how Siebel CRM uses a pick map. If Siebel CRM gets values for multiple fields through a join, then you can add multiple fields in the link_fields section. These link_fields must exist in the basic mapping but you do not need to add them to the metadata.

4 Use an XML editor open the connector_configuration.xml file.

5 Locate the following definition for the Opportunity object:

```
<type id="Opportunity"
```

6 Add the following code to the definition you located in Step 5:

```
<link>JVD Fulfillment Center Id</link>
```

This code specifies the JVD Fulfillment Center Id field as a link on the Opportunity object. In the metadata you specify that the JVD Fulfillment Center Id field is related to another object in CRM Desktop. CRM Desktop uses that relation during synchronization.

Defining the View

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

In this topic you define the view.

To define the view

1 Add a view that Siebel CRM Desktop uses to display the list of values in the SalesBook dialog box.

For more information, see "Code That Creates the View Definition That the SalesBook Control Uses" on page 322.

2 Open the lookup_view_defs.xml file and then locate the following code. This code resides at the beginning of the file:

```
<array key="all_lookup_types">
```

You specify the objects that define the SalesBook dialogs in the lookup_view_defs.xml file. For more information, see "Customizing the SalesBook Control" on page 163.

3 Add the following lookup type to the array you located in Step 2:

```
<lookup_view_def key="lookup:fulfillment-centers">
  <display name="Fulfillment Centers"></display>
  <filter dasl="([http://schemas.microsoft.com/mapi/proptag/0x001A001E] &gt;=
'IPM.Contact.SBL.Fulfillment_Center' AND [http://schemas.microsoft.com/mapi/
proptag/0x001A001E] &lt;= 'IPM.Contact.SBL.Fulfillment_Center')"></filter>
  <view id="fulfillment_centers:salesbook"></view>
  <quick_lookup dasl_format="[http://schemas.microsoft.com/mapi/id/{00062004-
0000-0000-C000-000000000046}/8005001E] ='%s'"></quick_lookup>
  <type id=""></type>
</lookup_view_def>
```

This code does the following:

- Hard codes the label for the display name. In an actual implementation it is recommended that you add resource strings to the resource file and that you replace this hard coding with references to these resource strings.
- Specifies the object name in the filter.
- Uses the view Id to reference the view that you define in Step 2.
- Uses the type Id to specify the type of object that CRM Desktop creates if the user clicks the New button in the SalesBook dialog box. Leaving the New button empty disables it in the SalesBook dialog box.

4 Add the salesbook control to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<stack layout="horz" spacing="3">
  <cell>
    <autocomplete id="jvd_fulfillment_center_id">
    <field>JVD Fulfillment Center
    Id</field>
    <source type="Fulfillment Center"format=":[:(Name) :]">
    </source>
    </autocomplete>
  </cell>
    <cell size="21" attraction="far">
  <  button id="btn_fulfillment_center_select" image="lookup_button" >
      <text>...</text>
    </button>
  </cell>
</stack>
```

This code adds an autocomplete list and links the JVD Fulfillment Center Id field to this text field. This text field allows the user to enter characters for the name of a fulfillment center. CRM Desktop displays an autocomplete list while the user enters these characters. The Fulfillment Center object provides the values that CRM Desktop displays in the autocomplete list, and also specifies to display the Name field in the autocomplete list. This code also adds a button that opens the SalesBook dialog box. For more information, see "Registering Autocomplete Controls" on page 173.

Modifying the Business Logic and Testing Your Work

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

In this topic you modify the business logic and test your work.

To modify the business logic and test your work

1. Use a JavaScript editor to open the business_logic.js file.
2. Locate the following function:

   ```
   create_siebel_meta_scheme2
   ```

   This function sets up the relationships between objects.
3. Locate the following section in the create_siebel_meta_scheme2 function:

   ```
   //Direct Links
   ```

   For more information, see "Types of Links You Can Specify" on page 222.
4. Create the link definition for the Fulfillment Center object relation with the Opportunity object. You add the following code to the Direct Links section:

   ```
   function add_direct_link(from_type, to_type, link_field, required_link,
   view_ids, refresh_required, status, primary_on_parent)
   ```

   where:

   - *from_type* specifies the object where the link field resides.
   - *to_type* specifies the object that the link field references.
   - *link_field* specifies the name of the field for the link.
   - *required_link* specifies if the link is required.
   - *view_ids* specifies the Id of the view to use in the salesbook control.
   - *refresh_required* specifies to refresh the parent object. If you specify link_fields, then you must specify refresh_required.
   - *status* specifies the status field for the from type.
   - *primary_on_parent* is not used in this example.

   In this example, you add the following code:

   ```
   add_direct_link("Opportunity", "Fulfillment Center", "JVD Fulfillment Center Id",
   false, ["lookup:fulfillment-centers"], true, "OpportunityStatus");
   ```
5. Locate the following function:

   ```
   opportunity_form
   ```

   To enable the autocomplete list and salesbook button on the form, you add JavaScript as part of the opportunity_form function.
6. Locate the following section in the opportunity_form function:

```
//CONTROLS EVENTS
```

7 Add the following code to the section that you located in Step 6:

```
register_autocomplete_control(ctx, "Fulfillment
Center","jvd_fulfillment_center_id","btn_fulfillment_center_select");
```

This code does the following:

- Enables the autocomplete list.
- Uses the ctx object as input.
- Specifies to store the possible values in the Fulfillment Center type.
- Specifies jvd_fulfillment_center_id as the form Id for the field that CRM Desktop uses with the autocomplete list.
- Specifies btn_fulfillment_center_select as the Id of the button that the user clicks to display the SalesBook dialog box.

For more information, see "Registering Autocomplete Controls" on page 173.

8 Upload and publish your work. Do "Uploading and Testing Your Static Picklist" on page 298.

9 Test your work:

a Open the client and then navigate to the opportunity form.

b Make sure the form displays the Center picklist.

c Type text into the Center picklist.

As you enter each character of text the Center picklist must display different values in the auto complete entries. These entries must change according to the character you enter. For example, if you enter the letter B, then the field must display entries that begin with the letter B.

d Click the button that opens the SalesBook dialog box.

CRM Desktop must display the SalesBook dialog box. This dialog box must include a text entry field that allows the user to enter the name of the fulfillment center. It must also include a list of fulfillment centers. For example:

- Concord Fulfillment Center
- Copy Center
- Marketing Department
- And so on

Defining Multiple Linked Fields

This task is a step in "Process of Creating Dynamic Picklists That Use a SalesBook Control" on page 313.

You can configure Siebel CRM Desktop to display multiple linked fields in the client. For example, it can display the phone number of the fulfillment center next to the name of the fulfillment center on the Opportunity form.

To define multiple linked fields

1 Add the Fulfillment Center Phone field to the basic mapping. Do "Adding Fields to the Basic Mapping to Support Static Picklists" on page 295, but use the following code:

```
<field id="Fulfillment Center Phone">
  <reader>
    <mapi_user>
      <user_field id="sbl Fulfillment Center Phone" ol_field_type="1"></
user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Fulfillment Center Phone" ol_field_type="1"></
user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

Bold text identifies the code that is different for this example.

2 Locate the following code line in the code that you added in Step 3 on page 316:

```
<field from="Name" to="JVD Fulfillment Center"></field>
```

3 Add the following code immediately after the code line you located in Step 2:

```
<field from="Main Phone Number" to="JVD Fulfillment Center Phone"></field>
```

This code adds the link field for the phone number.

4 Add the phone number field to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<edit id="jvd_fulfillment_center_phone">
  <field value="string">JVD Fulfillment Center Phone</field>
</edit>
```

5 Use a JavaScript editor to open the forms.js file and then add the following code:

```
ctx.form.jvd_fulfillment_center_phone.enabled = false;
```

This code makes the phone number field read only.

6 Add the following code to the opportunity_form function in the forms.js file:

```
function on_fulfillment_center_changed()
{
ctx.form.jvd_fulfillment_center_phone.value =
ctx.form.item.snapshot['JVDFulfillment Center Phone'];
}
```

This function makes sure that CRM Desktop gets the value for the client from the data model. It makes sure the phone number field updates correctly in the client. If you do not include this code, and if the user chooses a fulfillment center in the SalesBook dialog box, then CRM Desktop updates the data correctly but it does not immediately update the value in the client for the Phone Number field.

7 Make sure CRM Desktop calls the on_fulfillment_center_changed function. You add the following event connector:

```
ctx.events.connect(ctx.form["jvd_fulfillment_center_id"],"changed",on_fulfillme
nt_center_changed);
```

For more information, see "Customizing Event Connectors" on page 168.

To include more link fields, you can add another line for each link field that you must add in the on_fulfillment_center_changed function.

Code That Creates the View Definition That the SalesBook Control Uses

To create the view definition that Siebel CRM Desktop uses to display the list of values in the SalesBook dialog box, you add the following code to the views.xml file. This code makes the Fulfillment Center available to the client. It hard codes the column headings. In an actual implementation it is recommended that you add resource strings to the resource file and that you replace this hard coding with references to these resource strings:

```
<str key="fulfillment_centers:salesbook">
<![CDATA[<?xml version="1.0"?>
<view type="table">
<viewname>$view_siebel_fulfillment_centers$</view name>
<viewstyle>tablelayout:fixed;width:100%;font-family:Segoe
UI;fontstyle:normal;font-weight:normal;fontsize:8pt;color:Black;font-
charset:0<viewstyle>
<viewtime>214490418</viewtime>
<linecolor>8421504</linecolor>
<linestyle>3</linestyle>
<gridlines>1</gridlines>
<newitemrow>0</newitemrow>
<usequickflags>0</usequickflags>
<collapsestate/>
<rowstyle>backgroundcolor:window;color:windowtext</rowstyle>
<headerstyle>backgroundcolor:#D3D3D3</headerstyle>
<previewstyle>color:Blue</previewstyle>
<arrangement>
<autogroup>0</autogroup>
<collapseclient/>
```

```
</arrangement>
<column>
<name>HREF</name>
<prop>DAV:href</prop>
<checkbox>1</checkbox>
</column>
<column>
<editable>0</editable>
<heading>$head_icon$</heading>
<prop>http://schemas.microsoft.com/mapi/proptag/0x0fff0102</prop>
<bitmap>1</bitmap>
<width>18</width>
<style>padding-left:3px;;textalign:center</style>
</column>
<column>
<maxrows>4294901760</maxrows>
<heading>Fulfillment Center</heading>
<prop>urn:schemas:contacts:sn</prop>
<type>string</type>
<width>987</width>
<style>padding-left:3px;;textalign:left</style>
<editable>1</editable>
<userheading>FulfillmentCenter</userheading>
</column>
<column>
<maxrows>4294901760</maxrows>
<heading>Description</heading>
<prop>http://schemas.microsoft.com/mapi/string/{00020329-0000-0000-C000-
000000000046}/sbl%20Description</prop>
<type>string</type>
<width>987</width>
<style>padding-left:3px;;textalign:left</style>
<editable>1</editable>
<userheading>Description</userheading>
</column>
<orderby>
<order>
<heading>FulfillmentCenter</heading>
<prop>urn:schemas:contacts:sn</prop>
<type>string</type>
<userheading>FulfillmentCenter</userheading>
<sort>asc</sort>
</order>
</orderby>
<multiline>
<width>0</width>
</multiline>
<groupbydefault>0</groupbydefault>
<previewpane>
<markasread>0</markasread>
</previewpane>

</view>]]>
</str>
```

Process of Creating Hierarchical Picklists

To create a hierarchical picklist, you do the following:

1 Modifying Siebel CRM Objects to Support Hierarchical Picklists on page 324

2 Modifying the Metadata to Support Hierarchical Picklists on page 326

3 Modifying the Basic Mapping and Forms to Support Hierarchical Picklists on page 328

4 Linking Fields and Testing Your Hierarchical Picklist on page 330

This topic describes how to configure Siebel CRM Desktop to display a hierarchical picklist. In this example, two fields use static picklists. The values displayed in the second picklist depend on the value that the user chooses in the first picklist. For more information about the hierarchical picklist, see *Configuring Siebel Business Applications*.

The example in this topic adds a hierarchical picklist to the Opportunity form in CRM Desktop. Assume that a hierarchical list of values named JVD_HIER already exists in Siebel CRM. Table 23 describes the properties of this hierarchical list of values.

Table 23. Properties of the JVD_HIER Hierarchical List of Values

Type	Display Value	Parent LIC
JVD_HIER	Child 1 of Parent 1	Parent 1
JVD_HIER	Child 1 of Parent 2	Parent 2
JVD_HIER	Child 2 of Parent 1	Parent 1
JVD_HIER	Child 2 of Parent 2	Parent 2
JVD_HIER	Parent 1	Not applicable
JVD_HIER	Parent 2	Not applicable

Modifying Siebel CRM Objects to Support Hierarchical Picklists

This task is a step in "Process of Creating Hierarchical Picklists" on page 324.

In this topic you modify Siebel CRM objects to support a hierarchical picklist.

To modify Siebel CRM objects to support a hierarchical picklist

1 Open Siebel Tools.

2 In the Object Explorer, click Pick List.

3 In the Picklists list, create two new picklists using values from the following table.

Name	Business Component	Search Specification
JVD Hierarchical - Child	PickList Generic	Not applicable
JVD Hierarchical - Parent	PickList Generic	'[Parent] Is Null'

4 In the Object Explorer, click Business Component.

5 In the Business Components list, query the Name property for Opportunity.

6 In the Object Explorer, expand the Business Component tree and then click Field.

7 In the Fields list, create two new fields using values from the following table.

Name	Join	Column	Type	Picklist
JVD Hier Child	S_OPTY_X	ATTRIB_47	DTYPE_TEXT	JVD Hierarchical - Child
JVD Hier Parent	S_OPTY_X	ATTRIB_46	DTYPE_TEXT	JVD Hierarchical - Parent

8 Extend the integration object to expose the new field:

a In the Object Explorer, click Integration Object.

b In the Integration Objects list, query the name property for CRMDesktopOpportunityIO.

c In the Object Explorer, expand the Integration Object tree and then click Integration Component.

d In the Integration Components list, query the External Name Context property for Opportunity.

e In the Object Explorer, expand the Integration Component tree and then click Integration Component Field.

f In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Hier Parent
Data Type	DTYPE_TEXT
Length	30
External Sequence	233
External Name	JVD Hier Parent
External Data Type	DTYPE_TEXT
XML Sequence	233
XML Tag	JVDHierParent

g In the Object Explorer, expand the Integration Component Field tree and then click Integration Component Field User Prop.

h In the Integration Component Field User Props list, add a new user property using values from the following table.

Property	Value
Name	PICKLIST
Value	Y

i In the Object Explorer, click Integration Component Field.

j In the Integration Component Fields list, add a new field using values from the following table.

Property	Value
Name	JVD Hier Child
Data Type	DTYPE_TEXT
Length	30
External Sequence	234
External Name	JVD Hier Child
External Data Type	DTYPE_TEXT
XML Sequence	234
XML Tag	JVDHierChild

k In the Object Explorer, click Integration Component Field User Prop.

l In the Integration Component Field User Props list, add a new user property using values from the following table.

Property	Value
Name	PICKLIST
Value	Y

9 Deploy your changes to the Siebel Runtime Repository.

Modifying the Metadata to Support Hierarchical Picklists

This task is a step in "Process of Creating Hierarchical Picklists" on page 324.

In this topic you modify the metadata to support a hierarchical picklist.

To modify the metadata to support a hierarchical picklist

1 Use an XML editor open the siebel_meta_info.xml file.

2 Locate the following object:

```
TypeId="Opportunity"
```

3 Add new fields to the Opportunity object. You add the following code immediately following the object you located in Step 2:

```
<field Name='JVD Hier Parent' Label='JVD Hier Parent'
DataType='DTYPE_TEXT' HasPicklist='yes'
PicklistIsStatic='yes'
PicklistCollectionType='JVD_HIER'
PicklistTypeId=List_Of_Values_Parent'
IOElemName='JVDHierParent'/>

<field Name='JVD Hier Child' Label='JVD Hier Child'
DataType='DTYPE_TEXT' HasPicklist='yes'
PicklistIsStatic='yes'
PicklistCollectionType='JVD_HIER'
PicklistTypeId='PickList_Hierarchical_Child'
IOElemName='JVDHierChild'/>
```

4 Locate the following object:

```
<picklist TypeId="PickList_Hierarchical"
. . .
</picklist>
```

5 Add the child picklist. Add the following code following the object you located in Step 4:

```
<picklist
  TypeId='PickList_Hierarchical_Child'
  SrcObjectTypeId='PickList_Hierarchical'
  CollectionTypeFldName='Type'
  ValueFldName='Value'
  LabelFldName='Value'
  LangFldName='Language' >
  <extra_src_fldname Visible='true'>
    Parent
  </extra_src_fldname>
  <master_filter_expr>
    <![CDATA[
      NOT [Parent Id] Is Null
    ]]>
  </master_filter_expr>
</picklist>
```

The following filter makes sure that Siebel CRM Desktop gets only the four values that are applicable as child values:

```
NOT [Parent Id] Is Null
```

The following field sets up filtering for items that you configure later in this procedure:

```
<extra_src_fldname Visible='true'>
  Parent
</extra_src_fldname>
```

6 Add the parent picklist. You add the following code:

```
<picklist
  TypeId='List_Of_Values_Parent'
  SrcObjectTypeId='List_Of_Values'
  CollectionTypeFldName='Type'
  ValueFldName='Value'
  LabelFldName='Value'
  LangFldName='Language'>
  <master_filter_expr>
  <![CDATA[
    [Active] = 'Y' AND [Parent Id] Is Null
     ]]>
  </master_filter_expr>
</picklist>
```

The following filter makes sure that Siebel CRM Desktop gets the values that are allowed as parent values:

```
[Active] = 'Y' AND [Parent Id] Is Null
```

Modifying the Basic Mapping and Forms to Support Hierarchical Picklists

This task is a step in "Process of Creating Hierarchical Picklists" on page 324.

In this topic you modify the basic mapping and forms to support a hierarchical picklist.

To modify the basic mapping and forms to support a hierarchical picklist

1 Add the parent field to the basic mapping. Do "Adding Fields to the Basic Mapping to Support Static Picklists" on page 295, but use the following code:

```
<field id="JVD Hier Parent">
  <reader>
    <mapi_user>
      <user_field id="sbl JVD Hier Parent" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Hier Parent" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

2 Add the child field to the basic mapping. Do "Adding Fields to the Basic Mapping to Support Static Picklists" on page 295, but use the following code:

```
<field id="JVD Hier Child">
  <reader>
    <mapi_user>
      <user_field id="sbl JVD Hier Child" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl JVD Hier Child" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

3 Create the type that stores the list of values for the parent picklist. Do "Modifying the Basic Mapping to Store Values for Static Picklists" on page 296, but use the following code:

```
<type id="OpportunityJVD Hier ParentPicklist"
predefined_folder="1" ver="1">
<form message_class="IPM.Contact.SBL.OpportunityJVD_Hier_ParentPicklist"></
form>
```

Bold text identifies the code that is different for a hierarchical picklist. For more information about doing this step, see Step 4 on page 297.

4 Create the type that stores the list of values for the child picklist. Do "Modifying the Basic Mapping to Store Values for Static Picklists" on page 296, but use the following code:

```
<type id="OpportunityJVD Hier ChildPicklist"
predefined_folder="1" ver="1">
<form message_class="IPM.Contact.SBL.OpportunityJVD_Hier_ChildPicklist"></form>
```

Bold text identifies the code that is different for a hierarchical picklist. For more information about doing this step, see Step 4 on page 297.

5 Allow the child picklist to store the value from the parent field. You add the following code to the OpportunityJVD Hier ChildPicklist type that you created in Step 4 on page 329:

```
<field id="Parent">
  <reader>
    <mapi_user>
      <user_field id="sbl Parent" ol_field_type="1"></user_field>|
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
```

```
      <Outlook_user>
        <user_field id="sbl Parent" ol_field_type="1"></user_field>
        <convertor>
          <string/>
        </convertor>
      </Outlook_user>
    </writer>
  </field>
```

6 Add the parent field to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<combobox id="jvd_hier_parent">
  <field>JVD Hier Parent</field>
  <source type="OpportunityJVD Hier ParentPicklist" field="Value"
    format=":[:(Label):]"></source>
</combobox>
```

Bold text identifies the code that is different for a hierarchical picklist. For more information, see "Combobox Control of the Forms File" on page 477.

7 Add the child field to the Opportunity form. Do "Modifying the Form to Support Static Picklists" on page 297, but use the following code:

```
<combobox id="jvd_hier_child">
  <field>JVD Hier Child</field>
  <source type="OpportunityJVD Hier ChildPicklist" field="Value"
    format=":[:(Label):]"></source>
</combobox>
```

Bold text identifies the code that is different for a hierarchical picklist.

Linking Fields and Testing Your Hierarchical Picklist

This task is a step in "Process of Creating Hierarchical Picklists" on page 324.

If the user chooses a value in the parent picklist, then this value filters the values that CRM Desktop displays in the child picklist. To set up this relationship in:

- **Siebel Tools**. You set up a constraint on the pickmap.
- **Siebel CRM Desktop.** You write a JavaScript function and then add it to the opportunity_form function in the forms.js file.

This opportunity_form function implements the business logic for the Opportunity form. The forms.js file includes a similar function for each form that Siebel CRM Desktop displays.

If you do not establish this relationship between the parent picklist and the child picklist, then the parent dropdown list displays Parent 1 and Parent 2 as options and the child dropdown list displays all four child values.

To link fields and test your hierarchical picklist

1 Use a JavaScript editor to open the forms.js file.

2 Locate the following section in the opportunity_form function:

```
//FORM FUNCTIONS
```

3 Add the following function to the section you located in Step 2:

```
function on_parent_changed(initial)
{
  var parent_val = initial === true?
  ctx.item_ex.get_property("JVD Hier Parent"):
  ctx.form.jvd_hier_parent.value;
  ctx.form.jvd_hier_child.items.filter =
   ctx.session.create_expression("Parent","eq", parent_val == null ||
   parent_val == ''? '' : parent_val);
}
```

Siebel CRM Desktop uses this function in the following ways:

- **If the user opens the form.** CRM Desktop sets the search specification on the child dropdown list. The function gets the value for the parent field from the client.
- **If the user changes the value in the parent dropdown list.** The function gets the value for the parent field from the object model.

4 Add the following code immediately after the function you added in Step 3:

```
on_parent_changed(true);
```

If the user opens the form, then this code allows the function to get the value for the parent field from the client.

5 Locate the following section in the opportunity_form function:

```
//CONTROL EVENTS
```

6 Add the following event connector to the section you located in Step 5:

```
ctx.events.connect(ctx.form["jvd_hier_parent"], "changed", on_parent_changed);
```

If the user changes the value in the parent dropdown list, then this event handler calls the function to get the value for the parent field from the object model. For more information, see "Customizing Event Connectors" on page 168.

7 Upload and publish your work. Do "Uploading and Testing Your Static Picklist" on page 298.

8 Test your work:

a Open the client and then navigate to the opportunity form.

b Make sure the form displays the following picklists:

- Hier Parent
- Hier Child

c Choose the Hier Parent picklist and make sure it includes the following values:

- ❑ - None -
- ❑ Parent 1
- ❑ Parent 2

d Choose Parent 1 in the Hier Parent picklist.

CRM Desktop should automatically change the value for the Hier Child picklist to Child 1 of Parent 1.

e Choose the Hier Child picklist and make sure it includes the following values:

- ❑ - None -
- ❑ Child 1 of Parent 1
- ❑ Child 2 of Parent 1

f Choose Parent 2 in the Hier Parent picklist.

CRM Desktop should automatically change the value for the Hier Child picklist to Child 1 of Parent 2.

g Choose the Hier Child picklist and make sure it includes the following values:

- ❑ - None -
- ❑ Child 1 of Parent 2
- ❑ Child 2 of Parent 2

Configuring Unbounded Picklists

A *bounded picklist* is a type of list that allows the user to choose a value from the values that this list displays. An *unbounded picklist* is a type of list that provides the same functionality as a bounded picklist, but it also allows the user to enter a value in the field associated with this list. if the user enters a new value in an unbounded picklist, then Siebel CRM Desktop can do one of the following:

- Save this new value in the field but not add it to the values that the picklist displays.
- Save this new value in the field and add it to the values that the picklist displays.

Configuring an Unbounded Picklist That Adds New Values to Fields

This topic describes how to configure an unbounded picklist so that the user can enter a new value in the field associated with this picklist. It does not describe how to add this value to the values that this picklist displays.

To configure an unbounded picklist that adds new values to fields

1. Use an XML editor to open the form_xx.xml file.
2. Locate the combobox control that the picklist you must modify references.

   For more information, see "Combobox Control of the Forms File" on page 477.

3 Modify the combobox tag so that it allows the user to enter characters in the field associated with the picklist. Use the following code:

```
<combobox id="comboxbox_name" tab_order="8" style="editable">
```

where:

- *comboxbox_name* is the name of the combobox that you must modify.
- editable allows the user to enter values.

For example, the following code allows the user to enter characters in the predefined sales_method combobox:

```
<combobox id="sales_method" tab_order="8" style="editable">
```

Configuring an Unbounded Picklist That Adds New Values to Fields and to the Picklist

This topic describes how to configure an unbounded picklist so that the user can enter a new value in the field associated with this picklist. It also describe how to add the value that the user enters to the values that this picklist displays. The example in this topic modifies the picklist that the user uses to set the status of an opportunity.

Configuring an unbounded picklist that adds new values to fields and to the picklist

1 Replace the combobox control:

a Use an XML editor to open the form_xx.xml file.

b Locate the following code:

```
<combobox id="status" tab_order="14">
  <items format=":[:(Label):]" value_column="Value" has_null_item="true">
    <source type="auto" name="OpportunityStatusPicklist"></source>
    <order_by>
      <order ascend="true">SortOrder</order>
    </order_by>
  </items>
  <field>Status</field>
</combobox>
```

This code specifies the combobox control that the user uses to set the opportunity status. For more information, see "Combobox Control of the Forms File" on page 477.

c Replace the code you located in Step b on page 333 with the following code:

```
<cell>
  <edit id="status" tab_order="14">
    <field value="string">Status</field>
  </edit>
</cell>
  <cell size="21">
  <button id="btn_opportunity_status" tab_order="141">
```

```
    <text>...</text>
  </button>
</cell>
```

This step replaces the combobox control code with an edit control that the user can use to enter characters, and a button that the user can click to display the SalesBook control that contains an autocomplete list.

d Save your modifications.

2 Specify options for the SalesBook control in the business_logic.js file:

a Use a JavaScript editor to open the business_logic.js file.

b Add the following code:

```
scheme.objects.get_object("OpportunityStatusPicklist").selectors_options = {
"source": {
"caption": "obj_opportunity_status_plural",
"view_id": "opportunity_status:salesbook",
"search_by": ["Label"],
"online": {
"like_template": "*{keyword}*"
}
}
};
```

This code specifies options for the SalesBook control, such as the caption and view name that Siebel CRM Desktop displays with the SalesBook control.

c Save, and then close the business_logic.js file.

3 Specify options for the SalesBook control in the views.xml file:

a Use an XML editor to open the views.xml file.

b Add the following code:

```
<view id ="opportunity_status:salesbook">
  <image_list>
    <res_id type="normal">type_image:Generic:16</res_id>
  </image_list>
  <columns>
    <column width="100">
      <heading type="string">head_name</heading>
      <field>Label</field>
    </column>
  </columns>
</view>
```

where:

- `opportunity_status` is the view you specified in Step b on page 334.
- `head_name` is a string key that specifies the heading that Siebel CRM Desktop displays in the SalesBook control. You must make sure that the package_res.xml file specifies this `head_name`.

c Save, and then close the views.xml file.

d Use an XML editor to open the package_res.xml file.

e Make sure the following code exists:

```
<str key="head_name">
...
</str>
```

If this code does not exist, then add it now. If necessary, you can modify the head_name string key.

4 Add a controller and a link:

a Use a JavaScript editor to open the forms.js file.

b Locate the form handler you must modify.

You can add a controller and a link to the handler of any object. In this example you modify the following handler for the Opportunity object:

```
function opportunity_form(ctx)
{
...
}
```

c Locate the opportunity_form function that resides in the form handler that you located in Step b on page 335.

d Add the following code to the opportunity_form function:

```
function my_link()
  {
  this.type = ctx.item_ex.get_type();
  this.link_to = "OpportunityStatusPicklist";
  this.tag = "local_opportunity_status_link";
  this.create = function(ctx, spec_ctx)
  {
    var opportunity_status = ctx.session.open_item(spec_ctx.with_id);
    ctx.form.item["Status"] = opportunity_status.Value;
    ctx.form.status.value = opportunity_status.Value;
    return ({});
  }
  }
```

This code adds the controller that CRM Desktop uses to control the button that you add in Step c on page 333. CRM Desktop uses this controller to call the SalesBook dialog box.

e Add the following code immediately after the code you added in Step d on page 335:

```
ctx.links.add_link(new my_link());
var opportunity_status_options =
{
  "tag": "local_opportunity_status_link",
  "link_to": "OpportunityStatusPicklist",
  "sb_custom_view": true,
```

```
  "btn_show": ctx.form.btn_opportunity_status
}
ctx.form_links_manager.add_controller(new mvg_dialogs.custom_sales_book(),
opportunity_status_options);
```

This code specifies the link that CRM Desktop uses to add a value to the field.

f Save, and then close the forms.js file.

5 Add a new dialog box that allows the user to enter values:

a Use an XML editor to open the dialogs.xml file.

b Add the following code between any <dialog></dialog> tags:

```
<dialog id="Opportunity Status Dlg">
  <script><![CDATA[
  ]]></script>
  <layout sizable="true" id="General" min_height="120" min_width="400"
  caption="Opportunity Status"
  small_icon="type_image:Generic:16">
  <appearance height="120" width="400" position="parent_center"/>
  <cell>
    <stack layout="horz" padding="5">
  <cell>
    <stack layout="vert" padding="10" spacing="10">
  <cell>
    <stack layout="horz" spacing="3" padding="5">
    <!-- left side captions -->
      <cell size="128">
      <stack spacing="5" layout="vert" padding="4">
      <cell size="21">
      <static id="lbl_label" tab_order="1">
        <text>Label</text>
      </static>
     </cell>
      <cell size="21">
        <static id="lbl_value" tab_order="2">
          <text>Value</text>
        </static>
      </cell>
    </stack>
  </cell>
    <!-- left side fields -->
    <cell>
      <stack layout="vert" spacing="5">
    <cell>
      <edit id="label" tab_order="3">
        <field value="string">Label</field>
      </edit>
    </cell>
  <cell>
  <edit id="value" tab_order="4">
    <field value="string">Value</field>
  </edit>
  </cell>
```

```
    </stack>
      </cell>
        </stack>
          </cell>
            <cell size="2">
              <edge id="close_border"/>
            </cell>
              <cell size="24">
            <stack layout="horz" spacing="5" padding="5">
              <cell size="80">
              <button id="btn_ok" tab_order="5" image="button_image:Ok:16"
align="left">
              <text>#btn_ok</text>
              </button>
            </cell>
              <cell size="80">
                <button id="btn_cancel" tab_order="6"
image="button_image:Cancel:16"
                  align="left">
                  <text>#btn_cancel</text>
              </button>
             </cell>
          </stack>
        </cell>
      </stack>
    </cell>
    </stack>
    </cell>
  </layout>
</dialog>
```

c Save, and then close the dialogs.xml file.

6 Add the trigger that CRM Desktop uses to call the dialog box:

a Use a JavaScript editor to open the business_logic.js file.

b Locate the following code:

```
with (scheme.triggers)
{
...
}
```

c Add the following code to the code you located in Step b:

```
add_simple_trigger(form_helpers.create_dialog_show("Opportunity Status Dlg",
{ "form_handler": null }), null,
"OpportunityStatusPicklist", null, "show");
```

This code specifies to call the dialog box that you added in Step 5 if the user clicks New in the SalesBook dialog box.

d Save, and then close the business_logic.js file.

7 Test your work:

a Log into the client.

b Navigate to the Opportunity form.

c Enter a value in the Status field, and then step off the field.

d Click the Status field, and then make sure CRM Desktop displays the value you added in the picklist.

Configuring Lists of Values to Support Multiple Languages

The predefined Siebel CRM Desktop runs in the language that Outlook uses when you install CRM Desktop. This configuration works if Microsoft Outlook and the Siebel object manager use the same language. For example, you can run an English Outlook client that connects to an English object manager, or a German Outlook client that connects to a German object manager. This configuration does not work in a Siebel deployment that uses multiple languages. For example, you cannot use the predefined configuration to run an English object manager with English Outlook clients and German Outlook clients. This topic describes how to configure CRM Desktop so that you can run an English object manager with multiple languages.

To configure lists of values to support multiple languages

1 Open Siebel Tools.

2 Create the new business components that will handle the multilingual values:

a In the Object Explorer, click Business Component.

b In the Business Components list, locate the List Of Values business component.

c Click Edit and then click Copy Record.

d Set properties for the copy you made in Step c using values from the following table.

Property	Value
Name	CRM Desktop List Of Values
Search Specification	([Class Code] <> 'CLASS' OR [Class Code] IS NULL) AND [Language]= Language() This search specification makes sure this business component only returns values where the language of these values match the language that the object manager uses.

e In the Business Components list, locate the PickList Hierarchical business component.

f Click Edit and then click Copy Record.

g Set properties for the copy you made in Step f using values from the following table.

Property	Value
Name	CRM Desktop PickList Hierarchical
Search Specification	[Language] = Language()

3 Create business objects for the business components that you created in Step 2:

a In the Object Explorer, click Business Object.

b Right-click in the Business Objects list, click New Record, and then set properties using values from the following table.

Property	Value
Name	CRM Desktop List Of Values

c In the Object Explorer, expand the Business Object tree and then click Business Object Component.

d Right-click in the Business Object Components list, click New Record, and then set properties using values from the following table.

Property	Value
Name	CRM Desktop List Of Values

e Right-click in the Business Objects list, click New Record, and then set properties using values from the following table.

Property	Value
Name	CRM Desktop PickList Hierarchical

f Right-click in the Business Object Components list, click New Record, and then set properties using values from the following table.

Property	Value
Name	CRM Desktop PickList Hierarchical

4 Update the integration objects:

a In the Object Explorer, click Integration Object.

b In the Integration Objects list, locate the CRMDesktopListOfValuesIO integration object and then modify the properties of this integration object using value from the following table.

Property	Value
Name	CRM Desktop List Of Values

c In the Object Explorer, expand the Integration Object tree and then click Integration Component.

d In the Integration Components list, query the Name property for the following value:

```
List Of Values
```

e Modify the properties of the integration component you located in Step d using value from the following table.

Property	Value
Name	CRM Desktop List Of Values
External Name Context	CRM Desktop List Of Values
External Name	CRM Desktop List Of Values

f In the Integration Objects list, locate the CRMDesktopPickListHierarchicalIO integration object and then modify the properties of this integration object using value from the following table.

Property	Value
External Name	CRM Desktop PickList Hierarchical

g In the Integration Components list, query the Name property for the following value:

```
PickList Hierarchical
```

h Modify the properties of the integration component you located in Step g using value from the following table.

Property	Value
Name	CRM Desktop PickList Hierarchical
External Name Context	CRM Desktop PickList Hierarchical
External Name	CRM Desktop PickList Hierarchical

5 Deploy your changes to the Siebel Runtime Repository.

6 Update the CRM Desktop package so that it does not get the list of values in a certain language but instead accepts the language that the Siebel object manager returns:

a Use an XML editor open the siebel_meta_info.xml file.

b Remove the LangFldName='Language' attribute from the following PickList objects:

- StatePickList
- PickList_Hierarchical
- List_Of_Values
- AccountNoteType
- RevenueType

11 Customizing Multi-Value Groups

This chapter describes how to customize multi-value groups. It includes the following topics:

Overview of Customizing Multi-Value Groups

Siebel CRM Desktop uses an MVG field to display associations between objects and to allow the user to choose the primary association.

Figure 17 displays an example of a field that uses the mvg_primary_selector2 control and the MVG button. CRM Desktop uses an ellipsis (. . .) as the label for the MVG button that opens the MVG dialog box. It displays this field with one primary association, by default. If the user clicks the ellipsis, then it displays all associations in the mvg_dialog dialog box. The user must use this dialog box to add a new association. The user cannot add a new association directly in the field.

Figure 17. Primary Selector in the MVG Dialog Box

Figure 18 displays an example of an *inline MVG*, which is a control that displays multiple associations in one field, where a semicolon separates each association, by default. It uses bold font to indicate the primary association. In this example, Default Organization is the primary association. The user can right-click an association to set it as the primary or to delete it. An inline MVG uses the autocomplete_list control and the MVG button. Siebel CRM Desktop uses the same ellipsis (. . .) that it uses in Figure 17 as the label for the MVG button that opens the same mvg_dialog dialog box. CRM Desktop supports the inline MVG starting with Siebel CRM Desktop version 3.5.

Figure 18. Example of an Inline MVG

Figure 19 displays an example of an autocomplete list where the user has started to enter a value in the field. The user can enter characters directly in the field to add a new association. As the user enters each character, CRM Desktop displays associations in the autocomplete list. The user can then click one of these associations to add it. For more information about the code that renders an autocomplete list, see "Code That Renders an Autocomplete List" on page 366.

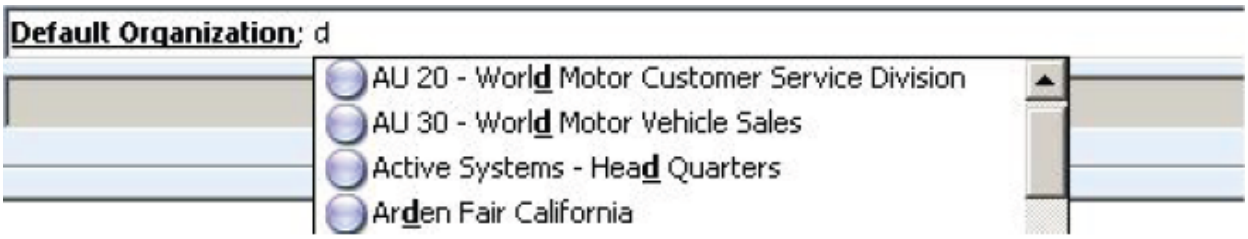


Figure 19. Entering a New Association in an Autocomplete List

Process of Creating MVG Fields

This topic describes how to create an MVG (multi-value group) field. You do the following work to add an MVG field:

1. Identifying Predefined MVG Objects in Siebel CRM on page 344
2. Process of Making Siebel CRM Data Available to Add an MVG on page 346
3. Process of Modifying the Customization Package to Add an MVG on page 351
4. Publishing and Testing a Custom MVG Field on page 356

An MVG field displays an association with other objects. This association can be a many to many association, or a many to one association. The user can use an MVG field to do the following:

- Add or remove an association
- Change a primary association
- Browse existing associations

You can also use an MVG to economize the layout of a form. You can display a set of associated records in a single-line control instead of using the Outlook_view control.

The example in this topic adds an MVG control to the Opportunity form. This MVG displays an association between an opportunity and channel partners.

For more information about MVGs, see *Configuring Siebel Business Applications*.

Identifying Predefined MVG Objects in Siebel CRM

This task is a step in "Process of Creating MVG Fields" on page 344.

In this topic, you identify the MVG objects that you use to add an MVG field for this example. These objects come predefined with Siebel CRM.

To identify predefined MVG objects in Siebel CRM

1 Identify the field that Siebel CRM associates with the MVG you must add:

a Open Siebel Call Center.

b Navigate to the Opportunities list and then click the link in the Opportunity Name field of an opportunity.

c Click the More Info tab.

d Locate the More Info field.

e Choose the Help menu and then click About View.

The About View dialog box lists the applets in the order in that Siebel Call Center displays them.

f Note the applet name that Siebel Call Center uses to display the More Info field.

In this example, this is the Contact Form Applet - Child applet.

g In Siebel Tools, in the Object Explorer, click Applet.

h In the Applets list, query the Name property for Opportunity Form Applet - Child Big.

i Right-click the Opportunity Form Applet - Child Big applet and then choose Edit Web Layout.

If Siebel Tools displays the Read-only Object dialog box, then you must check out the project. For more information, see "Checking Out Projects in Siebel Tools" on page 165.

j In the Applet Web Template editor, click the More Info control.

k In the Properties window, note the values for the following properties.

Property	Value
Field	Mail Stop
MVG Applet	Partner Lead Name Mvg Applet

In this example, Siebel CRM associates the Partner field with the Lead Partner MVG.

2 Identify the MVG link.

a In the Object Explorer, click Business Component.

b In the Business Components list, query the Name property for Opportunity.

c In the Object Explorer, expand the Business Component tree and then click Field.

d In the Fields list, query the Name property for Partner and then note the values for the following properties.

Property	Value
Multi Valued	TRUE
Multi Valued Link	Channel Partner

e In the Object Explorer, click Multi Value Link.

f In the Multi Value Links list, query the Name property for the Multi Valued Link that the field references that you noted in Step d. In this example, this link is Channel Partner.

g Note the values for the following properties.

Property	Value
Destination Link	Opportunity/Channel Partner
Primary Id Field	Primary Partner Id

h In the Object Explorer, click Link.

i In the Links list, query the Name property for the Destination Link that you noted in Step g.

In this example, this link is Opportunity/Channel Partner. Note the values for the properties described in the following table.

Property	Value
Inter Table	S_OPTY_ORG If the Inter Table property: ■ Contains an intersection table, then the link maintains a many to many association. ■ Is empty, then the link maintains a one to many association.
Child Business Component	Channel Partner The business component in the Child Business Component property must be displayed.

To identify the required business component, you can also examine the Business Component property of the Partner Lead Name MVG Applet.

For more information, see "Types of Links You Can Specify" on page 222.

Process of Making Siebel CRM Data Available to Add an MVG

This task is a step in "Process of Creating MVG Fields" on page 344.

To make Siebel CRM data available to add an MVG, you do the following:

1 Creating an Integration Object for the Channel Partner MVG on page 347

2 Creating an Integration Component for the Channel Partner MVG on page 348

3 Extending an Integration Object for the Primary Id Field on page 350

This topic describes how to make sure Siebel CRM data available to Siebel CRM Desktop. Some Siebel CRM data is available without customizing CRM Desktop, such as opportunities, accounts, and contacts. Other Siebel CRM data is not available. For example, if your implementation requires Channel Partner data, then you must configure integration objects to make this data available to CRM Desktop.

Creating an Integration Object for the Channel Partner MVG

This task is a step in "Process of Making Siebel CRM Data Available to Add an MVG" on page 346.

In this topic, you create a new integration object that makes the channel partner data available to the EAI Siebel Adapter. The work you do in this topic allows the PIM Client Sync Service business service to access channel partner data. For more information, see "Siebel Enterprise Components That Siebel CRM Desktop Uses" on page 27.

To create an integration object for the channel partner MVG

1. In Siebel Tools, choose the File Menu and then click New Object.
2. Click the EAI tab, click Integration Object and then click OK.
3. In the Integration Object Builder Dialog box, choose values for the following items and then click Next.

Property	Value
Project	Choose a project. It is recommended that you create a separate project for any customizations you make to Siebel CRM Desktop. For example, use a project named Siebel CRM Desktop.
Business Service	EAI Siebel Wizard

4. Choose values for the following items and then click Next.

Property	Value
Source Object	Channel Partner
Source Root	Channel Partner
Integration Object Name	CRMDesktopChannelPartnerIO

5. Expand the Channel Partner tree, choose the integration components you must include with this integration object, and then click Next.

   As the default, Siebel Tools includes a check mark for each integration component. For this example, accept the default.

6. Click Next and then click Finish.
7. Examine the properties of the integration object you created in Step 6:

a In the Object Explorer, click Integration Object, query the Name property for CRMDesktopChannelPartnerIO, and then note the following property.

Property	Value
External Name	Channel Partner You will use this property as a value for the IntObjName attribute.

b In the Object Explorer, expand the Integration Object tree and then click Integration Object Component.

c In the Integration Object Components list, query the Name property for Channel Partner and then note the following properties.

Property	Value
XML Tag	ChannelPartner You will use this property as a value for the SiebMsgXmlElemName attribute.
XML Container Element	ListOfChannelPartner You will use this property as a value for the SiebMsgXmlCollectionElemName attribute.

Creating an Integration Component for the Channel Partner MVG

This task is a step in "Process of Making Siebel CRM Data Available to Add an MVG" on page 346.

In this topic, you add a channel partner integration component as a child of the integration object that Siebel CRM Desktop uses for opportunities. This allows CRM Desktop to query the channel partners that are associated with the opportunities for a user.

To create an integration component for the channel partner MVG

1 In Siebel Tools, display the object type named Integration Object.

For more information, see "Displaying Object Types in Siebel Tools" on page 166.

2 In the Object Explorer, click Integration Object.

3 In the Integration Objects list, query the Name property for CRMDesktopOpportunityIO and then make sure the Object Locked property contains a check mark.

CRM Desktop adds the CRMDesktopOpportunityIO integration object to the Siebel Runtime Repository when you install CRM Desktop on the Siebel Server. You must install it before you can complete this task.

4 In the Object Explorer, expand the Integration Object tree and then click Integration Component.

5 In the Integration Components list, add a new record with the following values.

Property	Value
Business Component	Channel Partner
Name	Opportunity_ChannelPartner
External Name	Channel Partner
External Sequence	2 For more information, see "Requirements for the Sequence Property" on page 281.
XML Sequence	10002 For more information, see "Requirements for the Sequence Property" on page 281.
XML Container Element	ListOfOpportunity_ChannelPartner
XML Tag	Opportunity_ChannelPartner

6 In the Object Explorer, expand the Integration Component tree and then click Integration Component Field.

7 In the Integration Component Fields list, add new records with the following values.

Name	Data Type	Length
IsPrimaryMVG	DTYPE_TEXT	1
Location	DTYPE_TEXT	50
Organization BU Name	DTYPE_TEXT	50
Partner	DTYPE_TEXT	100
Partner Id	DTYPE_Id	30
Partner Status	DTYPE_TEXT	30
operation	DTYPE_TEXT	30
searchspec	DTYPE_TEXT	250

8 In the Object Explorer, click Integration Component Key.

9 In the Integration Component Keys list, add new records with the following values.

Name	Key Sequence Number	Key Type
Modification Key	1	Modification Key
Primary Key	1	User Key
Status Key	1	Status Key

10 In the Object Explorer, click Integration Component User Prop.

11 In the Integration Component User Props list, add new records with the following values.

Name	Value
MVGAssociation	Y
MVGLink	Channel Partner

12 Compile your changes.

For more information, see *Using Siebel Tools*.

13 In the Object Explorer, click Integration Component and then note the following properties of the integration component that you added in Step 5. You use these values when you modify the metadata for the customization package.

Property	Value
Parent Name	CRMDesktopOpportunityIO You will use this property as a value for the IntObjName attribute.
XML Tag	Opportunity_ChannelPartner You will use this property as a value for the SiebMsgXmlElemName attribute.
XML Container Element	ListOfOpportunity_ChannelPartner You will use this property as a value for the SiebMsgXmlCollectionElemName attribute.

Extending an Integration Object for the Primary Id Field

This task is a step in "Process of Making Siebel CRM Data Available to Add an MVG" on page 346.

In this topic, you make the Primary Id field available to Siebel CRM Desktop. You make the primary on the opportunity available so that CRM Desktop can identify the record to display in the opportunity form if the opportunity includes more than one channel partner.

In this example, the Primary Id Field property of the MVG link contains a value. If this property were empty, then you would skip this topic and proceed to Step 1 on page 351. For more information, see "Types of Links You Can Specify" on page 222.

To extend an integration object for the Primary Id field

1 In the Object Explorer, click Integration Object.

2 In the Integration Objects list, query the Name property for CRMDesktopChannelPartnerIO, and then make sure the Object Locked property contains a check mark.

You created the CRMDesktopChannelPartnerIO integration object in Step 6 on page 347.

3 In the Object Explorer, expand the Integration Object tree and then click Integration Component.

4 In the Integration Components list, query the Name property for Opportunity.

5 In the Object Explorer, expand the Integration Components tree and then click Integration Component Field.

6 In the Integration Component Fields list, add a new record with the following values.

Property	Value
Name	Primary Partner Id
External Name	Primary Partner Id
Length	15
Data Type	DTYPE_ID
External Data Type	DTYPE_ID
External Sequence	139 For more information, see "Requirements for the Sequence Property" on page 281.
XML Sequence	139 For more information, see "Requirements for the Sequence Property" on page 281.
XML Tag	PrimaryPartnerId

7 Compile your changes.

For more information, see *Using Siebel Tools*.

Process of Modifying the Customization Package to Add an MVG

This task is a step in "Process of Creating MVG Fields" on page 344.

To modify the customization package to add an MVG, you do the following:

1 Adding a Custom Object on page 352

2 Adding the MVG Link on page 352

3 Adding the Primary Field on page 352

4 Adding a Field on page 353

5 Adding a Lookup View on page 354

6 Adding a Label, a Button, and a Selector Control on page 354

7 Customizing the Validation Message and Labels on page 356

Adding a Custom Object

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

To add a custom object to Siebel CRM Desktop, you display the object in Siebel CRM and then modify customization package XML files. The example in this topic makes available and then adds the Channel Partner object.

To add a custom object

1. Make sure the object you must add is available.

   For more information, see "Creating an Integration Object for the Channel Partner MVG" on page 347.

2. Add a custom object type. You modify the siebel_meta_info.xml file.

   For more information, see "Code That Adds a Custom Object Type" on page 357.

3. Map objects. You modify the siebel_basic_mapping.xml file.

   For more information, see "Code That Maps a Custom Object" on page 358.

4. Configure synchronization for the custom object. You modify the connector_configuration.xml file.

For more information, see "Code That Configures Synchronization for a Custom Object" on page 359.

Adding the MVG Link

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

In this topic, you add the MVG link to the business logic file. For more information, see "Types of Links You Can Specify" on page 222.

To add the MVG link

1. Use a JavaScript editor to open the business_logic.js file.

   For more information, see "Files That the Customization Package Contains" on page 434

2. Locate the following function:

   ```
   create_siebel_meta_scheme2
   ```

3. Add the code to add a new association.

   For more information, see "Code That Adds a New Association" on page 361.

Adding the Primary Field

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

In this topic, you add the primary field to the customization package. This field displays in the MVG dialog box that you add for this example.

To add the primary field

1 Add the primary field. You do the following:

a Use an XML editor open the siebel_meta_info.xml file.

b Add the following code to the Opportunity object:

```
<field Name='Primary Partner Id' Label='Primary Partner Id'
DataType='DTYPE_ID' IsFilterable='no' IsRefObjId='yes'
RefObjTypeId='ChannelPartner' IOElemName='PrimaryPartnerId' />
```

c Save and then close the siebel_meta_info.xml file.

d Use an XML editor to open the siebel_basic_mapping.xml file:

e Add the following code to the Opportunity type tag:

```
<field id="Primary Partner Id">
  <reader>
    <mapi_user>
      <user_field id="sbl Primary Partner Id" ol_field_type="1"></user_field>
      <convertor><binary_hexstring/></convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Primary Partner Id" ol_field_type="1"></user_field>
      <convertor><binary_hexstring/></convertor>
    </Outlook_user>
  </writer>
</field>
```

f Save and then close the siebel_basic_mapping.xml file.

2 Add the link:

a Use an XML editor open the connector_configuration.xml file.

b Add the following code to the Opportunity type tag:

```
<link>Primary Partner Id</link>
```

c Save and then close the connector_configuration.xml file.

Adding a Field

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

This topic describes how to add the ChannelPartnerStatus field.

To add a field

1 Use an XML editor to open the siebel_basic_mapping.xml file.

2 Add the following code to the Opportunity.Channel_Partner.Association type:

```
<field id="ChannelPartnerStatus">
  <reader>
    <mapi_user>
      <user_field id="sbl ChannelPartnerStatus" ol_field_type="1"></user_field>
      <convertor><string/></convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl ChannelPartnerStatus" ol_field_type="1"></user_field>
      <convertor><string/></convertor>
    </Outlook_user>
  </writer>
</field>
```

3 Save and then close the siebel_basic_mapping.xml file.

Adding a Lookup View

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

To customize a salesbook control, you use the definition of a lookup view. The SalesBook dialog box displays a list of objects and then allows the user to choose any object from this list. Siebel CRM Desktop uses this control on forms and MVG dialogs. The example in this topic adds the definition for a lookup view for Channel Partner objects.

To add a lookup view

1 Use an XML editor to open the lookup_view_defs.xml file.

For more information, see "Customizing the SalesBook Control" on page 163.

2 Add the following code to the array tag that contains the lookup types in the file:

```
<item value="ChannelPartner"></item>
```

3 Add the definition for the lookup view.

For more information, see "Code That Adds a Lookup View" on page 359.

4 Save and then close the lookup_view_defs.xml file.

Adding a Label, a Button, and a Selector Control

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

In this topic, you add a label, a button, and a selector control.

To add a label, a button, and a selector control

1 Use an XML editor to open the forms_xx.xml file.

2 Add a label for the MVG:

a Locate the following section:

```
right side captions
```

b Insert the following code immediately under the code that defines the Probability label:

```
<cell size="22">
  <static id="0x20014" tab_order="166">
    <text>#lbl_channel_partner</text>
  </static>
</cell>
```

3 Add the button and primary selector control:

a Locate the following section:

```
right side fields
```

b Add the following code immediately under the code that defines the Probability control:

```
<cell size="22">
  <stack layout="horz">
    <cell>
      <mvg_primary_selector id="channel_partner_mvg">
        <source type="Opportunity.Channel_Partner.Association"
left_id="OpportunityId" item_value="ChannelPartnerId"
display_format=":[:(PartnerName):]"></source>
        <field>Primary Partner Id</field>
      </mvg_primary_selector>
|    </cell>
       <cell size="5">
    </cell>cell size="22">
      <button id="btn_mvgChannelPartner">
        <text>...</text>
      </button>
    </cell>
  </stack>
</cell>
```

Make sure you specify the Id for the button and the Id for the primary selector in the same way that you specified them in the script.

4 Increase the size of the cell that contains the label, the button, and the selector control:

a Locate the following section:

```
Category bar
```

b Locate the fifth cell that is included in the Category bar section.

c Change the code of the fifth cell to the following code:

```
<cell size="153">
```

5 Save and then close the forms_xx.xml file.

Customizing the Validation Message and Labels

This task is a step in "Process of Modifying the Customization Package to Add an MVG" on page 351.

In this topic, you customize the validation message and a label for the dialog box and forms.

To customize the validation message and labels

1. Use an XML editor open the package_res.xml file.
2. Add the following code to the Script section:

```
<str key="msg_channel_partner_present">This channel partner is already present in the list.</str> <str key="msg_channel_partner_is_primary">This channel partner is primary. The primary record cannot be removed. To remove this record, make another record primary first.</str> <str key="msg_channel_partner_add_caption">Add channel partner.</str><str key="lbl_channel_partner">Lead Partner Name</str>
```

3. Add the same code that you added in Step 2 to the Messages section.
4. Close and then save the package_res.xml file.

The following code specifies the values for labels you use in the dialog box and form layouts:

```
<str key="lbl_channel_partner">Lead Partner Name</str>
```

Publishing and Testing a Custom MVG Field

This task is a step in "Process of Creating MVG Fields" on page 344.

In this topic, you publish and test your customization.

To publish and test a custom MVG field

1. Publish your changes.

   For more information, see "Using the Windows Registry to Control Siebel CRM Desktop" on page 105.

2. Test your changes:
   a. Open the client and then navigate to the Opportunity form.
   b. Verify that the form includes an MVG for the Lead Partner Name field.
   c. Click the MVG that Siebel CRM Desktop displays next to the Lead Partner Name field, and then verify that CRM Desktop does the following:
      - Displays the Channel Partners MVG dialog box
      - Displays the list of partners in the Associated Channel Partners window of the dialog box
      - Includes a partner record in the Primary window
   d. Enter letters in the Enter Value to Find Record window.
   e. Verify that CRM Desktop automatically displays records in accordance with the letters you enter.

f Verify the salesbook control. You click the Salesbook icon and then verify that CRM Desktop displays the SalesBook dialog box, and that this dialog box displays a list of channel partners.

Example Code You Use to Add an MVG

This topic describes some of the code you use to add an MVG in this example. It includes the following topics:

Code That Adds a Custom Object Type

To add a custom object type, you add the following code anywhere in the siebel_meta_info.xml file

assist with debugging, it is recommended that you place this code after the last Type definition:

```
<object TypeId='ChannelPartner' Label='Channel Partner' LabelPlural='Channel
Partners' EnableGetIDsBatching='true' ViewMode='Sales Rep' IntObjName='Channel
Partner' SiebMsgXmlElemName='ChannelPartner'
SiebMsgXmlCollectionElemName='ListOfChannelPartner' >

<field Name='DS Updated' Label='DS Updated' DataType='DTYPE_DATETIME'
IsFilterable='no' IsHidden='yes' IOElemName='DSUpdated' />

<field Name='Id' Label='Id' IsPrimaryKey='yes' DataType='DTYPE_ID'
IsFilterable='no' IsHidden='yes' IOElemName='Id'/>

<field Name='Name' Label='Name' DataType='DTYPE_TEXT' IsPartOfUserKey='yes'
IOElemName='Name' />

<field Name='Location' Label='Location' DataType='DTYPE_TEXT' IsPartOfUserKey='yes'
IOElemName='Location'/>

</object>
```

This code does the following:

- Uses properties of the integration object and integration component as the values for attributes.
- References only a few of the many fields that exist in the Channel Partner object in Siebel CRM.
- Uses the Name and Location fields as parts of the user key. You define the natural key in the connector_configuration.xml file.

Code That Maps a Custom Object

You can map a field of a Siebel CRM object to the Outlook field or to a custom Siebel CRM Desktop field. For example, you can do the following:

- Map the Name field in Siebel CRM to the LastName field in Outlook
- Map the mapLocation field in Outlook to the Location field in Siebel CRM

To map objects, you add the following code to the siebel_basic_mapping.xml file:

```
<type id="ChannelPartner" hidden_folder="true" folder_type="10"
display_name="CHPT">
  <form message_class="IPM.Contact.SBL.Channel_Partner" display_name="Channel
Partner" icon="type_image:User:16"></form>
  <field id="Name">
  <reader>
    <mapi_std>
      <mapi_tag id="0x3A110000"></mapi_tag>
      <convertor><string/></convertor>
    </mapi_std>
  </reader>
  <writer>
    <Outlook_std>
      <Outlook_field id="LastName"></Outlook_field>
      <convertor><string/></convertor>
    </Outlook_std>
  </writer>
  <reader>
    <mapi_std>
      <mapi_tag id="0x3A060000"></mapi_tag>
      <convertor><string/></convertor>
    </mapi_std>
  </reader>
  <writer>
    <Outlook_std>
      <Outlook_field id="FirstName"></Outlook_field>
      <convertor><string/></convertor>
    </Outlook_std>
  </writer>
    </field>
      <field id="Location">
  <reader>
    <mapi_user>
      <user_field id="sbl Location" ol_field_type="1"></user_field>
      <convertor><string/></convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Location" ol_field_type="1"></user_field>
      <convertor><string/></convertor>
    </Outlook_user>
  </writer>
  </field>
</type>
```

Note the following requirements:

- The value for the *type id* tag in the siebel_basic_mapping.xml file must equal the value in the TypeId object in the siebel_meta_info.xml file.
- If you define a new type, then you must include the *form* tag. The forms_xx.xml file includes the form definition. The value for the form tag must equal the *form id* attribute in the form definition. This example does not require a form, so the form tag is empty. Even if your example does not require a form, to uniquely identify a type, you must define a value for the mesage_class attribute. This value must start with the following code:

```
IPM.Contact.SBL
```

Code That Configures Synchronization for a Custom Object

To configure synchronization for a custom object, you add the following code to the connector_configuration.xml file:

```
<type id="ChannelPartner">
  <view label="Channel Partner" label_plural="Channel Partners"
small_icon="type_image:Account:16" normal_icon="type_image:Account:24"
large_icon="type_image:Account:48">
  </view>
    <synchronizer name_format=":[:(Name):]">
  <links>
  </links>
  <natural_keys>
    <natural_key>
      <field>Name</field>
      <field>Location</field>
    </natural_key>
  </natural_keys>
  </synchronizer>
</type>
```

Note the following:

- The value in the *type id* tag must equal the value in the TypeId object in the meta_info.xml file.
- The natural_key tag includes the Name and Location fields as part of the user key.

Code That Adds a Lookup View

To add a lookup view, you add the following code to the lookup_view_defs.xml file. For more information, see "Customizing the SalesBook Control" on page 163:

```
<lookup_view_def key="lookup:channel_partners">
  <display name="Channel Partners"></display>
<filter dasl="[http://schemas.microsoft.com/mapi/proptag/0x001A001E] &gt;=
'IPM.Contact.SBL.Channel_Partner' AND [http://schemas.microsoft.com/mapi/proptag/
0x001A001E] &lt;= 'IPM.Contact.SBL.Channel_Partner'"></filter>
    <view id="channel_partner:salesbook"></view>
    <quick_lookup dasl_format="[http://schemas.microsoft.com/mapi/id/{00062004-
0000-0000-C000-000000000046}/8005001E] = '%s'"></quick_lookup>
```

```
    <type id=""></type>
  </lookup_view_def>
```

Table 24 describes important attributes you use in this code.

Table 24. Attributes in the Code That Adds a Lookup View

Attribute	Description
key	Id of the lookup control.
display name	Caption of the lookup control.
filter	Object that Siebel CRM Desktop displays on the lookup control. To specify an object type, you use the message_type attribute in the siebel_basic_mapping.xml file.
view	Id of the salesbook control that CRM Desktop uses for this lookup.
type id	The type of object that CRM Desktop creates if the user clicks New on the lookup control. If this attribute is empty, then CRM Desktop disables the button.

Code That Adds a View

To add the definition for a view, you add the following code to the views.xml file:

```
<str key="channel_partner:mvg">
  <![CDATA[<?xml version="1.0"?>
  <view type="table">
  <viewname>Phone List</viewname>
  <viewstyle>table-layout:fixed;width:100%;font-family:Segoe UI;font-
style:normal;font-weight:normal;font-size:8pt;color:Black;font-charset:0</
viewstyle>
  <viewtime>0</viewtime>
  <linecolor>8421504</linecolor>
  <linestyle>3</linestyle>
  <gridlines>1</gridlines>
  <collapsestate/>
  <rowstyle>background-color:window;color:windowtext</rowstyle>
  <headerstyle>background-color:#D3D3D3</headerstyle>
  <previewstyle/>
  <arrangement>
    <autogroup>0</autogroup>
    <collapseclient/>
  </arrangement>
  <multiline>
    <width>0</width>
  </multiline>
  <column>
    <name>HREF</name>
    <prop>DAV:href</prop>
    <checkbox>1</checkbox>
  </column>
    <column>
```

```
      <type>string</type>
      <heading>Name</heading>
      <prop>http://schemas.microsoft.com/mapi/string/{00020329-0000-0000-C000-
000000000046}/sbl%20PartnerName</prop>
      <width>426</width>
      <style>padding-left:3px;;text-align:left</style>
      <editable>1</editable>
      <userheading>Name</userheading>
    </column>
  <column>
    <type>string</type>
    <heading>Location</heading>
    <prop>http://schemas.microsoft.com/mapi/string/{00020329-0000-0000-C000-
000000000046}/sbl%20PartnerLocation</prop>
    <width>426</width>
    <style>padding-left:3px;;text-align:left</style>
    <editable>1</editable>
    <userheading>Location</userheading>
  </column>
  <orderby>
    <order>
      <heading>File As</heading>
        <prop>urn:schemas:contacts:fileas</prop>
        <type>string</type>
      <sort>asc</sort>
    </order>
  </orderby>
  <groupbydefault>0</groupbydefault>
  <previewpane>
    <markasread>0</markasread>
  </previewpane>
  </view>]]>
</str>
```

Code That Adds a New Association

To add a new association, you add the following code to the business_logic.js file:

```
var opportunity_channel_partner =
  add_mvg_link("Opportunity", "ChannelPartner", "Primary Partner Id",
null,"Opportunity.Channel_Partner.Association", "OpportunityId",
"ChannelPartnerId",null, "ChannelPartnerStatus",null,
["lookup:channel_partners"],false, false, false, true);
  deny_primary_delete(opportunity_channel_partner.mvg1);//optional
opportunity_channel_partner.mvg1.dialog_template_params = {
  "dialog_caption": "#obj_activity_employee_plural","autocomplete_display_format":
":[:(Name):]", "associations_view_caption": "#head_associated_channel",
"associations_view_id": "channel_partner:mvg", "primary_selector_display_format":
":[:(PartnerName):]"
  }
```

The ChannelPartnerStatus field contains the status of the Channel Partner object, such as unsaved, deleted, and so on.

Table 25 describes the important attributes you can use with the add_mvg_link function.

Table 25. Attributes in the Code That Add a New Association

Attribute	Description
left_type	The type of the first linked object. For example, Opportunity.
right_type	The type of the second linked object. For example, Channel Partner.
left_obj_primary	The field of the Opportunity object. This field contains the primary Id for the Channel Partner object.
right_obj_primary	The field of the Channel Partner object. This field contains the primary Id for the Opportunity object.
assoc_type	The Id of the association type described in the siebel_meta_info.xml file and the siebel_basic_mapping.xml file.
left_link	The field of the association object that contains the Id of the opportunity.
right_link	The field of the association object that contains the Id of the channel partner.
left_assoc_status	The field of the association that contains the status of the opportunity.
right_assoc_status	The field of the association that contains the status of the channel partner.
left_objs_view_ids	The list of lookup view definitions you must use to display opportunity objects.
right_objs_view_ids	The list of lookup view definitions you must use to display channel partner objects.
left_primary_refresh_required	If the primary object is changed, and if the left_primary_refresh_required attribute is true, then Siebel CRM Desktop updates the object Id for the primary field of the opportunity that you specify in the left_obj_primary attribute. If you use the following tag in the siebel_basic_mapping.xml file for this field, then set the left_primary_refresh_required attribute to true: `writer class="link_fields"`
right_primary_refresh_required	If the primary object is changed, and if the right_primary_refresh_required attribute is true, then CRM Desktop updates the object Id for the primary field of the channel partner that you specify in the right_obj_primary attribute. If you use the following tag in the siebel_basic_mapping.xml file for this field, then set the right_primary_refresh_required attribute to true: `writer class="link_fields"`

Table 25. Attributes in the Code That Add a New Association

Attribute	Description
assoc_left_link_refresh_required	If the opportunity object is changed, then CRM Desktop updates the OpportunityId field of the association that you specify in the left_link attribute. If you use the following tag in the siebel_basic_mapping.xml file for this field, then set the assoc_left_link_refresh_required attribute to true: `writer class="link_fields"`
assoc_right_link_refresh_required	If the channel partner object is changed, then CRM Desktop updates the ChannelPartnerId field of the association that you specify in the right_link attribute. If you use the following tag in the siebel_basic_mapping.xml file for this field, then set the assoc_right_link_refresh_required attribute to true: `writer class="link_fields"`

Code That Creates a SalesBook Control

To create a salesbook control, you add the following code to the views.xml file:

```
<str key="channel_partner:salesbook">
  <![CDATA[<?xml version="1.0"?>
    <view type="table">
      <viewname>Phone List</viewname>
      <viewstyle>table-layout:fixed;width:100%;font-family:Segoe UI;font-
style:normal;font-weight:normal;font-size:8pt;color:Black;font-charset:0</
viewstyle>
      <viewtime>0</viewtime>
      <linecolor>8421504</linecolor>
      <linestyle>3</linestyle>
      <gridlines>1</gridlines>
      <newitemrow>1</newitemrow>
      <collapsestate/>
      <rowstyle>background-color:window;color:windowtext</rowstyle>
      <headerstyle>background-color:#D3D3D3</headerstyle>
      <previewstyle/>
      <arrangement>
        <autogroup>0</autogroup>
        <collapseclient/>
      </arrangement>
      <multiline>
        <width>0</width>
      </multiline>
      <column>
        <name>HREF</name>
        <prop>DAV:href</prop>
        <checkbox>1</checkbox>
      </column>
      <column>
```

```
        <heading>Last Name</heading>
        <prop>urn:schemas:contacts:sn</prop>
        <type>string</type>
        <width>322</width>
        <style>padding-left:3px;;text-align:left</style>
        <editable>1</editable>
      </column>
      <column>
        <type>string</type>
        <heading>Location</heading>
        <prop>http://schemas.microsoft.com/mapi/string/{00020329-0000-0000-C000-
000000000046}/sbl%20Location</prop>
        <width>322</width>
        <style>padding-left:3px;;text-align:left</style>
        <editable>1</editable>
        <userheading>Location</userheading>
      </column>
      <orderby>
        <order>
          <heading>File As</heading>
          <prop>urn:schemas:contacts:fileas</prop>
          <type>string</type>
          <sort>asc</sort>
        </order>
      </orderby>
      <groupbydefault>0</groupbydefault>
      <previewpane>
        <markasread>0</markasread>
      </previewpane>
    </view>]]>
</str>
```

Making an MVG Field a Required Field

This topic describes how to make an MVG field a required field. The example in this topic describes how to make the Business Address field in the Account form a required field. It is recommended that you make an MVG field a read-only field only if necessary. Making an MVG field a read-only field might increase the effort required to replace a control at some future point, and also might add redundant code.

You can make an MVG field a required field only with an autocomplete list. You cannot make an MVG field a required field with the mvg_primary_selector2 control. If CRM Desktop creates a new record, and if the MVG field is a required field in this record, then this field must contain an association before CRM Desktop can save the record. The mvg_primary_selector2 control requires that the user use the MVG dialog box to add this association, but CRM Desktop cannot display this dialog box until it saves the record, which it cannot do without the association. For more information about these controls, see “Overview of Customizing Multi-Value Groups” on page 343.

To make an MVG field a required field

1 Replace the primary_selector control with the autocomplete_control:

a Use an XML editor to open the forms_xx.xml file.

b Locate the following code:

```
<cell>
  <mvg_primary_selector2 id="business_address_mvg" tab_order="27">
    <source type="Account.Business_Address.Association" left_id="AccountId"
item_value="Business AddressId"
    display_format=":[:(Street Address):] :[:(City):], :[:(State):]
:[:(Postal Code):] :[:(Country):]" />
    <field>Primary Address Id</field>
  </mvg_primary_selector2>
</cell>
```

This code renders the primary control.

c Replace the code you located in Step b with the code that renders an autocomplete list in the Business Address field.

For more information, see "Code That Renders an Autocomplete List" on page 366.

d Save, and then close the forms_xx.xml file.

e Use the following code to register the MVG:

```
register_mvg_dialog(ctx, "Business_Address", "sales_team_mvg",
"btn_sales_team_mvg", { "use_autocomplete_list": true });
```

For more information, see "Registering MVG Controls" on page 176.

2 Make the field that CRM Desktop uses with the autocomplete list a required field:

a Use a JavaScript editor to open the forms.js file.

b Locate the form handler for the form where you added the XML code in Step 1.

In this example you added the XML code in the Account form, so you must locate the form handler that the Account form uses.

c Modify the code you located in Step b to the following:

```
ctx.validator.validate_empty_field("Primary Address Id",
"business_address_mvg","msg_account_business_address_validation");
```

where:

- `validate_empty_field` function makes sure the field that CRM Desktop uses with the autocomplete list is a required field. For more information about this function, see "Making Sure Users Enter Information in a Field" on page 226.

3 Disable the MVG button:

a Add the following code to the end of the form handler function that you modified in Step 2:

```
ctx.form.[mvg_button_field_name].enabled = true;
```

b Locate the form_saved function, and then add the following code to this function:

```
ctx.form.btn_business_address_mvg.enabled = true;
```

The form_saved function resides in the form handler that you modified in Step 1. This configuration makes sure that CRM Desktop does not display the MVG dialog box before the it saves the object. It forces the user to enter characters in the field, and then to use the autocomplete list to choose an association.

c. Save, and then close the forms.js file.

4. Test your work:

a. Log in to the client, and then navigate to the Account form.

b. Create a new record.

c. Attempt to save the record without entering a value in the Business Address field.

d. Make sure CRM Desktop displays the dialog box informing you that you must enter a business address.

e. Click in the Business Address field, and then make sure it displays the account to the left of the semicolon, and the street address, city, state, postal code, and country to the right of the semicolon.

f. Click the street address and make sure CRM Desktop displays the autocomplete list.

g. Enter a few characters and make sure CRM Desktop modifies values in the autocomplete list so that they match the values you enter.

h. Choose a value from the autocomplete list and make sure CRM Desktop sets the value in the field to the value you choose.

Code That Renders an Autocomplete List

This topic describes the code that renders an autocomplete list. For more information about autocomplete lists, including the parent and child relationship that they use, see "Registering Autocomplete Controls" on page 173.

The following code renders an autocomplete list:

```
<autocomplete_list id="mvg_name" tab_order="tab_order">
  <items format=":[:{child_Id@child_object_type/(:[:(child_fields):])}:]">
  <source type="auto" name="search_string"/>
    <restriction>:
      <binary field="parent_field" condition="eq">
        <value type="variable">id()</value>
      </binary>
      <binary field="child_field" condition="ne">
        <value type="string">deleted</value>
      </binary>
    <suggestion format=":[:(autocomplete_source):]">
      <source type="auto" name="autocomplete_source_object_type"/>
    </suggestion>
</autocomplete_list>
```

where:

- *mvg_name* identifies the name of the MVG control.
- *tab_order* sets the position for the control that CRM Desktop makes active if the user presses the TAB key.
- *child_Id* identifies the object that contains the Id of the object that CRM Desktop displays to the right of the semicolon in the MVG field.
- *child_object_type* identifies the type of object that CRM Desktop displays to the right of the semicolon in the MVG field.
- *child_fields* identifies the fields that CRM Desktop displays to the right of the semicolon in the MVG field. You can use the following format to specify more than one field:

  "[:(*field_name*):][:(*field_name*):][:(*field_name*):]"

  CRM Desktop adds each field consecutively to the right of the semi-colon. For example, assume you specify the following:

  ```
  (:[:(Street Address):] :[:(City):], :[:(State):] :[:(Postal Code):]
  :[:(Country):])
  ```

  In this example, CRM Desktop displays Main Street: Los Angeles CA: 94865: USA. The character that occurs between the brackets specifies the separator. This example uses a colon:

  ```
  ]:[
  ```

- *search_string* includes the association that CRM Desktop uses to search in Outlook and in DB_FACADE. DB_FACADE is the native storage that CRM Desktop uses. It searches for an association object. An *association object* is a type of object that CRM Desktop defines in the siebel_basic_mapping.xml file.
- *parent_field* identifies the field that CRM Desktop uses as the source for the field that it displays to the left of the semicolon in the MVG field. CRM Desktop restricts the child fields that it displays to only fields that are children of the field that left_field identifies.
- *child_field* identifies the field that CRM Desktop uses as the source for the field that it displays to the right of the semicolon in the MVG field. CRM Desktop displays this field the first time it displays the MVG. If the user uses the autocomplete list to modify this field, then CRM Desktop sets right_field to the value that the user chooses.
- *autocomplete_source* identifies the fields that CRM Desktop uses as the source for the values that it displays in the autocomplete list. You can use the same format that child_fields uses to specify more than one field. To view an example autocomplete list, see Figure 19 on page 344.
- *autocomplete_source_object_type* identifies the object type of the objects that autocomplete_source contains.

For example, the following code renders an autocomplete list in the Business Address field:

```
<autocomplete_list id="business_address_mvg" tab_order="27">
  <items format=":[:{Business AddressId@Business_Address/(:[:(Street Address):]
:[:(City):], :[:(State):] :[:(Postal
    Code):] :[:(Country):])}">
<source type="auto" name="Account.Business_Address.Association"/>
    <restriction>
      <binary field="Business AddressId" condition="eq">
```

```
        <value type="variable">id()</value>
      </binary>
      <binary field="Status" condition="ne">
        <value type="string">deleted</value>
      </binary>
    </restriction>
  </items>
  <suggestion format=":[:(Street Address):] :[:(City):], :[:(State):] :[:(Postal
Code):] :[:(Country):]">
    <source type="auto" name="Business_Address"/>
  </suggestion>
</autocomplete_list>
```

Configuring Autocomplete Lists and Primary Selectors for MVGs

This topic describes how to configure Siebel CRM Desktop to display autocomplete lists and primary selectors for MVGs. You can use the autocomplete_list control with an MVG control. The autocomplete_list control is similar to the autocomplete control except it includes more functionality. You can use the autocomplete_list control instead of the mvg_primary_selector2 control. For more information about these controls, see "Overview of Customizing Multi-Value Groups" on page 343.

To configure autocomplete lists and primary selectors for MVGs

1. Use an XML editor to open the forms_xx.xml file.
2. Add the following code. This code adds an autocomplete list on the Contact object for the Account field:

```
:<autocomplete_list id="account_mvg" tab_order="9">
  <items format=":[:{AccountId@Account/(:[:(Name):])}:]">
    <source type="auto" name="AccountJointContact"></source>
    <restriction>
      <binary field="ContactId" condition="eq">
        <value type="variable">id()</value>
      </binary>
      <binary field="AccountStatus" condition="ne">
        <value type="string">deleted</value>
      </binary>
    </restriction>
  </items>
  <suggestion format=":[:(Name):]">
    <source type="auto" name="Account"></source>
  </suggestion>
</autocomplete_list>
```

For more information about this code, see "Code That Renders an Autocomplete List" on page 366

Configuring Siebel CRM Desktop to Use Autocomplete Lists or Primary Selectors in MVGs

You can configure Siebel CRM Desktop to use autocomplete lists or primary selectors in MVGs.

To configure CRM Desktop to use autocomplete lists or primary selectors in MVGs

1. Use a JavaScript editor to open the forms.js file.
2. Locate the register_mvg_dialog function.

   For more information about this function, see "Registering MVG Controls" on page 176.
3. Set the use_autocomplete_list parameter to one of the following values:

   - **true.** Use autocomplete list.
   - **false.** Use mvg primary selector.

   For example:

   ```
   register_mvg_dialog(ctx, "Position", "sales_team_mvg", "btn_sales_team_mvg", {
   "use_autocomplete_list": false });
   ```

   For a detailed example that uses the register_mvg_dialog function, see "Controlling the Search in Siebel Button That Does Online Lookup" on page 198.

12 Customizing Authentication

This topic describes how to customize authentication for Siebel CRM Desktop. It includes the following topics:

Overview of Customizing Authentication

This topic describes an overview of single sign on for Siebel CRM Desktop. It includes the following topics:

CRM Desktop SSO (CRM Desktop Single Sign On) is a single sign on feature that allows the user to log in to CRM Desktop one time and use CRM Desktop features without being prompted to log in again to gain access to features that use access control. It allows you to implement single sign on for the CRM Desktop client. It uses the Web transport protocol and the HTTP or HTTPS protocol. It supports standard Web browser functionality, such as HTTP forms, cookies, and process redirects. It provides the following capabilities:

- Customizable architecture. You can install CRM Desktop SSO as an add-on. Beginning with Siebel CRM Desktop version 3.7, the CRM Desktop installer automatically includes CRM Desktop SSO.
- Supports your existing Web single sign on feature. You can customize CRM Desktop SSO so that it works in your network topology and for mobile or remote access.
- Supports your custom user interface. You can use CRM Desktop SSO that is automated and transparent to the user. You can also use it with a custom authentication screen that you already use, perhaps that includes your company branding.
- Supports typical login name and password authentication or custom authentication that requires more than just a login name and password.
- Handles single sign on, session, and network errors in a way that is transparent to the user.

Unless noted otherwise, support for features that this chapter describes begins with Siebel CRM Desktop version 3.7.

Authentication That Comes Predefined with Siebel CRM Desktop

This topic describes authentication that comes predefined with Siebel CRM Desktop.

Direct Connection

Siebel CRM Desktop can connect to an unprotected EAI (Enterprise Application Integration) endpoint for a direct connection. It comes predefined to use direct connection. For more information, see "About Authentication and Session Management" on page 31.

Figure 20 includes the dialog box that CRM Desktop displays for a direct connection. This dialog box allows the user to enter the user name and password.

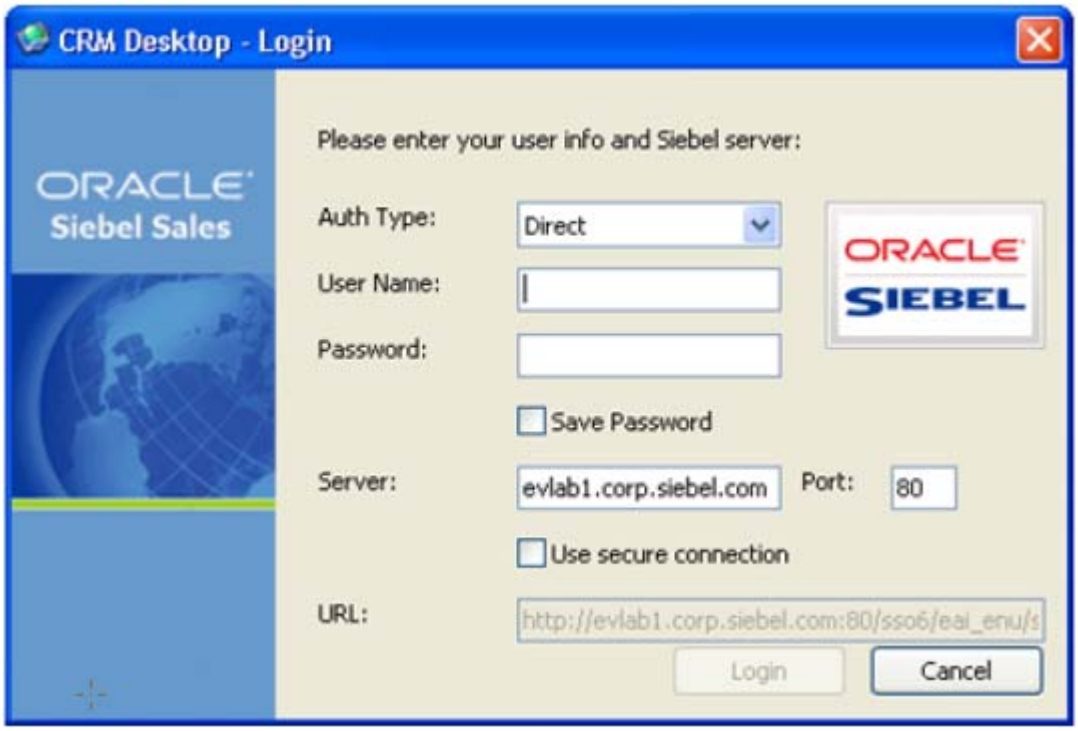


Figure 20. Dialog Box That CRM Desktop Displays for a Direct Connection

Single Sign On

CRM Desktop SSO can protect EAI on the client. In this situation, CRM Desktop SSO displays a dialog box that is identical to the dialog box in Figure 20 on page 372 except the user sets the Auth Type field to SSO and the Siebel:SSOUser parameter determines if CRM Desktop SSO enables the User Name field. For more information, see "Registry Keys That Control SSO for Credentials" on page 420.

If you configure CRM Desktop SSO to use interactive authentication, then it displays another dialog box after the user clicks Login in the CRM Desktop-Login dialog box. Figure 21 includes an example of this second dialog box. For more information, see "Types of Authentication That You Can Use With CRM Desktop SSO" on page 373.

Figure 21. Example Dialog Box That CRM Desktop SSO Displays for Interactive Authentication

Types of Authentication That You Can Use With CRM Desktop SSO

This topic describes the types of authentication that you can use with CRM Desktop SSO.

Interactive Authentication

Interactive authentication is a type of authentication in CRM Desktop SSO that displays a second login dialog box that is native to the client. It displays this native dialog box in the following situations:

- After the user sets values in the CRM Desktop - Login dialog box and then clicks Login.
- The first time CRM Desktop logs in to the Siebel Server after Outlook restarts.
- A previously obtained SSO session expires.

Siebel CRM supports interactive authentication beginning with Siebel CRM Desktop version 3.7.

If you enable CRM Desktop SSO, then it uses interactive authentication by default. It is the only SSO authentication that comes predefined with CRM Desktop.

Interactive authentication includes the following functionality:

- **Detect session expiration.** If a CRM Desktop SSO session expires, then CRM Desktop SSO prompts the user to provide credentials and then reestablishes the session.

- **Multifactor authentication.** Supports more than only the login name and password from the the dialog box that is native to the browser, such as requiring code from a security token in addition to the password. The CRM Desktop SSO login dialog box can include more fields, images, a list of questions that the user must answer, ActiveX controls, and other items that your implementation requires for authentication. It can support an input field for an RSA (Rivest Shamir Adleman) token or other information that only the user can provide.
- **Supports Internet Explorer.** The CRM Desktop SSO log-in dialog box is an Internet Explorer ActiveX dialog box.

Interactive authentication provides the following benefits:

- Allows CRM Desktop SSO to support a more complex SSO login
- Allows you to use script to customize an SSO login on the client that supports a nonstandard configuration.
- Interactive SSO provides the following challenges:
- Cannot store credentials in the product configuration that CRM Desktop SSO can use later. This situation might require the user to reenter credentials during the first and subsequent SSO session logins.
- Supports only Internet Explorer version 7 or later.

How Siebel CRM SSO Starts and Stops Interactive Authentication

If the POST request to the EAI web service returns one of the following values, then CRM Desktop SSO starts interactive authentication:

- HTTP Redirect (302)
- HTML content

CRM Desktop SSO does one or more request and reply iterations during an interactive authentication. It monitors these iterations for the presence of one of the following stop conditions. If it encounters one of these conditions, then it stops the iteration:

- If the setting for the EndpointRegExp registry key is:
  - **Not defined.** The URL must match the URL parameter that resides on the Login dialog box. This setting is the default value.
  - **Defined.** The destination URL must match the EndpointRegExp registry setting.
- If the setting for the SuccessLoginRegExp registry key is:
  - **Not defined.** The HTML body must match the expression that the Siebel EAI server returns. This setting is the default value. For more information, see the description about SuccessLoginRegExp in “Registry Keys That Control SSO for Siebel CRM Desktop” on page 419.
  - **Defined.** The HTML body must match the setting of the SuccessLoginRegExp registry key.

The user can also close the dialog box at any time to stop interactive authentication. CRM Desktop interprets this closure and cancels interactive authentication. For more information, see “Registry Keys That Control SSO for Siebel CRM Desktop” on page 419.

Noninteractive Authentication

Noninteractive authentication is a type of authentication in CRM Desktop SSO that uses a separate dialog box as the login window for Siebel CRM Desktop. It is similar to the dialog box that Figure 16 on page 346 displays. It includes the following functionality:

- **Save password.** The user can set the Save Password option on the CRM Desktop - Login dialog box to one of the following values:
  - **Include a check mark.** CRM Desktop SSO stores encrypted credentials in the Windows Registry. It does not prompt the user for credentials the next time the user starts CRM Desktop.
  - **Do not include a check mark.** CRM Desktop SSO prompts the user for credentials the next time the user starts CRM Desktop and after the first SSO session starts. This behavior typically occurs during the first synchronization that occurs after the user restarts CRM Desktop.
- **Detect session expiration.** If a CRM Desktop SSO session expires, then CRM Desktop SSO reestablishes the session without involving the user. CRM Desktop SSO requires user interaction only if the user credentials are not valid.
- **Use only the login name and password.** Noninteractive authentication does not support multifactor authentication, such as requiring code from a security token in addition to the password.
- **Detect invalid user password.** The SSO script can detect if the user enters an invalid password and then react accordingly.
- **Does not include Web pages.** The user interacts only with CRM Desktop Web pages that do not include CRM Desktop SSO.

Benefits and Challenges of Using Noninteractive Authentication

If CRM Desktop SSO uses noninteractive authentication, then SSO script interprets HTTP requests and replies that do not involve the user. It handles HTTP redirects, HTML form submits, automatic submits, and so on. It emulates the user interaction that typically occurs during a login that is native to the Web browser.

Noninteractive SSO provides the following benefits:

- CRM Desktop SSO can log in without user intervention.
- CRM Desktop SSO can reestablish a session automatically. User interaction is required only if a password lockout occurs.

Noninteractive SSO provides the following challenges:

- Requires slightly more customization than interactive authentication.

The implementation can be complex and difficult to scale. If you modify the login process that your company uses, then these modifications might be difficult to implement. For example, if your company must change from a simple username and password sign on to a complex sign on that requires more than these factors. In this situation, you must modify, test, and redeploy the SSO script.

Single Sign On Services That CRM Desktop SSO Supports

Predefined CRM Desktop SSO supports the following Web single sign on services:

- Oracle Identity Management
- IBM Tivoli
- Computer Associates Siteminder

CRM Desktop can be customized to support other SSO authorities.

For information about the operating systems that CRM Desktop SSO supports, see the Certifications tab on My Oracle Support. For information about Certifications, see article Using Certifications on My Oracle Support for Siebel CRM Products (Doc ID 1492194.1) on My Oracle Support.

Installing CRM Desktop SSO

This topic describes how to install CRM Desktop SSO. It includes the following topics:

- "Setting Windows Registry Keys to Enable CRM Desktop SSO" on page 378
- "Options for Installing CRM Desktop SSO" on page 378
- "Removing or Upgrading CRM Desktop SSO" on page 382

Starting with Siebel CRM Desktop version 3.7, it is not necessary to install CRM Desktop SSO. It comes predefined starting with these version. For more information about using an installation package, see "Overview of Installing the CRM Desktop Add-In" on page 81.

To install CRM Desktop SSO:

1. Verify the network and infrastructure:
   a. Modify the Windows Registry settings, as necessary.

      For more information, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380.

   b. Make sure SSO script supports the single sign on capabilities that your implementation requires.

      CRM Desktop SSO cannot connect directly to a single sign on system. To operate properly, it requires SSO script that you customize for the target single sign on system. For a list of options, see "Windows Registry Keys You Must Set to Enable CRM Desktop SSO" on page 416.

   c. Determine the sequence you will use to install CRM Desktop SSO.

      You can install, remove, or upgrade CRM Desktop SSO independent of CRM Desktop. You can install it before or after you install CRM Desktop.

2. Make sure you are a member of the Administrators group for the operating system that you are using on the client computer.

   This membership provides the rights you require to run the executable file that CRM Desktop SSO uses in the installation package.

3 Set the Windows Registry keys.

For more information, see "Setting Windows Registry Keys to Enable CRM Desktop SSO" on page 378.

4 (Optional) The installer calls the UAC (User Account Control) prompt in Windows. To disable this prompt, you can set the following properties on the operating system on the client computer:

```
ALLUSERS=2
MSIINSTALLPERUSER=1
```

5 Copy the InvisibleSSOModule.msi file from the release location to the client computer.

To install CRM Desktop SSO, you use an installation package that comes predefined as a single MSI file. This file includes the following data:

- Installation information for CRM Desktop SSO
- CRM Desktop SSO binary files

To copy this file, you can do one of the following

- Manually copy the InvisibleSSOModule.msi file to the client computer.
- Use third-party deployment software to deploy the InvisibleSSOModule.msi file to multiple computers. For more information, see "Installing Siebel CRM Desktop in the Background" on page 98.

6 Log in to the client computer.

7 If Microsoft Outlook is open, then close it.

8 Locate the InvisibleSSOModule.msi installation package.

This location depends on where the user saves the InvisibleSSOModule.msi file. The following directory is a typical location:

```
C:\Documents And Settings\username\Desktop
```

9 Run the InvisibleSSOModule.msi installation package:

a In the Welcome dialog box, click Next.

b In the Customer Information dialog box, specify the user name and organization and then chose to do this install for a single user or for any user who uses this client computer.

If you choose to do this installation for any user who uses this client computer, then you must be a member of the Administrators group. If you are not, then this installation will fail.

It is recommended but not required that you use the same installation configuration for CRM Desktop SSO that you use for CRM Desktop. For example, if you install CRM Desktop for each user, then it is recommended that you install CRM Desktop SSO for each user. if you install CRM Desktop for anyone who uses this computer, then it is recommended that you install CRM Desktop SSO for anyone who uses this computer. If you do not use this configuration, then CRM Desktop SSO might not be available for some users.

c In the Destination Folder dialog box, specify the folder where the installer must install CRM Desktop SSO.

You can specify a directory. For more information, see "Setting the Installation Directory for CRM Desktop SSO" on page 381.

d In the Ready to Install the Program dialog box, click Install.

Because you can install CRM Desktop for multiple users, the user who is currently logged in can view application files that CRM Desktop stores in the default directory described in "Setting the Installation Directory for CRM Desktop SSO" on page 381.

e In the InstallShield Wizard dialog box, click Finish.

The installer installs CRM Desktop SSO. The next time CRM Desktop starts it loads CRM Desktop SSO according to the Windows Registry settings. For more information, see "For more information, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380." on page 381.

After you complete the installation, CRM Desktop SSO is configured and CRM Desktop uses it when it communicates with the Siebel Server.

10 Notify the user that CRM Desktop SSO is installed.

Setting Windows Registry Keys to Enable CRM Desktop SSO

This topic describes Windows Registry settings that you must set if you install Siebel CRM Desktop for Siebel CRM Desktop version 3.7. You do not set registry keys with version 3.7, or later. This description applies only to installing CRM Desktop. It does not apply to using the InvisibleSSOModule.msi installer.

To set Windows Registry keys to enable CRM Desktop SSO

- Enter the required command-line parameters when you run msiexec.exe.

  For example:

  ```
  msiexec.exe /I CRMDesktop.msi SSOENABLE=1 SSOSCRIPTINCLUDEPATH=%appdata%
  SSOURL=http://myurl
  ```

  For more information, see "Windows Registry Keys You Must Set to Enable CRM Desktop SSO" on page 416.

Options for Installing CRM Desktop SSO

This topic describes options for installing CRM Desktop SSO. It includes the following topics:

Siebel CRM Desktop version 3.7 and later comes predefined with these options enabled. It is only necessary to enable these options if you are using version 3.7 or earlier.

Installing CRM Desktop SSO If You Use Autoupdate

Autoupdate is a CRM Desktop SSO feature that automatically updates the SSO script if any change occurs to this script. CRM Desktop SSO scripts must reside in a single ZIP file. The CRM Desktop SSO installer uses the SSOURL parameter to determine the file path where these scripts reside. CRM Desktop SSO uses this location to download these scripts during installation and later during updates. This configuration allows you to deploy CRM Desktop SSO across your enterprise. It makes sure that the scripts on each client are consistent and valid. If autoupdate is enabled, then CRM Desktop SSO deploys the SSO script as a single ZIP file.

To install CRM Desktop SSO if you use autoupdate

1 Enter the following parameter on the msiexec command line anywhere after `CRMDesktop.msi`:

```
SSOENABLE=1 SSOURL=URL
```

For example:

```
msiexec.exe /I CRMDesktop.msi SSOENABLE=1 SSOURL=http://myurl
```

You can also set the following optional parameters:

```
SSOSCRIPTINCLUDETEMPLATE = template
SSOCHECKINTERVAL = new interval
SSOSCRIPTFILENAME = file name
```

For more information, see “Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO” on page 380.

Installing CRM Desktop SSO If You Do Not Use Autoupdate

If you do not use autoupdate, then you configure CRM Desktop SSO to read scripts from a fixed location that you specify on the local drive. If autoupdate is not enabled, then CRM Desktop SSO deploys the SSO script as separate JavaScript files.

To install CRM Desktop SSO if you do not use autoupdate

1 Enter the following command on the msiexec command line anywhere after `CRMDesktop.msi`:

```
SSOENABLE=1 SSOUPDATEDISABLE=1 SSOSCRIPTINCLUDEPATH=path_to_scripts
```

For example:

```
msiexec.exe /I CRMDesktop.msi SSOENABLE=1 SSOUPDATEDISABLE=1
SSOSCRIPTINCLUDEPATH=%appdata%
```

You can also set the following parameters optional:

```
SSOCHECKINTERVAL = new interval
SSOSCRIPTFILENAME = file name
```

For more information, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380.

2 If you use MST or a prepackaged MSI, the you must make sure that the path that you specify for the SSOSCRIPTINCLUDEPATH parameter matches the path where you install the InvisibleSSOModule.msi file.

Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO

This topic describes how to use the Windows command line to set optional parameters for Siebel CRM SSO. CRM Desktop SSO supports all parameters that you can set in the Windows Installer msiexec command line. This option is not available starting with Siebel CRM Desktop version 3.7. CRM Desktop SSO comes predefined starting with these versions. For more information, see "Using the Windows Command Line to Set Optional Parameters" on page 100.

To use the Windows command line to set optional parameters

1 On the client computer, open the Windows command line interface.

2 Navigate to the directory that contains the InvisibleSSOModule.msi file.

For example:

```
c:\Documents And Settings\username\Desktop
```

3 Enter the Windows Installer command using the following format:

```
msiexec.exe /I InvisibleSSOModule.msi optional_parameter_1 optional_parameter_n
```

where:

- *optional_parameter* is a parameter that the installer can run. For example:

```
msiexec.exe /I InvisibleSSOModule.msi INSTALLDIR=c:\My_Custom_Directory
```

Note the following conditions:

- You must specify each optional parameter in the same command line after the name of the InvisibleSSOModule.msi file.
- To separate each optional parameter, you must enter a space.
- You can arrange optional parameters in any order.

4 Press Enter.

5 The CRM Desktop SSO setup wizard displays the welcome dialog box.

Abbreviating the Installation Procedure

To automatically run the windows that normally require user action, you can use the optional QR parameter. If you use it, then the InvisibleSSOModule.msi installation package does not display dialog boxes that require user action. This option is not available starting with Siebel CRM Desktop version 3.7. CRM Desktop SSO comes predefined starting with these versions.

To abbreviate the installation procedure

- Append the QR parameter to the msiexec command.

  For example:

  ```
  msiexec.exe /I InvisibleSSOModule.msi INSTALLDIR=c:\My_Custom_Directory /QR
  ```

  For more information, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380.

Setting the Installation Directory for CRM Desktop SSO

The InvisibleSSOModule.msi installation package saves binary files during installation in one of the following directories, by default:

- Installation directory for each user on Micorsoft Windows. For Microsoft Windows, it installs the binary files into the following folder:

  ```
  C:\Users\user\AppData\Roaming\
  ```

- Installation directory for each computer:

  ```
  C:\Program Files\InvisibleCRM\SSO\
  ```

To change this directory, the user can choose a different location in the InstallShileld wizard or you can use the INSTALLDIR parameter.

If you install CRM Desktop SSO for any user who uses this computer, then you must make sure that all users can read this directory. Predefined CRM Desktop SSO suggests a different directory depending on if you install for each user or for anyone who uses this computer.

This option is not available starting with Siebel CRM Desktop version 3.7. CRM Desktop SSO comes predefined starting with these versions.

To set the installation directory for CRM Desktop SSO

- Enter the following parameter on the msiexec command line anywhere after the InvisibleSSOModule.msi name parameter:

  ```
  INSTALLDIR=directory_path
  ```

  For example:

  ```
  msiexec.exe /I InvisibleSSOModule.msi INSTALLDIR=c:\My_Custom_Directory /QR
  ```

  For more information, see "Using the Windows Command Line to Set Optional Parameters for Siebel CRM SSO" on page 380.

Removing or Upgrading CRM Desktop SSO

This topic describes how to remove or upgrade CRM Desktop SSO for versions that occur earlier than Siebel CRM Desktop version 3.7. For information about removing multiple users, see "Removing the CRM Desktop Add-In for Multiple Users" on page 117.

Removing CRM Desktop SSO for a Single User

This topic describes how to remove CRM Desktop SSO for a single user.

To remove CRM Desktop SSO for a single user

1. Log on to the computer where CRM Desktop SSO is installed.
2. Synchronize and back up all personal data:
   a. In Microsoft Outlook, perform synchronization.
   b. Backup personal data.

      It is recommended that the user use the export feature in Microsoft Outlook to export personal data to a file.
3. Remove CRM Desktop SSO:
   a. In Microsoft Windows, click the Start menu, choose Settings, and then click Control Panel.
   b. In the Control Panel, right-click Add or Remove Programs and then choose Open.
   c. In the Add or Remove Programs dialog box, in the Currently Installed Programs window, click Invisible SSO Module and then click Remove.

      CRM Desktop removes the registry settings and the files that CRM Desktop SSO uses.

Upgrading CRM Desktop SSO

This topic describes how to upgrade CRM Desktop SSO.

To upgrade CRM Desktop SSO

1. Remove CRM Desktop SSO.

   For more information, see "Removing CRM Desktop SSO for a Single User" on page 382.
2. Install CRM Desktop SSO.

   For more information, see "Installing CRM Desktop SSO" on page 376.

About CRM Desktop SSO Architecture

This topic describes the architecture that CRM Desktop SSO uses. It includes the following topics:

Architecture That CRM Desktop SSO Uses

Figure 22 illustrates the architecture that CRM Desktop SSO uses.

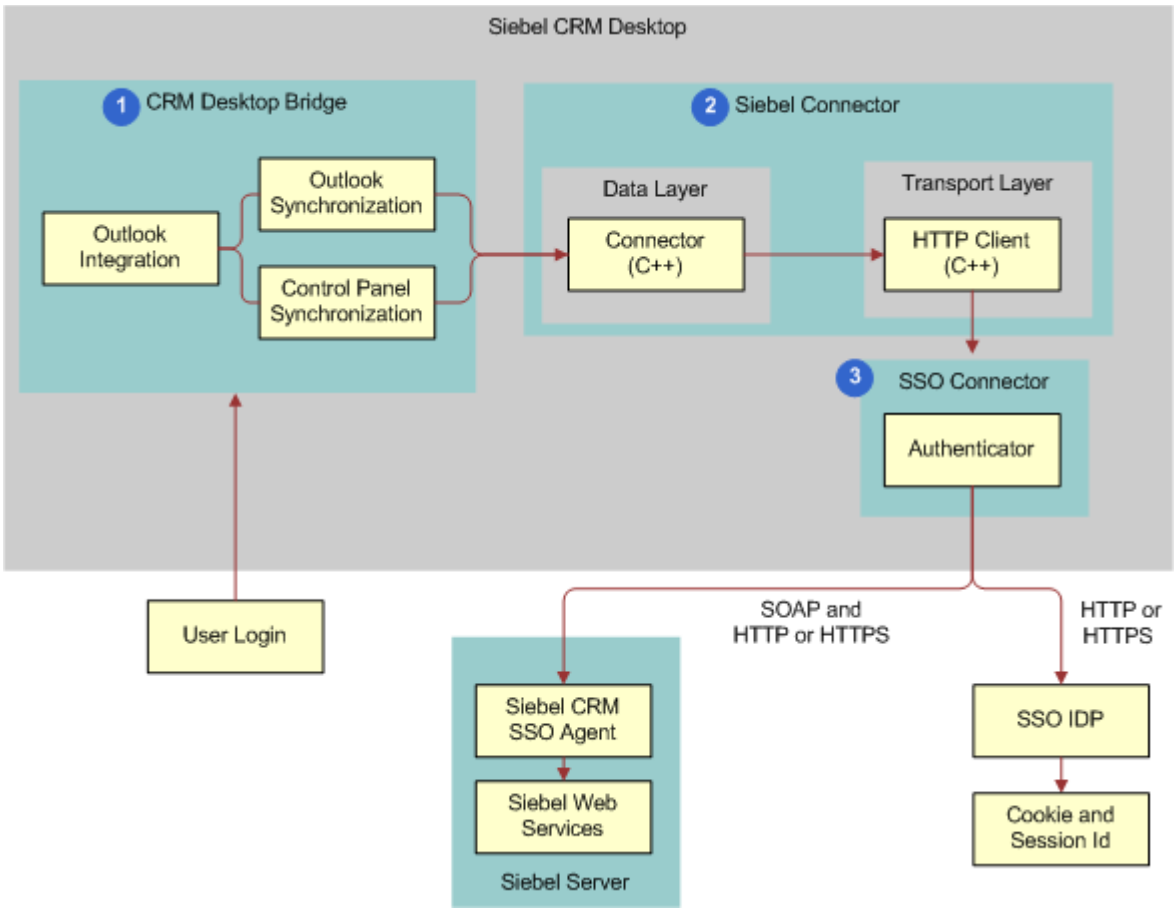


Figure 22. Architecture That CRM Desktop SSO Uses

Explanation of Callouts

The architecture that CRM Desktop SSO uses includes the following items:

1. CRM Desktop **Bridge.** Uses a customization package that allows CRM Desktop to synchronize data. This synchronization uses the credentials that the user provides.
2. **Siebel Connector.** Communicates information between CRM Desktop and Siebel CRM. It communicates through the SSO Connector to synchronize data, get the customization package, and open objects in Outlook
3. **SSO Connector.** A set of JavaScript files that CRM Desktop SSO deploys to the client during installation. CRM Desktop uses the SSO Connector as a proxy to emulate a direct communication channel to the Siebel Connector. It uses SSO script. It includes the following items:
   - **SSO session.** Includes state information and code that handles the request and reply that allows CRM Desktop SSO to establish, monitor, and exchange data between the connector and the Siebel Server. The Outlook Bridge uses the connector to start this session. To start multiple SSO sessions, this bridge can create a separate connector instance for each session.
   - **SSO session data.** Includes a global JavaScript context and other state information that the SSO script uses to track an SSO session.
   - **SSO session handler.** Each handler handles communication for a single SSO session.
4. CRM Desktop **SSO Proxy.** Proxy for the CRM Desktop SSO Connector that handles all requests from this connector and provides replies back to this connector.

Flow That CRM Desktop SSO Uses During Authentication

Figure 23 illustrates the flow that CRM Desktop SSO uses during authentication.

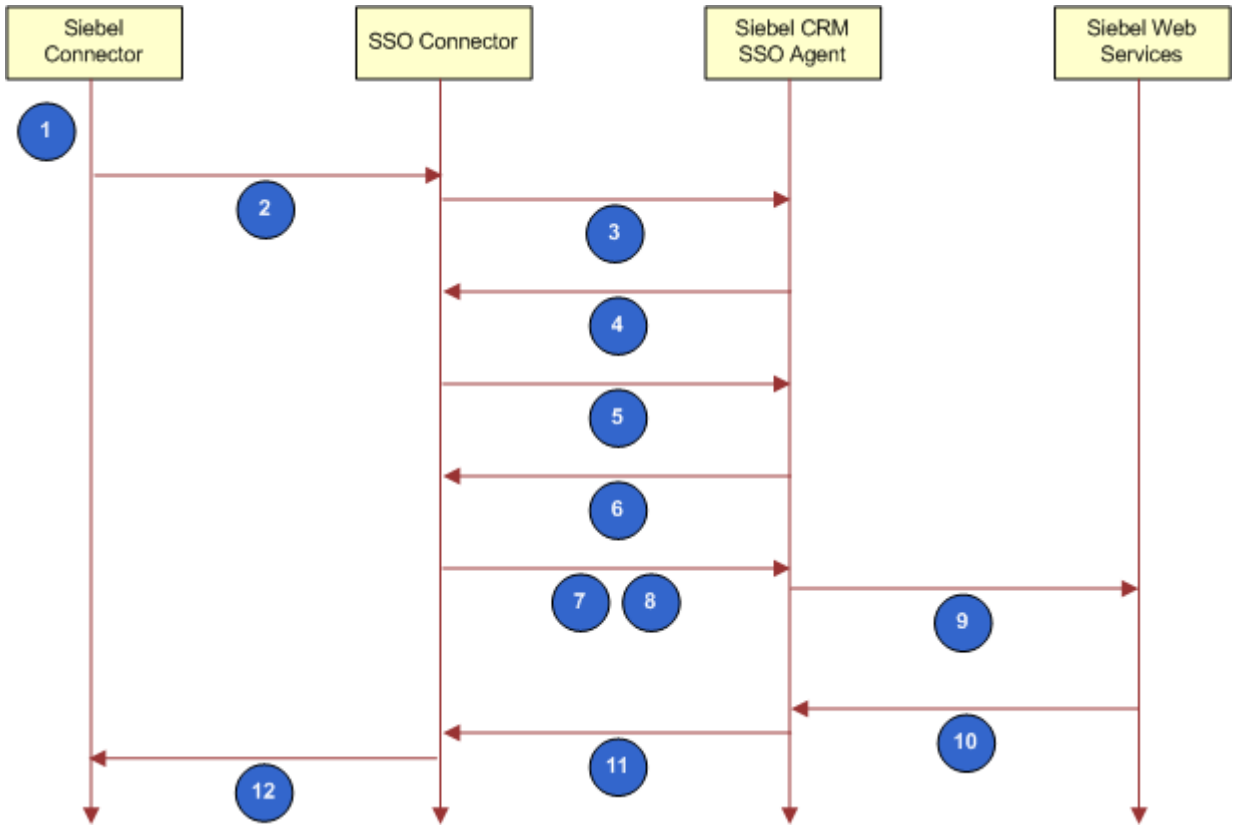


Figure 23. Flow That CRM Desktop SSO Uses During Authentication

Explanation of Callouts

The architecture for CRM Desktop SSO does the following during authentication:

1. User opens Outlook and then enters user name and password or uses credentials that CRM Desktop saved during a prior session.
2. Siebel Connector sends SOAP request with saved credentials to the SSO Connector. CRM Desktop SSO is a plug-in to CRM Desktop and acts as a local HTTP proxy. If the agent SessionID cookie that Siebel CRM SSO uses is set, then flow continues to Step 9.
3. SSO Connector attempts to send a request to the SSO Agent.
4. If the Agent SessionID cookie that Siebel CRM SSO uses is not set, or if this cookie is expired, then the SSO Agent sends a request for authentication in an HTTP redirect form or in an HTML form. This HTML form allows the user to reenter authentication information.
5. SSO Connector detects a request for authentication and then starts interactive or noninteractive authentication.

6 SSO connector sends HTTP request to the SSO Agent.

7 The SSO Agent sends a reply to the client and does something depending on the following authentication that CRM Desktop uses:

- **Interactive authentication.** The user must enter authentication information and then start the next step, for example, by clicking Login, and so forth.
- **Nonnteractive authentication.** The SSO Connector interprets the HTML reply.

Step 6 and Step 7 might repeat multiple times until authentication successfully finishes. CRM Desktop SSO considers this authentication successful if the HTTP can redirect to the original Siebel EAI address. When it meets this criteria is met, The SSO connector can use the session cookies when authentication successfully finishes.

8 The SSO Connector sends the original SOAP request and the session cookies to the SSO Agent.

9 The request now includes valid session information so the SSO Agent sends the original SOAP request to Siebel Web Services.

10 Siebel Web Services sends a reply to the SSO Agent.

11 The SSO Agent sends this reply to the SSO Connector. If the Siebel Server does not reply with HTTP 200 or HTTP 500, or if the reply does not include XML content, then the session is not valid and CRM Desktop SSO goes to Step 5. The presence of XML content indicates that the user has logged in into the native Web SSO that the browser uses.

12 The SSO Connector sends a reply to the Siebel Connector for processing. The SSO Connector can store an Agent SessionID cookie while Outlook runs. It can reuse this cookie in subsequent connection attempts. If this cookie expires, then Siebel CRM SSO requests the user to log in again.

Flow That the CRM Desktop SSO DLL Uses

Figure 24 illustrates the flow that the CRM Desktop SSO DLL uses.

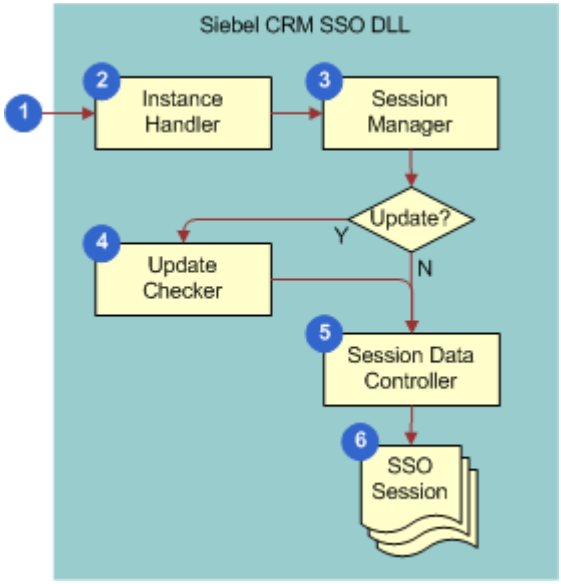


Figure 24. Flow That the CRM Desktop SSO DLL Uses

Explanation of Callouts

The CRM Desktop SSO DLL does the following:

1. CRM Desktop SSO loads the DLL.
2. The Instance Handler loads and initializes the DLL.
3. The Session Manager starts and maintains SSO sessions.
4. If you enable autoupdate, then the Update Checker automatically updates the script for each new session instance. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.
5. The Session Data Controller stores the names and values of parameters that the SSO sessions share. It allows data exchange between SSO sessions and provides a single location to store data that these sessions can share. For more information, see "About Authentication Sessions and Data Exchange Sessions" on page 387.
6. The SSO sessions are a collection of SSO sessions that are currently active. For more information, see "Architecture That an SSO Session Uses" on page 389.

About Authentication Sessions and Data Exchange Sessions

CRM Desktop SSO can create the following types of sessions:

- **Authentication session.** Starts if the user must change the login name and password or if CRM Desktop SSO requests the user to reenter the password to confirm these credentials. Note the following:
  - The SSO script does not use any cached information from a previous SSO session during an authentication session.
  - In some situations the SSO script cannot prevent Internet Explorer from allowing the user to access CRM Desktop without entering credentials. This situation typically occurs if CRM Desktop SSO uses a persistent cookie to identify the Web SSO user session. To allow the user to modify credentials when CRM Desktop SSO uses a persistent cookie, the user must use Internet Explorer to log out from CRM Desktop. This log out removes the persistent cookie. An authentication session prompts the user for a user name and password the next time the user attempts to connect to the Siebel Server from CRM Desktop.
- **Data exchange session.** Starts during a normal operation, such as synchronization, opening the Control Panel, and so on. CRM Desktop SSO can create multiple data exchange sessions. To avoid displaying unnecessary login prompts, the SSO Connector caches any session cookies that exist in the shared session cache or that reside in the cookie cache that Internet Explorer uses.

Architecture That an SSO Session Uses

Figure 25 illustrates the architecture that an SSO session uses.

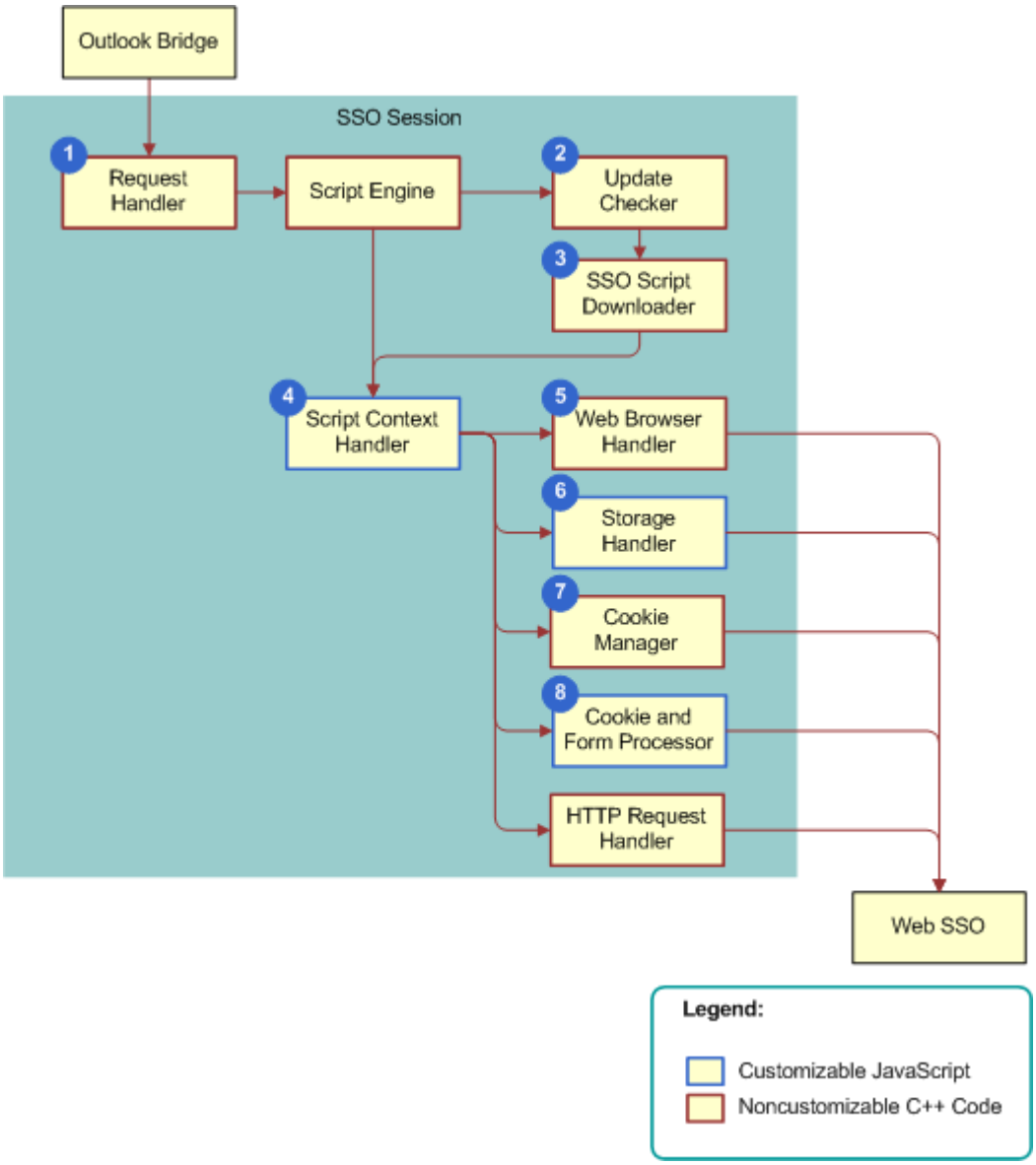


Figure 25. Architecture That an SSO Session Uses

Explanation of Callouts

The architecture that an SSO session uses includes the following items:

1. **Request handler.** Accepts each request from the Outlook Bridge, wraps the request in an object that an SSO script can access, routes the request through SSO script, and then sends the reply from the SSO to the client. It initializes the script, registers the request handler to handle connector requests and initializes the request handler. For example, it reads configuration settings, initializes global variables, and so forth. It runs the function that the request handler code registers for callback. For each incoming connector request, the SSO script establishes or reuses an SSO session with the Siebel Server and sends a reply to this server. For more information, see "Request Handler Function" on page 398.
2. **Update checker.** Calls the SSO script that runs autoupdate. It does this work the first time this session uses this script. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.
3. **SSO script downloader.** Downloads and prepares the SSO script.
4. **Script context handler.** Uses one instance of a Microsoft ActiveScript engine that runs SSO script and sends a reply to a request handler notification.
5. **Web Browser handler.** Displays an interactive login prompt to user, interprets the login information that this user enters, and notifies the SSO script.
6. **Storage handler.** Handles persistent and session storage.
7. **Cookie manager.** Gets and sets Internet Explorer cookies.
8. **Cookie and form processor.** Processes cookies, forms, and redirects. For more information, see "Cookie Handling" on page 391.

To customize the following items, you can reuse SSO objects or you can use your own set of common code. For more information, see "CRM Desktop SSO Objects You Can Customize" on page 394.

- Script context handler
- Storage handler
- Cookie and form processor

You cannot use JavaScript to customize items in Figure 25 that use C++ code, but you can change registry settings that affect these items.

SSO Script Lifecycle

If CRM Desktop SSO is enabled, and if the first connector starts, then CRM Desktop SSO loads the SSO module into Siebel CRM Desktop and it remains loaded until CRM Desktop closes.

The SSO script context includes all JavaScript global variables and state information. It is part of the SSO session data.The SSO Session Manager creates it and it exists until the SSO session ends.

Requests to start or end a session depend on the connector lifetime. CRM Desktop SSO starts a new session when it starts each new connector instance. If it ends a connector instance, then it also ends the SSO script context.

SSO Script Autoupdate

If SSO Script Autoupdate is enabled, then this Autoupdate determines if updated SSO script is available. If updated script is available, then CRM Desktop SSO loads this updated script instead of loading the old script. This configuration might result in the memory containing multiple versions of SSO script and SSO script context. When the connector sessions finish, CRM Desktop SSO unloads any old SSO script that exists and replaces it with the updated script.

Sharing Information Between Contexts

CRM Desktop SSO isolates script contexts and makes them independent from each other. To avoid unnecessary reauthentication, a script can handle different SSO sessions that share information. To do this, CRM Desktop SSO uses the settings_cache global object to read the configuration from one SSO session and reuse it or modify it in another SSO session.

SSO Script Operation

This topic describes SSO script operation.

Initialization

CRM Desktop SSO initializes SSO script when it creates a new SSO session. The initialization code must register a handler for the request_handler so that it handles connector requests and does the initialization that makes sure request handling is operational. For example, to set the read configuration settings, initialize global variables, and so forth.

Request Handling

To handle an SSO script request, CRM Desktop SSO runs a function for the request_handler callback. SSO script establishes or reuses an SSO session with the Siebel Server and returns a reply from this server for each incoming connector request.

Credentials Handling

CRM Desktop SSO handles credentials in one of the following ways:

- **Noninteractive authentication.** Sets user credentials in the CRM Desktop login dialog box and then communicates them to the SSO script through the get_sso_username function and the get_sso_password function of the sso_client global object.
- **Interactive authentication.** Does not send user credentials to SSO script. This SSO script must make sure that CRM Desktop allows the user to authenticate and that the authentication session runs correctly. It must use the ia_state object to capture cookie information and then use this information in the request and reply with the Siebel Server.

Cookie Handling

CRM Desktop SSO uses the WinHTTP protocol to support cookie handling. For more information, see the topic about Manual and Automatic Cookie Handling in the Cookie Handling in WinHTTP topic in the Dev Center - Desktop section of the Microsoft Developer Network website.

The execute_request call returns cookies that the Siebel Server sets as part of the HTTP handling. WinHTTP interprets this call and adds it to the cookie cache that CRM Desktop SSO reuses during subsequent requests. The client can also specify cookies and then add them to a request. Interactive authentication requires special handling of cookies. Noninteractive authentication uses WinHTTP while interactive authentication uses Internet Explorer. CRM Desktop SSO sends all required cookies from the script session to the Internet Explorer session before it starts an interactive authentication. CRM Desktop sends these cookies back to the WinHTTP noninteractive session after interactive authentication finishes.

How CRM Desktop SSO Handles Errors

SSO script avoids creating JavaScript exceptions. If the user enables a JIT Debugger on the client computer, such as Visual Studio, then this debugger might display a separate prompt and debug message for each exception. These prompts can significantly affect the user experience. Siebel CRM SSO uses the following functions to handle exceptions:

- cpp_exception_occurred
- drop_exception
- raise_not_logged_in_exception
- raise_not_valid_exception
- raise_cancel_exception
- execute_request

The execute_request function is the only function that can fail with exception. It depends on the network state and the availability of Siebel Servers. If execute_request fails, then Siebel CRM SSO returns null instead of a response object to handle logic that does not include exceptions. The SSO Connector must examine the return code, determine if an error occurred, and then take corrective action.

If an exception occurs, then the SSO script does one of the following:

- **Pass error to Siebel Connector.** Recreates the exception when flow returns to the Siebel Connector so that this connector can handle the exception. The SSO Connector script must finish running without calling any other execute_request function or calling a function that modifies the exception, such as a raise_*value*_ exception or drop_exception. It does this to avoid overwriting the original exception message that the script created as a result of the error.
- **Clear the exception.** Use the drop_exception function to clear the exception.
- **Run another request.** Use the other execute_request function to clear the previous exception.
- **Create an exception:**
  - Use the raise_not_valid_exception function. In this situation, the user provided the credentials in the client but they are not valid.
  - Use the raise_not_logged_in_exception function. In this situation, the credentials are not complete or are missing.

For more information about these methods, see “SSO Client Object” on page 394.

How the SSO Script Sends an Error to the Connector

The following type of error determines how the SSO script sends an error to the connector:

- **Network error.** If the execute_request function returns a value of Null, then the SSO script sends the error back to the connector and the connector processes the error.
- **Authentication or SSO script logic error.** The SSO script translates the error to a not_valid or a not_logged_in error so that the corresponding function in the sso_client object can handle the error. It uses the following functions:
  - Uses the raise_not_logged_in_exception function for a not_logged_in error
  - Uses the raise_not_valid function for a not_valid error
- **User cancelled interactive authentication.** The SSO script runs the raise_cancel_exception function to start a cancel_exception exception.

How the SSO Script Logs Errors and Messages

Siebel CRM SSO writes log messages to the CRM Desktop general log which simplifies debugging and

and gathering diagnostics. The SSO script logs information about the SSO API and application failures. For more information, see "Logger Object" on page 400.

Modifying SSO JavaScript

You can use the API (application programming interface) that comes predefined with CRM Desktop SSO (SSO API) to create a custom script that does the following:

- Get HTML and XML pages
- Submit forms
- Follow redirects
- Support cookies

CRM Desktop SSO uses JavaScript to implement the logic it requires. This script can accommodate some layout or workflow changes but it might be necessary to modify it to meet your deployment requirements. CRM Desktop SSO comes predefined with the following JavaScript files:

- **core.js and utils.js.** A set of reusable functions and building blocks that handle authentication. The lib.js file includes detailed documentation in the source code that describes the API that you can use to customize CRM Desktop SSO.
- **sso.js.** An entry point that handles authentication.

You must distribute any update you make by uploading the new script you create to an autoupdate location or you can use external provisioning. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.

CRM Desktop SSO Objects You Can Customize

This topic describes the objects that you can use to customize CRM Desktop SSO. It includes the following topics:

SSO Client Object

This topic describes functions that you can use with the SSO client object. It includes the following topics:

To communicate the credentials that the user enters in the CRM Desktop login dialog box to the SSO script, CRM Desktop SSO uses the get_sso_username and get_sso_password functions of the sso_client object.

The sso_client object is a global object that is available anywhere in the main script file. The following registry key identifies this main script file:

```
SSO\ScriptFileName
```

For more information, see "Windows Registry Keys You Must Set to Enable CRM Desktop SSO" on page 416.

Create Request Function

The create_request function creates a request and returns a new request object. It uses the following format:

```
create_request(url, method, body, encoding)
```

where:

- *url* is a string that contains the URL that identifies the request endpoint.
- *method* is a string that identifies the HTTP request method. It can include one of the following values:
  - GET
  - POST
- *body* is a string that contains the body of the request.
- *encoding* is a string that identifies how to encode the request body. It can include one of the following values:
  - iso-8859-1
  - utf-8
  - utf-16

For example, the following code creates a simple GET request to a URL:

```
var req = sso_client.create_request("http:\\siebel\eai_enu", "GET", "", "utf-8");
```

For more information, see "Request Object" on page 401.

Create Response Function

The create_response function creates and returns a new response object. It uses the following format:

```
create_response(http_code, http_status, body, encoding)
```

where:

- *http_code* is an integer that identifies an HTTP status code. For example, 200, 500, or 404. For more information, see the topic about status code definitions at the World Wide Web Consortium (W3C) website.
- *http_status* includes a string that identifies an HTTP status message that corresponds to the value that http_code identifies. For example, OK for 200, Not found for 404, and so forth.

- *body* includes a string that is the response body.
- *encoding* includes a string that identifies how to encode the response body. It can include one of the following values:
  - iso-8859-1
  - utf-8
  - utf-16

  You must use double quotes to enclose each value. For example:

```
var resp_body = "… response body …";
var resp = sso_client.create_response(500, "Internal Server Error", resp_body,
"UTF-8");
  resp.get_headers().add_header("Content-Type", "text/xml; charset=utf-8");
  resp.get_headers().add_header("Content-Length", ""+resp_body.length);
```

Decode URL Function

The decode_url function decodes a URL. It does the following:

1. Splits a URL into components and then reconstructs this URL. It adds any missing components.
2. Replaces encoded sequences into their corresponding values. The following format identifies a sequence:

   %xx

   where:

   - *%* represents percent encoding that allows you to encode information in a Uniform Resource Identifier (URI).

The decode_url function uses the following format:

```
decode_url(url)
```

where:

- *url* is a string that contains the URL that the decode_url function decodes.

Encode URL Function

The encode_url function encodes a URL. It replaces each nonstandard character with the encoded value. The following format identifies a sequence:

%xx

The encode_url function uses the following format:

```
encode_url(url)
```

where:

- *url* is a string that contains the URL that the encode_url function encodes.

Exception Occurred Function

The cpp_exception_occurred function is a Boolean function that returns True if the last call to an execute_request function resulted in C++ code creating an exception. Any subsequent call to the execte_request function resets a previously recorded exception, so you must use this function immediately after the call to the execute_request function.

Get Platform Cookie Function

The get_platform_cookie function gets the Internet Explorer cookie. It can only get Internet Explorer cookies that are persistent and that are not of type HTTPOnly. It uses the following format:

```
get_platform_cookie(url, name)
```

where:

- *url* is a string that contains the URL that CRM Desktop SSO uses to get the cookie.
- *name* is a string that contains the name of the cookie.

Interactive Function

The interactive function instructs the SSO script to use interactive authentication. It uses callbacks to monitor the state of the interactive session. CRM Desktop SSO runs the statement that occurs immediately after the statement that calls the interactive function only after the interactive authentication finishes. While interactive authentication runs, the SSO script gets notifications from the function that the callback parameter identifies. This callback delivers the information that this script requires to monitor the interactive authentication. It can send a request to stop the interactive authentication when the interactive session finishes.

The interactive function uses the following format:

```
interactive(ia_state, callback)
```

where:

- *ia_state* is a string that identifies the object that contains information about interactive authentication.
- *callback* is a string that identifies the function that CRM Desktop SSO calls on every change that occurs in the ia_state.status. This callback must return a Boolean value. If it returns True, then CRM Desktop SSO stops the interactive authentication. *ia_state* is an object that the sso_client.create_ia_state function creates and then uses to control how interactive authentication runs. The callback is a function that returns one of the following values:
  - **true.** Interactive authentication finished.
  - **false.** Interactive authentication has not yet finished.

The following example does a simple callback for interactive authentication. It does not do any processing. It always return false to indicate that CRM Desktop SSO must display the native browser dialog box in every subsequent web page where the user navigates:

```
function ia_callback(ia_state)
{
  switch (ia_state.status) {
    case "before":
    // before navigate to url
    break;

    case "finished":
    // page download complete
    break;

    case "cancelled":
    // user closed dialog
    break;
}
  return false;
}
```

This example includes the following code. This code is part of the request handler function:

```
var ia_state = sso_client.create_ia_state();
ia_state.url = url;
ia_state.dialog.width = 1024;
ia_state.dialog.height = 768;
ia_state.dialog.title = "SSO";
ia_state.dialog.visible = false;
sso_client.interactive (ia_state, ia_callback);
```

Request Handler Function

CRM Desktop SSO routes each request from the Siebel Connector through the request_handler function. This function handles requests that occur from the Siebel Connector to the Siebel Server. This request_handler function expects a single argument that contains the initial request object from the SSO Connector. It sends a return value to the SSO Connector as if the Siebel Server sent this reply. The Siebel Connector is a proxy that resides between the Siebel Connector and Siebel Web Services. The SSO Agent protects these Web services. This proxy hides the SSO Agent from the Siebel Connector.

The following example includes a CRM Desktop SSO script that passes all requests from the Siebel Connector to the Siebel Server without doing any processing. This example is for illustration purposes only. An actual SSO Connector script includes the process_request function to do processing:

```
sso_client.request_handler = process_request;
function process_request(request)
{
return sso_client.execute_request(request);
}
```

The following line from this code defines the entry point where CRM Desktop SSO registers the handler:

```
sso_client.request_handler = process_request
```

Set Platform Cookie Function

The set_platform_cookie function sets the Internet Explorer cookie. It can only get Internet Explorer cookies that are persistent and that are not of type HTTPOnly. It uses the following format:

```
get_platform_cookie(url, name, value)
```

where:

- *url* is a string that contains the URL that CRM Desktop SSO uses to set the cookie.
- *name* is a string that contains the name of the cookie.
- *value* is a string that contains the data that CRM Desktop SSO associates with the cookie.

For example:

```
sso_client.set_platform_cookie("http:\\some.site.com", "Cookie_name",
"Cookie_value");
```

Other Functions That the SSO Client Object Includes

You can use the following functions in the SSO Client object. CRM Desktop supports each of these functions starting with Siebel CRM Desktop version 3.7 except for the execute_request function. Support for the execute_request function starts with an earlier release:

- **create_ia_state.** Creates and returns a new ia_state object. CRM Desktop SSO uses this object for initial configuration with interactive authentication to send status information. ia_state is an object, while create_ia_state is a function that creates the ia_state object. For more information, see “Interactive State Object” on page 405.
- **execute_request**. Runs the request and returns a reply. It accepts a request from the request_handler function or a request that the create_request function creates. It uses the following format:

  ```
  execute_request(request_object)
  ```

- **get_sso_username.** Gets the user name that the user provides in noninteractive authentication. CRM Desktop SSO does not use this function for interactive authentication.
- **get_sso_password.** Gets the password that the user provides in noninteractive authentication. CRM Desktop SSO does not use this function for interactive authentication.
- **drop_exception.** Drops any exceptions that the connector reports. If the script finishes running after CRM Desktop SSO calls this method, then CRM Desktop SSO does not create an exception.
- **raise_not_logged_in_exception.** Drops any not_logged_in exceptions that the connector reports and then creates a not_logged_in exception with a description that states it cannot process the login because a problem occurred with the Siebel Server or credentials are not valid. CRM Desktop SSO does not stop the session flow. The SSO script must finish running correctly before the connector creates the exception.
- **raise_not_valid_exception.** Drops any not_valid exceptions that the connector reports and creates a not_valid exception with a description that states the user password is not valid or is missing. CRM Desktop SSO does not stop the session flow. The SSO script must finish running correctly before the connector creates the exception.

- **raise_cancel_exception.** Clears any reported exception. It creates the following exception for the SSO Connector:

  ```
  cancelled
  ```

  CRM Desktop SSO does not stop the session flow. The SSO script must finish running the request_handler after it creates an exception.

Logger Object

The logger object is a global object that is available anywhere in the main script file. It allows CRM Desktop SSO to log messages to a general log. It uses the following format:

```
log(component,message,class)
```

where:

- *component* is a string that identifies the source of the log message
- *message* is a string that identifies the message that CRM Desktop SSO logs.
- *class* is a string that identifies the error class.

For example:

```
logger.log ("SSO", "Sample message", 0);
```

Settings Cache Object

The settings_cache object is a global object that is available anywhere in the main script file. It is a shared name-value cache. Multiple SSO script instances that run in different threads of a single process can reuse this cache to get session information.

It includes the following function that sets the value of a cache setting:

```
set(name, value)
```

where:

- *name* is a string that identifies the setting name
- *value* is a string that identifies the setting value

It includes the following function that gets the value of a cache setting:

```
get(name)
```

where:

- *name* is a string that identifies the setting name

For example:

- The following code stores a value that CRM Desktop SSO gets in another instance of the SSO script:

```
settings_cache.set("MyKey", "MyValue");
```

- The following code gets the stored value. If no value exists, then this code returns an empty string:

```
var v = settings_cache.get("MyKey");
```

Settings Object

The settings object is a global object that is available anywhere in the main script file. It gets the value of a key that CRM Desktop SSO maps from the product key in the Windows Registry. It uses the following format:

`get(`*`name`*`)`

where:

- *name* is a string that identifies the name of a registry key

Request Object

The request object represents the initial HTTP request that the connector sends to the SSO script. You can also use it to allow the SSO script to create an instance of a request object that establishes an SSO session for the connector, as necessary. It includes the following functions:

- **get_headers.** Gets the object that represents the HTTP request header. For more information, see "Header Object" on page 403.
- **get_server.** Gets the host part of the request URL. For example, `http://server.com/`.
- **get_object.** Gets the path part of the request URL. For example, `path/page.asp`.
- **get_method.** Gets the HTTP method. For example, GET, POST, and so forth.
- **get_contents.** Gets the content object that contains the request body and that CRM Desktop SSO can use to get the request in plain text. For more information, see "Content Object" on page 402.
- **get_credentials.** Gets the credentials object that contains the security credentials for the HTTP request. These credentials include the login name, password, and other authentication information. For more information, see "Credentials Object" on page 404.

Response Object

The response object contains the HTTP response that CRM Desktop SSO gets from the Siebel Server. It includes the properties that CRM Desktop SSO requires to retrieve the response status, headers, and the body. It includes the following functions:

- **get_status_code.** Gets the HTTP status code for the current response.

- **get_headers.** Gets the header object that contains the HTTP headers for the response. For more information, see "Header Object" on page 403.
- **get_contents.** Gets the content object that contains the response body and that CRM Desktop SSO can use to get the response in plain text. For more information, see "Content Object" on page 402.

Content Object

The content object gets the string content of a request or a response. It includes the following function that gets the text content of a request or response:

```
get_text(encoding)
```

where:

- *encoding* is a string that identifies how the body of the request or response is encoded. It can include one of the following strings:
  - iso-8859-1
  - utf-8
  - utf-16

  For example:

  ```
  get_text(iso-8859-1)
  ```

It includes the following function that sets the text content of a request or response:

```
set_text(text, encoding)
```

where:

- *text* is the text of the body of the request or response.
- *encoding* is a string that identifies how CRM Desktop SSO must encode the body of the request or response. It can include one of the following strings:
  - iso-8859-1
  - utf-8
  - utf-16

  For example:

  ```
  set_text("My text string", "iso-8859-1");
  ```

You must enclose each string in JavaScript quotes, so the following code returns the body of the request converted from utf-8:

```
var body = req.get_contents().get_text("utf-8");
```

CRM Desktop SSO does not allow you to modify the text of the initial request that comes as a parameter of the request_handler function. For more information, see "Request Handler Function" on page 398.

For more information, see "Request Object" on page 401 and "Response Object" on page 401.

Header Object

The header object includes the HTTP headers of a request or response. It is read-only for a response and writable for a request. It includes the following functions:

- **get_header_count.** Gets the number of headers that the request or response contains.
- **get_string_value.** Gets all headers as a single string.

Get Header Name Function

The get_header_name function gets the name of a header. It uses the following format:

```
get_header_name(index)
```

where:

- *index* is a numeric value that identifies the header index. This index starts with 0 as the first value. To access the first header, you use an index value of 0. To access the second header, you use an index value of 1, and so forth.

For example, the following code gets the name of the first header:

```
var header = req.get_headers().get_header_name(0);
```

Get Header Value Function

The get_header_value function gets the value of a header. It uses the following format:

```
get_header_value(index)
```

where:

- *index* is a numeric value that identifies the header index. This index starts with 0 as the first value. To access the first header, you use an index value of 0. To access the second header, you use an index value of 1, and so forth.

For example, the following code gets the value of the first header:

```
var header_val = req.get_headers().get_header_value(0);.
```

Add Header Function

The add_header function adds a new header. You can use it only for a request. You cannot use it for a reply. It uses the following format:

```
add_header(name, value)
```

where:

- *name* is a string that contains the header name.
- *value* is a string that contains the header value.

For example, the following code adds the content type for the request:

```
req.get_headers().add_header("Content-Type", "text/html");
```

For more information, see "Request Object" on page 401 and "Response Object" on page 401.

Credentials Object

The credentials object is a subobject of a request object that specifies the HTTP authorization parameters that CRM Desktop SSO uses. It includes the following functions:

- **get_username.** Gets the user name associated with a request.
- **get_password.** Gets the password associated with a request.
- **get_auth_schemes.** Gets the list of authorization schemes that this request supports, in order of preference. For a list of schemes, see "Set Authorization Schemes Function" on page 404.

Set User Name Function

The set_username function sets the user name for a request. It uses the following format:

```
set_username(username)
```

where:

- *username* is a string that contains the user name.

Set Password Function

The set_password function sets the password for a request. It uses the following format:

```
set_password(password)
```

where:

- *password* is a string that contains the password.

Set Authorization Schemes Function

The set_auth_schemes function sets the authorization schemes that CRM Desktop SSO uses with a request. It uses the following format:

```
set_auth_schemes(auth_schemes)
```

where:

- *auth_schemes* is a string that contains the authorization schemes. It can contain any combination of the following values:
  - **B, b.** Specifies to use basic authorization.
  - **N, n.** Specifies to use NTLM (NT LAN Manager) authorization.
  - **P, p.** Specifies to use passport authorization.
  - **D, d.** Specifies to use digest authorization.

You must use the following format:

- The order that you use is important. For example, if you specify NB, then this request only supports NTLM or basic authentication and it uses NTLM first, if possible.
- JavaScript requires that you use double quotes to enclose each string.
- Values are not case-sensitive.

For example, the following code configures the HTTP client to use NTLM basic authentication if the direct request fails, where NTLM authentication takes priority over basic authentication:

```
req.get_credentials().set_auth_schemes("nb");
```

For information about basic authorization and passport authorization, see the topic about HTTP authentication schemes at the Microsoft Developer Network website.

Interactive State Object

The interactive state object contains information about the state of the interactive authentication. The create_ia_state function of the sso_client object creates the interactive state object so that the interactive function of the sso_client object can use it when it calls the SSO script. CRM Desktop SSO uses the same interactive state object to do a callback from the code that handles the interactive authentication to the SSO script. For an example that uses a callback, see "Interactive Function" on page 397.

The ia_state object includes the following properties:

- **url.** A string that identifies the URL of the page that Internet Explorer displays when Siebel CRM SSO runs the following function:

  ```
  interactive()
  ```

  CRM Desktop SSO provides the current URL that the Web browser object processes. It does this during the callback. CRM Desktop SSO uses ia_state later as a callback to JavaScript, and the URL contains the current URL of the page that the browser control processes. For an example of this usage, see "Example Code That Customizes CRM Desktop SSO" on page 406.

- **cookies.** A string that identifies the cookies that are associated with the HTML page that CRM Desktop SSO loads into the control Internet Explorer.
- **headers.** A string that identifies any other HTTP headers that the Browser sends.

- **post_data.** A string that identifies the data that CRM Desktop SSO sends through an HTTP POST function to the Siebel Server. This property exists on form submission. CRM Desktop SSO uses HTML forms to get user information on the Web page and to send this information to the Siebel Server for processing. For example, to get the login name and password that the user enters during login.
- **html_body.** A string that identifies the HTML body of the document that CRM Desktop loads in Internet Explorer.
- **status.** A string that indicates the type of callback that the interactive authentication handler uses. It can include one of the following values:
  - **before.** CRM Desktop SSO sends the callback before it navigates the Web browser to a URL.
  - **finished.** CRM Desktop SSO sends the callback after the page download finishes.
  - **cancelled.** CRM Desktop SSO sends the callback if the user cancels the login and closes the dialog box.

  JavaScript requires that you enclose each string in double quotes.
- **title.** A string that contains the title of the dialog box. CRM Desktop SSO displays this title in the dialog box during interactive authentication. The default configuration that the SSO Connector uses adds the following prefix to any page title that the Siebel Server renders:

  `SSO:`
- **dialog.** A string that identifies the dialog object. For more information, see "Dialog Object" on page 406.

Dialog Object

The dialog object is a subobject of the ia_state object. It defines the size, position, visibility, and title of the SSO login dialog box that CRM Desktop SSO displays during interactive authentication. It includes the following properties:

- **width.** A number that determines the width of the dialog box, in pixels.
- **height.** A number that determines the height of the dialog box, in pixels.
- **visible.** A Boolean value. If True, then CRM Desktop SSO displays the dialog box.
- **title.** A string that contains the title of the dialog box. CRM Desktop SSO displays this title in the dialog box during interactive authentication.

Example Code That Customizes CRM Desktop SSO

The following code comes predefined with CRM Desktop:

```
include("utils.js", "_utils");
include("core.js", "_core");
_utils.logger = logger;
```

```
_utils.sso_client = sso_client;
_core.sso_client = sso_client;
_core.settings_cache = settings_cache;

var Utils = _utils.Utils;
var Lib = _core.Lib;
var CookieManager = new Lib.CookieManager();
var SSOConfiguration = {
  "CookieBuffer": "DomainCookies",
  "AuthType": settings.get("AuthType"),
  "EndpointRegExp": settings.get("EndpointRegExp"),
  "SuccessLoginRegExp": settings.get("SuccessLoginRegExp")
};
var interactive_params = {
  "InitialWidth": 1024,
  "InitialHeight": 768,
  "InitialTitle": "SSO",
  "InitialTitlePrefix": "SSO",
  "CheckFn": InteractiveCheckFunction
};

var persistent_cookies = true;

sso_client.request_handler = process_request;

function InteractiveCheckFunction (ia_state, ia_result, original_request) {
  var path = (original_request.get_object()).replace(/\//i,"").split("?");

  if (SSOConfiguration.EndpointRegExp == "") {
    is_original_url = ( (ia_state.url).toLowerCase()
).indexOf((original_request.get_server() + path[0]).toLowerCase() ) != -1;
  } else {
  is_original_url = (new RegExp(SSOConfiguration.EndpointRegExp,
'i')).test(ia_state.url)
  }
  if (!is_original_url && ia_state.status == "before") {
  persistent_cookies = false;
  }
  if (ia_state.status == "finished" && ia_state.html_body != "") {
    if (is_original_url) {
      var regexp = SSOConfiguration.SuccessLoginRegExp == "" ? "FAULTSTRING.
*?10944629" : SSOConfiguration.SuccessLoginRegExp;

if ( ia_state.html_body.match( new RegExp(regexp,'mi') ) != null) {
return true;
}
}
}
return false;
}
function RequestData(request, clear_session) {
if (GetCache() !== "" && CookieManager.GetAllCookies().length == 0) {
Utils.SetRequestCookies(request, GetCache());
}
var response = Lib.ExecuteRequest(request, CookieManager, clear_session);
```

```
if (CookieManager.GetAllCookies().length > 0) {
UpdateCache(CookieManager.GetAllCookiesAsString());
}
return response;
}
function RedefineInteractiveDescriptor (response, redefine_location) {
return function (descriptor, ia_state) {
Utils.Log("Clear browser cookies", "info");
var cookies = Utils.ParseCookieString(ia_state.cookies);
var cookie = {};
for (var i = 0, len = cookies.length; i < len; i++) {
cookie = Utils.ParseCookie(cookies[i]);
ia_state.cookies = cookie.Name + "=;";
}
if (response !== null) {
if (redefine_location) {
descriptor.SetEndpoint(Utils.GetSpecificHeader(response.get_headers(),
"Location")[0]);
}
}
return descriptor;
}
}
function UpdateCache (value) {
Utils.Log("Update cache cookies", "info");
var cached = Utils.ParseCookies(Utils.ParseCookieString(GetCache()));
var browser = Utils.ParseCookies(Utils.ParseCookieString(value));
for (var i = 0, iLen = browser.length; i < iLen; i++) {
isset = false;
for (var j = 0, jLen = cached.length; j < jLen; j++) {

if (browser[i].Name == cached[j].Name) {
isset = true;
cached[j] = browser[i];
}
}
if(!isset) {
cached.push(browser[i]);
}
}
settings_cache.set(SSOConfiguration.CookieBuffer,CookieManager.ConvertCookiesToString(
cached));
}
function ClearCache () {
settings_cache.set(SSOConfiguration.CookieBuffer, "");
}
function GetCache() {
return settings_cache.get(SSOConfiguration.CookieBuffer);
}
function process_request(sso_client_request) {
var ignore_cache = settings.get("IgnoreCache");
var response;
if (SSOConfiguration.AuthType == "NTLM") {
var creds = sso_client_request.get_credentials();
```

```
creds.set_username('');
creds.set_password('');
creds.set_auth_schemes('n');
response = sso_client.execute_request(sso_client_request);
if (response == null) Utils.Log("No response received", "warning");
} else {
var request_x_type = Utils.GetSpecific-
Header(sso_client_request.get_headers(), "X-CRMD-TYPE");
if (request_x_type == "verify" && ignore_cache == "1") {
ClearCache();
}
response = RequestData(sso_client_request, false);
var status_code = response.get_status_code();
if (request_x_type == "logout") {
response = null;
} else if (status_code == "401" || status_code == "407") {
Utils.Throw("not_valid", "script_not_valid_sso_ntlm_attempt");

Utils.Log("Attempt to use SSO mode with NTLM-protected EAI", "info");
return null;
} else if (status_code == "302" || Utils.Transitions.IsHtml(response)) {
var redefine = status_code == "302" ? true : false;
Utils.Log("Interactive mode initialized", "info");
var result = Lib.RunInteractive(sso_client_request, interactive_params,
CookieManager, RedefineInteractiveDescriptor(response, redefine));
if (result == "success") {
if (persistent_cookies == true && ignore_cache == "1" && settings.
get("UserChanged") == "1") {
Utils.Throw("not_valid", "script_not_valid_clear_cookies");
Utils.Log("Persistent cookies exist", "info");
return null;
} else {
persistent_cookies = true;
Utils.Log("Login successful", "info");
}
} else if (result == "canceled") {
Utils.Log("Login canceled", "info");
sso_client.raise_cancel_exception("User canceled login dialog.");
return null;
}
Utils.Log("Interactive mode finished", "info");
response = RequestData(Utils.CloneRequest(sso_client,
sso_client_request, null), true);
}
}
return response;
}
```

A Reference Information for Siebel CRM Desktop

This appendix describes reference information for Siebel CRM Desktop. It includes the following topics:

Registry Keys You Can Use with Siebel CRM Desktop

This topic describes Windows Registry keys you can use with Siebel CRM Desktop. It includes the following topics:

CAUTION: Modifying the Windows Registry can cause serious and permanent problems that you might not be able to resolve. You must be very careful to make only the modifications you require, and that the modifications you make do not negatively affect functionality or performance.

For more information, see “Using the Windows Registry to Control Siebel CRM Desktop” on page 105.

Registry Keys That Affect Siebel CRM Desktop Behavior

Table 26 describes the Windows Registry keys that you can modify to change Siebel CRM Desktop behavior. In the Registry Editor (regedit), you can modify these keys in the following path:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop
```

Table 26. Windows Registry Keys That Affect Behavior of the CRM Desktop Add-In

Windows Registry Key	Description
AppLanguageID	ID of the current installation package of Outlook.
customization_path	Path to the customization package files.
DestinationProfile	Name of the Outlook profile where the add-in is installed.
DestinationStore	Name of the Outlook profile where the add-in is installed.
DisableLiveUpdate	Specifies if the live update feature is allowed. The following values are valid: ■ **0.** Live update is allowed. ■ **1.** Live update is not allowed.
DisableSyncConfirmation	Suppresses confirmation for deleting an object. The following values are valid: ■ **1.** Suppress confirmation for deleting an object. ■ **0.** Do not suppress confirmation for deleting an object. The corresponding attribute in the connector_configuration.xml file of the customization package automatically overwrites the DisableSyncConfirmation key.
FiltersEstimateOnTimer	Sets the interval in milliseconds for an automatic estimation of records after the filters are changed in the control panel. The following values are valid: ■ **1.** Estimate automatically. ■ **0.** Do not estimate automatically. This entry is not accessible through the administrative interface.
HTTPClient:AcceptCompression	A flag that instructs the Web Service Connector to accept zipped HTTP content. This key is not accessible through the administrative interface.
HTTPClient:CompressOutgoing	A flag that instructs the Web Service Connector to send zipped HTTP content. This key is not accessible through the administrative interface.
HTTPClient:ConnectTimeout	The timeout for the connection in milliseconds.
HTTPClient:ReceiveTimeout	The timeout for the receiving requests in milliseconds.

Table 26. Windows Registry Keys That Affect Behavior of the CRM Desktop Add-In

Windows Registry Key	Description
HTTPClient:SendRetryCount	The count for the connection retries in milliseconds.
HTTPClient:SendTimeout	The timeout for the sending requests in milliseconds.
LogoutTimeout	The time to wait in milliseconds after CRM Desktop sends the log out request. This parameter stops the session without waiting for the reply from the server.
Page:Feedback:AttachLog	Specifies a log for the feedback form. The following values are valid: ■ **0.** Do not attach a log to the feedback form. ■ **1.** Attach a log to the feedback form.
ProxyLogin	The login for the proxy server.
ProxyPassword	The password for the proxy server.
ProxyServer	The host name for the proxy server.
ProxyServerPort	The port number for the proxy server.
ProxyUsage	Flag that specifies a proxy server. The following values are valid: ■ **0.** Do not use a proxy server. ■ **1.** Use a proxy server.
RunPeriodicalSyncAlways	Determines if CRM Desktop starts a scheduled synchronization at the scheduled time or waits until Outlook is idle. The following values are available: ■ **0.** Wait until Outlook is idle to start the scheduled synchronization. ■ **1.** Start the scheduled synchronization immediately when the synchronization is scheduled to occur. The value is 1.
SessionsKeepAliveAmount	Defines the number of synchronization sessions to store in the internal database as history. CRM Desktop stores statistical information for each synchronization session. To view information about synchronization issues, the user can use the list control in the Sync Issues tab of the Synchronization Control Panel.

Table 26. Windows Registry Keys That Affect Behavior of the CRM Desktop Add-In

Windows Registry Key	Description
SharedByDefault:NewItems	Determines how CRM Desktop shares newly created Outlook items. The SharedByDefault:NewItems registry key controls the following option: Always share with Siebel new: Calendar Appointment, Contacts, tasks To access this option, the user right-clicks the CRM Desktop icon in the system tray, chooses Options, and then clicks the Advanced tab in the CRM Desktop - Options dialog box.
Siebel:HideSavePasswordOption	Determines how CRM Desktop displays the Save Password check box on the login screen. The following values are valid: ■ **0.** Display the Save Password check box. CRM Desktop displays this check box, by default. ■ **1.** Disable the Save Password check box.
Siebel:MetaInfoFilePath	Describes the path to the siebel_meta_info.xml file in the customization package.
SuppressSyncEstimating	Estimates the number of objects that CRM Desktop will synchronize for the current user. The following values are valid: ■ **0.** Do estimate the number of the objects. ■ **1.** Do not estimate the number of the objects. If SuppressSyncEstimating is 1, then CRM Desktop does not use the logic that the MaximumSyncPassthrough key controls. It uses MaximumSyncPassthrough only after it estimates the number objects it must synchronize.
SuppressSyncIssues	If a synchronization problem occurs, then this key determines if CRM Desktop changes the application icon in the system tray to an exclamation point and displays a message. The following values are valid: ■ **0.** Change the icon in the system tray and display a message. The default value is 0. ■ **1.** Do not change the icon in the system tray and do not display a message.

Registry Keys That Affect Credentials

Table 27 describes the Windows Registry keys that you can modify to change how Siebel CRM Desktop handles credentials. In the Registry Editor (regedit), you can modify these keys in the following path:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\Credentials
```

Table 27. Windows Registry Keys That Affect Credentials

Windows Registry Key	Description
RememberPassword	Determines how CRM Desktop remembers the password. You can use one of the following values: ■ **0.** Remember the password for all Outlook sessions and add a check mark to the Save Password check box in the CRM Desktop - Login dialog box. ■ **1.** Remember the password only for the current Outlook session and remove the check mark from the Save Password check box in the CRM Desktop - Login dialog box.
Siebel:ComponentName	Siebel Server component name that processes incoming requests. The following value is the default value: `eai/enu` The Siebel:ComponentName key is appended to the URL. For more information, see "Overriding Windows Registry Keys That Locate the Siebel Server" on page 106.
Siebel:LoginName	The name of the CRM Desktop user who is currently logged in.
Siebel:Password	Stores the user password in binary format.
Siebel:Protocol	Defines the URL protocol. The default is http. The following values are valid: ■ http ■ https
Siebel:RequestSuffix	Sets the suffix of the URL to the Siebel Server. The following is the default value: `?SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1` For more information, see "Overriding Windows Registry Keys That Locate the Siebel Server" on page 106.

Table 27. Windows Registry Keys That Affect Credentials

Windows Registry Key	Description
Siebel:Server	Sets the host name of the URL. The default value is *empty*.
Siebel:ServerPort	Sets the port of the URL. The default value is 80.

Registry Keys That Affect CRM Desktop SSO

This topic describes the Windows registry keys that affect CRM Desktop SSO. It includes the following topics:

- "Windows Registry Keys You Must Set to Enable CRM Desktop SSO" on page 416
- "Registry Keys That Control SSO for Siebel CRM Desktop" on page 419
- "Registry Keys That Control SSO for Credentials" on page 420

You must not modify the following Windows registry keys:

- **UpdateLastCheck.** A timestamp value in 100 UTC nanosecond units that stores the time and date of the last successful or unsuccessful update attempt.
- **UpdateZipTimestamp.** A timestamp value in 100 UTC nanosecond units that stores the time and date of the last successful update.

To establish an outgoing HTTP connection, CRM Desktop SSO also uses the following registry keys:

- HTTPClient keys
- Proxy keys

For more information, see "Overview of Customizing Authentication" on page 371.

Windows Registry Keys You Must Set to Enable CRM Desktop SSO

Table 28 describes command-line parameters that you must use with the msiexec.exe installer. Each command-line parameter modifies a Windows Registry key that CRM Desktop SSO requires. For more information, see "Using the Windows Command Line to Set Optional Parameters" on page 100.

In the Registry Editor (regedit), you can modify these keys in the following path:

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\SSO
```

Table 28. Windows Registry Keys You Must Set to Enable CRM Desktop SSO

Command Line Parameter	Description
SSOENABLE	Specifies to enable CRM Desktop SSO. You can use one of the following values: ■ **0.** Disable CRM Desktop SSO. The default value is 0. ■ **1.** Enable CRM Desktop SSO. If disabled, then CRM Desktop SSO it is not active when CRM Desktop communicates with Siebel CRM. CRM Desktop copies the value that the SSOENABLE parameter contains to the `SSO\Enable` registry key.
SSOSCRIPTFILENAME	Specifies the name of the JavaScript file that implements the CRM Desktop SSO logic. This file must define the entry point for SSO scenario handling. The file name must be relative to the directory that the SSOSCRIPTINCLUDEPATH parameter specifies. The default value is `sso.js`. CRM Desktop copies the value that the SSOSCRIPTFILENAME parameter contains to the `SSO\ScriptFileName` registry key. For more information about the entry point, see "Request Handler Function" on page 398.
SSOSCRIPTINCLUDEPATH	Specifies the directory path where the CRM Desktop SSO script file resides. This directory must also contain any files that this script references. For example: `C:\users\user1\AppData\Roaming\Oracle\CRM Desktop\bin` The default value is an empty string. CRM Desktop copies the value that the SSOSCRIPTINCLUDEPATH parameter contains to the `SSO\ScriptIncludePath` registry key. For autoupdate, you must use the SSOSCRIPTINCLUDETEMPLATE parameter instead of the SSOSCRIPTINCLUDEPATH parameter. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.

Table 28. Windows Registry Keys You Must Set to Enable CRM Desktop SSO

Command Line Parameter	Description
SSOUPDATEDISABLE	Specifies to enable autoupdate. You can use one of the following values: ■ **0.** Enable autoupdate. The default value is 0. ■ **1.** Disable autoupdate. CRM Desktop copies the value that the SSOUPDATEDISABLE parameter contains to the `SSO\UpdateDisable` registry key. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.
SSOURL	Specifies the URL or UNC path that CRM Desktop uses to download autoupdate information. The default value is an empty string. CRM Desktop copies the value that the SSOURL parameter contains to the `SSO\UpdateZIPURL` registry key. You must make sure you set this parameter during deployment. If you do not, then autoupdate will not work. If you use external provisioning, then the SSOURL parameter is not required. For more information, see "Installing CRM Desktop SSO If You Use Autoupdate" on page 379.

Table 28. Windows Registry Keys You Must Set to Enable CRM Desktop SSO

Command Line Parameter	Description
SSOCHECKINTERVAL	Specifies the timestamp value that CRM Desktop uses as the minimum time interval between update attempts. It measures this value in 100 nanosecond units. If this value is smaller than 36000000000 (1 hour), then CRM Desktop ignores this smaller value and sets the interval to 36000000000. The default value is 864000000000 (24 hours). CRM Desktop copies the value that the SSOCHECKINTERVAL parameter contains to the `SSO\UpdateCheckInterval` registry key.

SSOSCRIPTINCLUDETEMPLATE	Specifies the template that CRM Desktop uses to create a unique directory name. It uses this directory to store the scripts that it downloads from the URL that the SSOURL parameter identifies. You must use the following format:
	
	`%path%`
	
	For example:
	
	`%appdata%`
	
	The default value is `%appdata%\InvisibleSSO\Script`.
	
	CRM Desktop copies the value that the SSOSCRIPTINCLUDETEMPLATE parameter contains to the `SSO\ScriptIncludeTemplate` registry key.

Registry Keys That Control SSO for Siebel CRM Desktop

Table 29 describes the Windows Registry keys that you can modify to control SSO for Siebel CRM Desktop. For more information, see the topic about Regular Expression Syntax for JavaScript at the Microsoft Developer Network website. You can modify these keys in the following path in the Registry Editor (regedit):

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop
```

Table 29. Windows Registry Keys That Control SSO for Siebel CRM Desktop

Windows Registry Key	Description
AuthType	Determines the authentication type that the CRM Desktop-Login dialog box displays, by default. You can set it to one of the following values: ■ DIRECT ■ NTLM ■ SSO You can set this default value externally before the first time CRM Desktop runs. If you do this, then you must make sure the external value you provide is the same value that the following key contains: `HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\SSO\Enable` You must set this Enable key to one of the following values: ■ 1 for NTML or SSO ■ 0 for DIRECT
EndpointRegExp	Sets the regular expression that CRM Desktop uses to evaluate the rule that stops the interactive authentication. It must match the URL of the last web page that the interactive authentication displays. If you do not set EndpointRegExp, then CRM Desktop compares the URL that displays the login dialog box to the URL that displays the last web page.
SuccessLoginRegExp	Sets the regular expression that CRM Desktop uses to evaluate the rule that stops the interactive authentication. It must match the body of the last web page that the interactive authentication displays. If you do not set SuccessLoginRegExp, then CRM Desktop compares the page contents to the following regular expression: `FAULTSTRING.*?10944629` This expression matches the response that the Siebel EAI server returns in reply to an empty GET request. It is a SOAP FAULT response that includes a message that indicates that it passed the empty request body. This is the default value.

Registry Keys That Control SSO for Credentials

Table 30 describes the Windows Registry keys that you can modify to control SSO for credentials. You can modify these keys in the following path in the Registry Editor (regedit):

```
HKEY_CURRENT_USER\Software\Oracle\CRM Desktop\Credentials
```

Table 30. Windows Registry Keys That Control SSO for Credentials

Windows Registry Key	Description
Siebel:SSOUser	Specifies the value for the user name that Siebel CRM SSO enforces for installation. Applicable only if the authorization type is SSO. If you: ■ **Set Siebel:TrustToken.** The User name field is read-only and Siebel CRM SSO automatically populates a value in this field. ■ **Do not set Siebel:TrustToken.** The user can enter a value in the User name field.

Parameters You Can Use with Log Files

This topic describes parameters that you can use with log files. It includes the following topics:

For additional information, see “Using the Windows Registry to Control Siebel CRM Desktop” on page 105.

Parameters You Can Use with the General Log

Table 31 describes the parameters that you can use with the General Log with Siebel CRM Desktop.

Table 31. Parameters You Can Use with the General Log

Parameter	Description
enable	You can use one of the following values: ■ **1.** General Log is enabled. ■ **0.** General Log is disabled. To disable the general log, you must use the Windows Registry. You cannot use the Logging Configuration dialog box. For more information, see "Assigning Logging Profiles for Siebel CRM Desktop" on page 120.
log_level	Defines logging verbosity. You can use one of the following integers: ■ **4294967295.** Disable all. ■ **0.** Enable all. ■ **1000.** Debug. ■ **2000.** Information messages. ■ **3000.** Warnings. ■ **4000.** Errors. ■ **5000.** Fatal errors. For more information, see "Setting Logging Verbosity" on page 121.
out_file	You can include a string that identifies the file name for the general log. For example, assume you use the following value: `log.txt` In this example, the first file name is log.0000.txt, the second file name is log.0001.txt, and so on. If you do not include the out_file parameter, then CRM Desktop uses `log.txt` as the default value.
enable_dbg_window	You can use one of the following values: ■ **1.** Enables output through the OutputDebugString that CRM Desktop displays in the VS Debug Output window during a debugging session. The OutputDebugString is a string that CRM Desktop sends to the debugger for display. The VS Debug Output window is a window in Microsoft Visual Studio that displays debugging results. For more information about Visual Studio, see the documentation at the Microsoft TechNet Web site. ■ **0.** Disables the General Log.

Table 31. Parameters You Can Use with the General Log

Parameter	Description
enable_cout	You can use one of the following values: ■ **1.** Enables logging in the console. ■ **0.** Disables logging in the console.
max_size_bytes	You can use an integer that sets the maximum number of bytes that a single log file can contain. For example, if you set max_size_bytes to 10485760 bytes, and if the current log reaches 10485760 in size, then CRM Desktop creates a new log file. This behavior is similar to setting the rotate_on_start_only parameter to 0.
reuse_not_exceeded	Determines the file that CRM Desktop uses when a new logging session starts. You can use one of the following values: ■ **1.** If the size of the most current log file is less than the value that the max_size_bytes parameter sets, then CRM Desktop reuses this file. 1 is the default value. ■ **0.** CRM Desktop does not reuse the most current log file.
rotate_on_start_only	*Log file rotation* is a configuration that CRM Desktop uses to prevent log files from growing indefinitely. You can use one of the following values: ■ **1.** Examines the log file size that exists when Outlook starts. If this log file size is larger than the value that the max_size_bytes parameter specifies, then CRM Desktop creates a new log file and then logs all subsequent entries to this new file. The log file size can exceed the value that the max_size_bytes parameter specifies. CRM Desktop stores all log entries for one Outlook session to one file. 1 is the default value. ■ **0.** Examines the log file size every time CRM Desktop writes to the log. If this log file size is larger than the value that the max_size_bytes parameter specifies, then CRM Desktop creates a new log file. Multiple log files can exist for a single Outlook session.
file_count	An integer that specifies the maximum number of rotated log files.
initial_erase	You can use one of the following values: ■ **1.** Deletes the contents of all general log files when CRM Desktop starts. ■ **0.** Does not delete the contents of any general log file when CRM Desktop starts.

Table 31. Parameters You Can Use with the General Log

Parameter	Description
time_format	A string that specifies the date and dime format in the log files. You can use one of the following values: ■ **$nano.** Nanoseconds. ■ **$micro.** Microseconds. ■ **$mili.** Milliseconds. ■ **$ss.** A 2 digit second. ■ **$mm.** A 2 digit minute. ■ **$hh.** A 2 digit hour. ■ **$dd.** A 2 digit day. ■ **$MM.** A 2 digit month. ■ **$yy.** A 2 digit year. ■ **$yyyy.** A four digit year. Nanoseconds or microseconds are available only if the system clock allows nanoseconds or microseconds. If the system clock does not allow nanoseconds or microseconds, and if you specify nanoseconds or microseconds, then CRM Desktop pads the value that it creates for the log entry with zeros.
log_format	A string that specifies the logging message format. You can use one of the following values: ■ **%index%.** Sequential message index. ■ **%time%.** Time in the format that the time_format parameter sets. ■ **%thread%.** Thread ID that CRM Desktop uses to log the message. ■ **%level%.** Message logging level in human readable format. ■ **%level_num%.** Message logging level in numeric format. ■ **%log_src%.** Source of the logging message. This source can be a CRM Desktop internal component. For example, a connector or application manager.

Table 31. Parameters You Can Use with the General Log

Parameter	Description
starter	A string that specifies the first message that CRM Desktop adds to the log file when it starts logging. For example: `--- logging is initialized ---`
finisher	A string that specifies the last message that CRM Desktop adds to the log file when it stops logging. For example: `--- logging is finished ---` If a log does not include the finisher message, then it indicates that CRM Desktop stopped logging abnormally.

Parameters You Can Use with the Exception Log

Table 32 describes the parameters that you can use with the Exception Log.

Table 32. Parameters You Can Use with the Exception Log

Parameter	Description
enable	You can use one of the following values: ■ **1.** Exception Log is enabled. ■ **0.** Exception Log is disabled.
file_count	An integer that specifies the maximum number of rotated log files.
initial_erase	You can use one of the following values: ■ **1.** Deletes the contents of all exception log files when CRM Desktop starts. ■ **0.** Does not delete the contents of any exception log file when CRM Desktop starts.
max_size_bytes	An integer that sets the maximum number of bytes that a single log file can contain. For more information, see the description for the max_size_bytes parameter in "Parameters You Can Use with the General Log" on page 422.
out_file	A string that specifies the base file name that CRM Desktop uses for logging.

Parameters You Can Use with the Crash Log

Table 33 describes the parameter that you can use with the crash log. You can use the following parameters:

- any_in_trace
- all_in_trace

- ex_white_list
- ex_black_list

To separate values in these parameters, you can use the following symbols:

- , (comma)
- ; (semicolon)
- | (vertical bar)
- / (forward slash)
- \t (backward slash with a t)
- (single space)

Table 33. Parameters You Can Use with the Crash Log

Parameter	Description
enable	You can use one of the following values: ■ **1.** Crash log is enabled. ■ **0.** Crash log is disabled.
count	An integer that specifies the maximum of old log files that CRM Desktop preserves in the output logging directory.
on_top	A string that specifies the filtering condition for the crash log. CRM Desktop applies this filtering only if the item you specify resides on the top of the stack trace when the failure occurs. For example, you can use the following value: `CRMDesktop3D.dll` If you do not specify a value, then CRM Desktop ignores this parameter.
on_top_op	You can use one of the following values: ■ **0.** Combine the item that the on_top parameter filters with the item that the any_in_trace parameter specifies or with the item that the all_in_trace parameter specifies. Use an OR logic operation. ■ **1.** Combine the item that the on_top parameter filters with the item that the any_in_trace parameter specifies and with the item that the all_in_trace parameter specifies. Use an AND logic operation. ■ **2.** Combine the item that the on_top parameter does not filter with the item that the any_in_trace parameter specifies or with the item that the all_in_trace parameter specifies. Use an OR logic operation. ■ **3.** Combine the item that the on_top parameter does not filter with the item that the any_in_trace parameter specifies and with the item that the all_in_trace parameter specifies. Use an AND logic operation.

Table 33. Parameters You Can Use with the Crash Log

Parameter	Description
any_in_trace	A string that specifies a list of items. If the stack trace includes an item you specify, then CRM Desktop saves a crash log for this item. For example, you can use the following value: `Ntdll.dll,Kernel32.dll` In this example, if the stack trace includes an entry for Ntdll.dll or Kernel32.dll, then CRM Desktop saves a crash log for this item.
all_in_trace	A string that specifies a list of items. If the stack trace includes all the items you specify, then CRM Desktop saves a crash log for these items. For example, you can use the following value: `Ntdll.dll,Kernel32.dll` In this example, if the stack trace includes an entry for Ntdll.dll and an entry for Kernel32.dll, then CRM Desktop saves a crash log for these items.
delay_ms	An integer that specifies a delay in milliseconds. If the time that elapses after a previous exception exceeds the value you specify, then CRM Desktop saves a crash log. The default value is 0. This default makes sure that CRM Desktop always saves a crash log.
ex_white_list	A string that specifies a list of C++ exception names. If an exception occurs for an exception name you specify, then CRM Desktop creates a minilog for this exception.
ex_black_list	A string that specifies a list of C++ exception names that CRM Desktop uses to not create a minilog. If the white_list parameter is empty, then CRM Desktop creates a log for all exceptions except the exceptions that the black_list parameter lists.
sigabrt	*SIGABRT* is a signal that CRM Desktop sends to a process to tell it to end. You can use one of the following values: ■ **1.** Intercept the SIGABRT signal to save a crash log. You must use this parameter only for troubleshooting. Do not enable it permanently. ■ **0.** Do not intercept the SIGABRT signal to save a crash log.
minidump_type	An integer that specifies the minilog type. The default value is 1 (MiniDumpNormal|MiniDumpWithDataSegs). Use this parameter with caution because other values could increase log size significantly. For more information, see documentation about the DumpType parameter of the MiniDumpWriteDump function at the Microsoft MSDN website.

Parameter You Can Use with the SOAP Log

Table 34 describes the parameters that you can use with the SOAP log.

Table 34. Parameters You Can Use with the SOAP Log

Parameter	Description
enable	You can use one of the following values: ■ **1.** SOAP log is enabled. ■ **0.** SOAP log is disabled.
enable_dbg_window	You can use one of the following values: ■ **1.** Enables output through the OutputDebugString. The SOAP log displays in the VS Debug Output window during a debugging session. ■ **0.** Disables output through the OutputDebugString. The SOAP log does not display in the VS Debug Output window during a debugging session.
enable_cout	You can use one of the following values: ■ **1.** Enables logging to the console. ■ **0.** Disables logging to the console.
file_count	An integer that specifies the maximum number of rotated log files.
root_tag_name	A string that specifies the root XML tags that CRM Desktop uses in generated XML files. The default value is `soap_comm_log`. If you use the following value, then it does not write the opening or closing root tags: `""`
file_extension	A string that specifies the file extension that CRM Desktop uses for generated log files. The default value is `xml`.
bin_mode	You can use one of the following values: ■ **1.** CRM Desktop uses the `\n` format for new line termination. ■ **0.** CRM Desktop uses the `\r\n` format for new line termination.

Parameters You Can Use with the Synchronization Log

Table 35 describes the parameters that you can use with the synchronization log.

Table 35. Parameters You Can Use with the Synchronization Log

Parameter	Description
enable	You can use one of the following values: ■ **1.** Synchronization log is enabled. ■ **0.** Synchronization log is disabled.
file_count	An integer that specifies the maximum number of rotated log files.
out_file	A string that specifies the base file name that CRM Desktop uses for logging.

Filters in the CRM Desktop Filter - Edit Criterion Dialog Box

Table 36 describes the filters that the user can specify in the CRM Desktop Filter - Edit Criterion dialog box. The user chooses values in the Condition field and the Value field to specify a filter. It is recommended that the user use the = (equal) operator or the <> (not equal) operator only for the Exact Date filter. For more information, see "Controlling the Date Range in the Filter Records Tab" on page 134.

Table 36. Filters in the CRM Desktop Filter - Edit Criterion Dialog Box

Filter	Description
Days From Today	Sets a date range that includes a specific number of days before or after today. The user can enter one of the following values: ■ **Positive value.** Indicates the number of days to include in the filter starting with tomorrow and continuing into the future. ■ **Negative value.** Indicates the number of days to include in the filter starting with yesterday and continuing into the past. For example, the user can specify one of the following filters: ■ **Start < 2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start before January 24, 2012. ■ **Start > 2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start after than January 24, 2012. ■ **Start <= 2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start on or before January 24, 2012. ■ **Start >= 2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start on or after January 24, 2012. ■ **Start < -2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start before January 20, 2012. ■ **Start > -2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start after January 20, 2012. ■ **Start <= -2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start on or before January 20, 2012. ■ **Start >= -2 Days From Today.** If today is January 22, 2012, then CRM Desktop synchronizes activities that start on or after January 20, 2012.

Table 36. Filters in the CRM Desktop Filter - Edit Criterion Dialog Box

Filter	Description
Exact Date	The user can choose an exact date from the calendar in the CRM Desktop Filter - Edit Criterion dialog box. CRM Desktop only synchronizes records for the date that the user chooses. For example, the following condition synchronizes activities that are due only for January 1, 2012: ■ Field is Due ■ Condition is = ■ Value is January 1, 2012
Month Ago	Sets a date range that is related to 30 days prior to today. For example, the user can specify one of the following filters: ■ **Start < Month Ago.** If today is January 31, 2012, then CRM Desktop synchronizes activities that start before January 1, 2012. ■ **Start > Month Ago.** If today is January 31, 2012, then CRM Desktop synchronizes activities that start after January 1, 2012. ■ **Start <= Month Ago.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start on or before January 1, 2012. ■ **Start >= Month Ago.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start on or after January 1, 2011.
Month Ahead	Sets a date range that is related to 30 days after today. For example, the user can specify one of the following filters: ■ **Start < Month Ahead.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start before January 31, 2012. ■ **Start > Month Ahead.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start after January 31, 2012. ■ **Start <= Month Ahead.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start on or before January 31, 2012. ■ **Start >= Month Ahead.** If today is January 1, 2012, then CRM Desktop synchronizes activities that start on or after January 31, 2012.
Today	Sets a date range that is related to today. For example, the user can specify one of the following filters: ■ **Start < Today.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 14, 2012. ■ **Start > Today.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 16, 2012. ■ **Start <= Today.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 15, 2012. ■ **Start >= Today.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 15, 2012.

Table 36. Filters in the CRM Desktop Filter - Edit Criterion Dialog Box

Filter	Description
Tomorrow	Sets a date range that is related to tomorrow. For example, the user can specify one of the following filters: ■ **Start < Tomorrow.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 15, 2012. ■ **Start > Tomorrow.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 17, 2012. ■ **Start <= Tomorrow.** If if today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 16, 2012. ■ **Start >= Tomorrow.** If if today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 16, 2012.
Yesterday	Sets a date range that is related to yesterday. For example, the user can specify one of the following filters: ■ **Start < Yesterday.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 13, 2012. ■ **Start > Yesterday.** If today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 15, 2012. ■ **Start <= Yesterday.** If if today is January 15, 2012, then CRM Desktop synchronizes activities that start on or before January 14, 2012. ■ **Start >= Yesterday.** If if today is January 15, 2012, then CRM Desktop synchronizes activities that start on or after January 14, 2012.

Threshold That Siebel CRM Desktop Uses to Display the Confirm Synchronization Tab

Table 37 lists the minimum number of records that the user must delete to cause Siebel CRM Desktop to display the Confirm Synchronization tab. For information on how to configure this behavior, see "Specifying the Type of Object the User Can Confirm for Deletion" on page 153.

Table 37. Threshold That CRM Desktop Uses to Display the Confirm Synchronization Tab

Object	Number of Deleted Records
Account	3
Account.Account_Note	10
Account.Assignment_Group.Association	5
Account.Business_Address (HOR)	10

Table 37. Threshold That CRM Desktop Uses to Display the Confirm Synchronization Tab

Object	Number of Deleted Records
Account.Business_Address.Association (SIA)	10
Account.Contact.Association	20
Account.Industry.Association	5
Account.Position.Association	20
Action	20
Action.Contact.Association	100
Action.Employee.Association	100
Assignment_Group	Not applicable
Attachment	10
Business_Address (SIA)	10
Contact	10
Contact.Business_Address.Association (SIA)	10
Contact.Contact_Note	10
Contact.Contact_Sync_Owner	Not applicable
Contact.Personal_Address (HOR)	10
Contact.Position.Association	20
Currency	Not applicable
Defaults	Not applicable
Employee	Not applicable
Employee.Position.Association	Not applicable
Industry	Not applicable
Internal_Division	Not applicable
Internal_Product	Not applicable
Opportunity	5
Opportunity.Assignment_Group.Association	5
Opportunity.Contact.Association	20
Opportunity.Internal_Division.Association	5
Opportunity.Opportunity_Note	10
Opportunity.Opportunity_Product	5
Opportunity.Position.Association	20

Table 37. Threshold That CRM Desktop Uses to Display the Confirm Synchronization Tab

Object	Number of Deleted Records
Position	Not applicable
Sales_Method.Sales_Cycle_Def	Not applicable

Files That the Customization Package Contains

To customize some Siebel CRM Desktop features, you can modify XML files and JavaScript files in the customization package. CRM Desktop includes the following basic customization capabilities:

- Adjusting the business logic to suit the business environment
- Customizing the user interface
- Specifying security and data validation rules

For more information, see "Where Siebel CRM Desktop Stores Data in the File System" on page 83.

Files in the Customization Package

Table 38 describes the XML files that Siebel CRM Desktop includes in the customization package. For more information, see "About the Customization Package" on page 33.

Table 38. XML Files in the Customization Package

File Name	Description
code_pages.xml	This XML file provides a simple listing of code pages used for conversion when necessary and indicates that it is not customizable by the user.
connector_configuration.xml	This XML file provides the following capabilities: ■ Defines objects that are synchronized ■ Defines the criteria that CRM Desktop uses to detect duplicate objects in the Siebel database ■ Defines the preset filters for a custom synchronization ■ Defines the object types that CRM Desktop displays in the Filter Records tab of the Synchronization control panel. ■ Defines presets for sliding date ranges. For more information, see "Customizing Synchronization" on page 161.
data_sources.xml	Defines the data source that CRM Desktop uses in views, forms, and dialog boxes for Auto-Complete.

Table 38. XML Files in the Customization Package

File Name	Description
dialogs.xml	Defines the layout for a custom dialog box. For more information, see "Customizing Dialog Boxes" on page 162.
forms_xx.xml	CRM Desktop uses the forms XML files according to the version of Outlook installed on the client. These XML files provide the following capabilities: ■ Indicates the fields that CRM Desktop uses to store references between objects ■ Defines the form layout for each object ■ Defines the field validation rules on a form ■ Defines business logic in JavaScript ■ Defines controls that CRM Desktop uses on a form Throughout this book, the term forms_xx.xml refers generically to these forms files. The xx refers to the internal version of Outlook as defined by Microsoft. If you edit a forms_xx.xml file, then make sure you edit the file that supports the version of Outlook that is installed on the client. The business_logic.js file describes most business logic. To specify business logic, it is recommended that you use JavaScript in one of the forms_xx.xml files only if CRM Desktop runs this logic in the current form. If many objects are involved, then it is recommended that you specify business logic in the business_logic.js file. For more information, see "Customizing Forms" on page 161.
info.xml	This XML file provides the following capabilities: ■ Identifies the product name ■ Identifies the package version and the CRM Desktop version ■ Identifies the product versions that are compatible with the package ■ Identifies the Siebel Server versions that are compatible with the package ■ Identifies the general comments for the package You can use it to track the package version. You can use the product name and version to check compatibility. For more information, see "How Siebel CRM Desktop Determines Compatibility" on page 75.

Table 38. XML Files in the Customization Package

File Name	Description
lookup_view_defs.xml	Defines the configuration for view lookup definitions. This file applies only for versions that occur prior to Siebel CRM Desktop version 3.5. For more information, see "Customizing the SalesBook Control" on page 163.
package_res.xx_YY.xml	Provide string and image resources for use throughout the application. The appropriate language and region is determined by the Outlook client. In the event that a string cannot be located in the language-specific packge_res_xx_YY.xml file, either because the file is not present or the individual string is missing, CRM Desktop will use the default string declared in package_res.xml (which is also the English version). Throughout this book, the term package_res.xx_YY.xml refers generically to these package files. Where xx and yy define the language and region. For example, package_res.pt_BR indicates the language is Brazilian and the region is Portuguese.
platform_configuration.xml	Defines the configuration for the platform. For example, the platform_configuration.xml file defines how to do the following: ■ Connect to the Internet ■ Share native Outlook items ■ Convert contacts ■ Control the synchronization intervals that display in the Synchronization tab ■ Control the data that CRM Desktop removes if the user removes CRM Desktop
siebel_basic_mapping.xml	This XML file provides the following capabilities: ■ Defines field mapping between CRM Desktop and Outlook ■ Defines field mapping between CRM Desktop and Siebel CRM ■ Describes objects to add to Outlook ■ Defines the form that CRM Desktop uses to display an object in Outlook ■ Defines a set of custom Outlook views that CRM Desktop applies for an object For more information, see "Customizing Field Mapping" on page 160.

Table 38. XML Files in the Customization Package

File Name	Description
siebel_meta_info.xml	This XML file provides the following capabilities: ■ Defines the object types that CRM Desktop supports ■ Defines fields and their types ■ Defines the XML element names that CRM Desktop uses to create a Siebel message For more information, see "Customizing Meta Information" on page 164.
toolbars_xx.xml	Defines custom toolbars that CRM Desktop displays on a native Outlook form, custom form, or in the Outlook window. CRM Desktop uses the toolbars files according to the version of Outlook installed on the client. Throughout this book, the term toolbars_xx.xml refers generically to one of these toolbars files. If you edit a toolbars.xml file, then make sure you edit the file that supports the version of Outlook that is installed on the client. External JavaScript files specify all programmable actions for a toolbar. For more information, see "Customizing Toolbars" on page 162.
views.xml	Defines the views that CRM Desktop uses in CRM Desktop forms and Outlook windows. For more information, see "Customizing Views" on page 163.

JavaScript Files in the Customization Package

Table 39 describes the JavaScript files that Siebel CRM Desktop includes in the customization package.

Table 39. JavaScript Files in the Customization Package

JavaScript File Name	Description
actions.js	Defines actions for the toolbar.
actions_support.js	Defines action support functions for the toolbar.
activity_processor.js	Defines activity processing. You must not modify this file.
application_script.js	Defines entry points for scripts and to call scripts from other files.
autoresolver.js	Defines functions to resolve conflicts.
autoresolve_helpers.js	An internal CRM Desktop file. You must not modify this file.

Table 39. JavaScript Files in the Customization Package

JavaScript File Name	Description
business_logic.js	Defines logic for the following items: ■ Activities ■ Mail processing ■ The data model
data_model.js	Defines the data model and functions for objects.
forms.js	Defines user interface actions.
form_helpers.js	Defines functions to handle user interface events.
helpers.js	Defines utility functions.
idle.js	Defines the Idle Processing Manager and idle handlers.
md5.js	Defines how to implement MD5 (Message Digest Algorithm).
mvg_dialogs.js	Defines controls for multi-value groups.
raw_item_functions.js	Defines functions that access Outlook items.
recurrence_processing.js	Defines patterns for recurrence processing. For more information, see "How Siebel CRM Desktop Transforms Objects Between Siebel CRM Data and Microsoft Outlook Data" on page 449.
sb_helpers.js	Defines utility functions that are specific to Siebel CRM.
security_manager.js	Defines the security model.
security_utils.js	Defines security definitions that are specific to Siebel CRM.

JavaScript Files That Siebel CRM Desktop Uses Internally

Siebel CRM Desktop uses the following files internally. You must not modify these files:

- actions_support.js
- activity_processor.js
- autoresolve_helpers.js
- data_model.js
- form_helpers.js
- helpers.js
- idle.js
- md5.js
- mvg_dialogs.js
- security_manager.js

Microsoft Outlook Field Types and Equivalent Convertor Classes

This topic includes example Microsoft Outlook field types and their equivalent convertor classes. Siebel CRM Desktop defines Outlook field types in the siebel_basic_mapping.xml file.

Example of a Number Field

The following code is an example of a number field:

```
<field id="Percent Complete">
    <reader>
      <mapi_user>
        <user_field id="sbl Percent Complete" sbl_field_type="3"></user_field>
      <convertor>
        <double/>
      </convertor>
      </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Percent Complete" ol_field_type="3"></user_field>
      <convertor>
        <double/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

Example of a String Field

The following code is an example of a string field:

```
<field id="Currency Short Name">
  <reader>
    <mapi_user>
      <user_field id="sbl Currency Short Name" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Currency Short Name" ol_field_type="1"></user_field>
      <convertor>
        <string/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

Example of a Datetime Field

The following code is an example of a datetime field:

```
<field id="Primary Revenue Close Date">
    <reader>
      <mapi_user>
        <user_field id="sbl CloseDate" ol_field_type="5"></user_field>
        <convertor>
          <datetime/>
        </convertor>
      </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl CloseDate" ol_field_type="5"></user_field>
      convertor>
        datetime/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

Example of a Boolean Field

The following code is an example of a boolean field:

```
<field id="Executive Priority Flag">
  <reader>
    <mapi_user>
|      <user_field id="sbl Priority" ol_field_type="6"></user_field>
    <convertor>
        <bool/>
      </convertor>
    </mapi_user>
  </reader>
  <writer>
    <Outlook_user>
      <user_field id="sbl Priority" ol_field_type="6"></user_field>
      <convertor>
        <bool/>
      </convertor>
    </Outlook_user>
  </writer>
</field>
```

B How Siebel CRM Desktop Maps Fields Between Siebel CRM Data and Microsoft Outlook Data

This appendix describes how Siebel CRM Desktop maps fields between Siebel CRM data and Microsoft Outlook data. It includes the following topics:

How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar

Table 40 describes how Siebel CRM Desktop maps objects between a Siebel CRM activity and Outlook Calendar.

Table 40. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar

Siebel Field	Outlook Field	Required	Description
Account	Account association that the user specifies when the user links a CRM record.	No	Not applicable.
Comments	Description A private calendar item does not include a description.	No	Not applicable.
Contacts	List of contacts that is received from the list of participants that remain after employees are removed from this list when CRM Desktop resolves the primary email of the contacts.	No	The logic that applies for the Employees field also applies for the Contacts field.
Created by	User ID of the employee who creates the activity.	Yes	Not applicable.
Created Date	Timestamp of when the activity is created.	Yes	Not applicable.

Table 40. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar

Siebel Field	Outlook Field	Required	Description
Description	For more information, see "How Siebel CRM Desktop Handles Private Activities" on page 443.	No	The value that is set for a private task is defined in the customization package and is included in the localization resources.
Duration	Duration	Yes	Not applicable.
Employees	List of employees that is received from the list of participants when CRM Desktop resolves the primary email of the employees.	No	CRM Desktop does the preliminary resolution when the user saves a calendar appointment. To improve the quality of this list, the Siebel Server analyzes the processing that occurs on the server. It does this work after the next synchronization.
ExceptionsList	No direct mapping to the Outlook field exists.	Yes	For more information, see "How Siebel CRM Desktop Handles a Repeating Calendar Appointment" on page 54.
Meeting Location	Location	No	Not applicable.
Opportunity	Opportunity association that the user specifies when the user links a CRM record.	No	Not applicable.
Owner	For more information, see "How Siebel CRM Assigns Meeting Organizers" on page 44.	Yes	Not applicable.
PIM Meeting Participants	List of participants for the calendar appointment, delimited by a semicolon (;).	No	Not applicable.
Planned	Start Date/Time	Yes	Not applicable.
Planned Completion/ End time	End Date/Time	Yes	Not applicable.

Table 40. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook Calendar

Siebel Field	Outlook Field	Required	Description
Priority Values are: ■ 1-ASAP ■ 2-High ■ 3-Medium ■ 4-Low	Outlook priority Values are: ■ 1-High ■ 2-High ■ 3-Medium ■ 4-Low	No	For more information, see "How Siebel CRM Desktop Maps the Priority Field" on page 444.
Private	Private	No	Not applicable.
Repeat	Recurring flag	No	Not applicable.
Repeat Frequency	CRM Desktop uses a binary field that combines the data from a number of Outlook fields. The frequency is daily, weekly, monthly, or yearly.	No	Not applicable.
Repeat Until	Recurrence Range End	No	Not applicable.
Type	Type that the user specifies when linking the CRM record. The default value is Calendar Appointment.	Yes	The user can choose the activity type in the form of the Outlook calendar item only for an activity that contains the following *Display in* value: Calendar and Activities The customization package defines the default value. You can change this value.

How Siebel CRM Desktop Handles Private Activities

Siebel CRM Desktop handles a private activity in the following ways:

- If an activity in Siebel CRM is marked private, then it does not synchronize this activity from the Siebel Server to Outlook. The master filters in the customization package restrict it from synchronizing a private activity.
- If the user creates a private activity in Outlook and then attempts to share it with Siebel CRM, then validation disallows this attempt. For example, assume a user creates a shared activity in a Siebel CRM calendar appointment. If the user attempts to mark this calendar appointment as private, then CRM Desktop displays a warning message that it cannot share a private Outlook item with Siebel CRM.

How Siebel CRM Desktop Maps the Priority Field

Siebel CRM data includes an ASAP priority value but Microsoft Outlook data does not. It is not possible to save the ASAP value during synchronization, so Siebel CRM Desktop does not include it in the mapping. If a Siebel task is marked 1-ASAP, then CRM Desktop creates a mapping error.

The Outlook activity includes another field to store the Priority value that CRM Desktop derives from the Siebel CRM data. Note the following behavior:

- If the Outlook item is updated, and if the Priority of the Outlook item is High, then CRM Desktop validates the stored value.
- If the value that is stored for the activity is ASAP, then CRM Desktop uses the ASAP value from the activity.
- If the value that is stored for the activity is not ASAP, then CRM Desktop uses the value from the Outlook record.

How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook To Do Items

Table 41 describes how CRM Desktop maps fields between a Siebel CRM activity and the Outlook task.

Table 41. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook To Do Items

Siebel Field	Outlook Field	Required	Description
Owner	Owner	Yes	For more information, see "How Siebel CRM Desktop Maps the Owner Field Between Siebel CRM Activities and Outlook To Do Items" on page 446.
Type	The type that the user specifies when the user links a CRM record. The default value is To Do.	Yes	If a user shares a task, then CRM Desktop creates an activity in Outlook with a *Display in* value that includes the following: To Do and Activities The customization package defines the default value. You can change this value.
Description	For more information, see "How Siebel CRM Desktop Handles Private Activities" on page 443.	No	The customization package defines the value that is set for a private task. Localization resources include this value.

Table 41. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook To Do Items

Siebel Field	Outlook Field	Required	Description
Priority Values are: ■ 1-ASAP ■ 2-High ■ 3-Medium ■ 4-Low	Outlook priority Values are: ■ 1-High ■ 2-High ■ 3-Medium ■ 4-Low	No	For more information, see "How Siebel CRM Desktop Maps the Priority Field" on page 444.
Comments	Description A private task does not include a description.	No	If the Private check box is set to allow a shared task, then CRM Desktop sets the Comments field to a value that the customization package defines, such as Unavailable.
Start Date	Start Date	No	The Start Date for an activity is the start date that the user sets for a task. If the user does not enter the start date, then the start date for the activity is also empty.
Done	Completed Date	No	The Done date is the date that Siebel CRM displays as the Actual End date in the Siebel Web Client.
Completed flag	Completed flag	No	Not applicable.
Due	Due Date	No	Not applicable.
Percent complete	Percent complete	No	The value of the Percent complete field is related to the status. For more information, see "How Siebel CRM Desktop Maps the Status Field of an Activity" on page 446.
Status	Status	No	For more information, see "How Siebel CRM Desktop Maps the Status Field of an Activity" on page 446.
Account	Account association that the user specifies when the user links a CRM record.	No	Not applicable.
Opportunity	Opportunity association that the user specifies when the user links a CRM record.	No	Not applicable.

Table 41. How Siebel CRM Desktop Maps Fields Between Siebel Activities and Outlook To Do Items

Siebel Field	Outlook Field	Required	Description
Contacts	Contact association that the user specifies when the user links a CRM record.	No	Not applicable.
Created by	User ID of the employee who creates the activity.	Yes	Not applicable.
Created Date	Timestamp of when the activity is created.	Yes	Not applicable.

How Siebel CRM Desktop Maps the Owner Field Between Siebel CRM Activities and Outlook To Do Items

This topic describes how Siebel CRM Desktop maps the owner field between a Siebel CRM activity and the Outlook task. CRM Desktop sets the owner differently for each of the following situations:

- A shared task is assigned to another employee. It sets the assignee as the owner of the activity.
- A shared task is assigned to a number of employees. It sets each assignee as the owner of their own activity.
- A shared task is assigned to a shared contact or to an external person. It sets the employee who created the task as the owner of the activity.
- A task is created by an external person and assigned to an employee and this employee shares the task. It sets the employee as the owner of the activity.
- An external person who is not a Siebel employee sends a task to a number of employees who are CRM Desktop users. It sets each employee as the owner of the activity and adds the remaining employees to the employee team. It does this work after each employee accepts and shares the task.

How Siebel CRM Desktop Maps the Status Field of an Activity

Table 42 describes how Siebel CRM Desktop maps the status of a Siebel CRM activity to the status of the Outlook task when the user shares a task.

Table 42. How Siebel CRM Desktop Maps the Status Field of an Activity

Siebel Status	Outlook Status
Not Started	Not Started
In Progress	In Progress
Done	Completed
On Hold	Waiting on someone else
Canceled	Deferred

Table 43 describes how CRM Desktop maps the Status field of a Siebel CRM activity when the value of the Display In field of a To Do activity contains the following value:

To Do and Activities

CRM Desktop does this work during synchronization.

Table 43. How Siebel CRM Desktop Maps the Status of a Siebel CRM Activity When To Do Contains *To Do and Activities*

Siebel Status	Outlook Status
Not Started/Acknowledged	Not Started
In Progress	In Progress
Done	Completed
On Hold	Waiting on someone else
Canceled/Declined	Deferred
Any other value	The logic described in Table 44 on page 447 determines the value for the status.

Table 44 describes how CRM Desktop uses the Percent Complete value in the Outlook task to determine the value of the Status field.

Table 44. How Siebel CRM Desktop Maps the Status of a Siebel CRM Activity That Is Determined By Percent Complete

Siebel Status	Value of Percent Complete for the Outlook To Do Item
Not Started	0
In Progress	Greater than zero and less than one hundred
Completed	100

Scenario for Mapping the Status Field of an Activity

This topic gives one example of how Siebel CRM Desktop maps the status field of an activity. The following sequence occurs:

1. A user creates a shared task in Outlook that includes a status that is In Progress and a Percent Complete that is 0.
2. CRM Desktop creates an activity on the Siebel Server that includes a status of In Progress.
3. The user synchronizes with the Siebel Server.
4. On the Siebel Server, CRM Desktop changes the status to Requested.
5. The user synchronizes with the Siebel Server again.
6. In Outlook, CRM Desktop updates the activity status to Requested.

7 At this point, the Percent Complete field of the task is 0. CRM Desktop updates the task status to Not Started.

How Siebel CRM Desktop Maps Fields Between Siebel CRM Activities and Outlook Emails

Table 45 describes how Siebel CRM Desktop maps fields between a Siebel CRM activity and the Outlook email.

Table 45. How Siebel CRM Desktop Maps Fields Between a Siebel CRM Activity and the Outlook Email

Siebel CRM Field	Required	Outlook Field
Type	Yes	One of the following: ■ If the user is the receiver, then Email - Inbound ■ If the user is the sender, then Email - Outbound
Description	No	Email Subject
Display	Yes	Changes to Communication and Activities. This value is the default value.
Email Attachment Flag	No	Changes to Y. This value is the default value.
Email BCC Line	No	Email BCC Line
Email Body	No	Email Body
Email CC Line	No	Email CC Line
Email Sender Address	No	CRM Desktop maps part of the value in the From Line to the Email Address. For example, assume the From Address includes the following address: Jane Smith, or Jane Smith <jane.smith@pcscomputing.com> In this example, CRM Desktop resolves the Email Sender Address to the following address: jane.smith@pcscomputing.com
Email Sender Name	No	CRM Desktop maps part of the From Line to the Display Name. For example, using the example from the Email Sender Address, the Email Sender Name resolves to Jane Smith.
Email To Line	No	Email To Line.

Table 45. How Siebel CRM Desktop Maps Fields Between a Siebel CRM Activity and the Outlook Email

Siebel CRM Field	Required	Outlook Field
Priority	No	Importance. CRM Desktop applies the following mapping: ■ 1-ASAP in Siebel CRM data is ASAP in Outlook ■ 2-High in Siebel CRM data is High in Outlook ■ 3-Medium in Siebel CRM data is Normal in Outlook ■ 4-Low in Siebel CRM data is Low in Outlook For more information, see "How Siebel CRM Desktop Maps the Priority Field" on page 444.
Account Id	No	The Primary account association that the user specifies when the user creates the activity.
Opportunity Id	No	The opportunity association that the user specifies when the user creates the activity.
Attachment	No	The original email that CRM Desktop saves as an attachment to the activity.

How Siebel CRM Desktop Transforms Objects Between Siebel CRM Data and Microsoft Outlook Data

This topic describes how Siebel CRM Desktop transforms objects between Siebel CRM Data and Microsoft Outlook data. It includes the following topics:

If the user synchronizes a repeating activity between the Siebel Server and **Outlook**, then **CRM Desktop** uses the following logic:

1 Maps changes between Siebel activities in Outlook and native Outlook items.

   CRM Desktop updates changes in the Outlook Calendar event in the activity that is linked to the Calendar event.

2 Synchronizes changes between Siebel activities in Outlook and Siebel CRM activities.

   CRM Desktop updates this change in Siebel CRM data after the synchronization finishes.

How Siebel CRM Desktop Transforms a Calendar Event That Does Not Repeat

Table 46 describes how Siebel CRM Desktop transforms a Calendar event in Microsoft Outlook that does not repeating.

Table 46. How Siebel CRM Desktop Transforms a Calendar Event That Does Not Recur

Action in Microsoft Outlook	Work That Siebel CRM Desktop Performs
The user creates an activity that does not repeat in Outlook.	Creates a corresponding Siebel CRM activity.
The user changes an activity that does not repeat in Outlook.	Changes the corresponding Siebel CRM activity.
The user deletes an activity that does not repeat in Outlook.	Deletes the corresponding Siebel CRM activity.
The user changes an activity that does not repeat in Outlook to a repeating Calendar event that matches the Siebel repeating pattern.	Changes the corresponding Siebel CRM activity.
The user changes an activity that does not repeat in Outlook to a repeating Calendar event that does not match the Siebel repeating pattern.	CRM Desktop does the following work: ■ Changes the corresponding activity. It uses the Siebel repeating pattern that contains the longest interval between occurrences and that can incorporate all occurrences of the chosen Outlook pattern. ■ To match the calendars in Siebel CRM and in Outlook, it identifies the list of exceptions for the activity.

How Siebel CRM Desktop Transforms a Repeating Calendar Event That Matches a Siebel Repeating Pattern

Table 47 describes how Siebel CRM Desktop transforms a repeating Calendar event that matches a Siebel repeating pattern.

Table 47. How Siebel CRM Desktop Transforms a Repeating Calendar event That Matches a Siebel Repeating Pattern

Action in Microsoft Outlook	Work That Siebel CRM Desktop Performs
The user creates a repeating Calendar event that matches a Siebel repeating pattern.	Creates a corresponding Siebel CRM repeating activity.
The user changes a single occurrence for a repeating Calendar event that matches a Siebel repeating pattern.	CRM Desktop does the following work in Outlook: ■ Adds the date of the exception that changed to the exceptions list for the target activity. ■ Adds the new, nonrepeating activity to the newly created calendar item in Outlook.
The user changes a repeating Calendar event that matches a Siebel repeating pattern.	Changes the corresponding Siebel CRM repeating activity.
The user deletes a single occurrence for a repeating Calendar event that matches a Siebel repeating pattern.	CRM Desktop does the following work in Outlook: ■ Adds the date of the exception to the exceptions list for the target activity. ■ Deletes this single occurrence of the activity.
The user changes a repeating Calendar event that matches a Siebel repeating pattern into the Calendar event that does not repeat.	Changes the corresponding Siebel CRM activity to an activity that does not repeat.
The user changes a repeating Calendar event that matches a Siebel repeating pattern into a repeating Calendar event that does not match any Siebel repeating patterns.	CRM Desktop does the following work in Outlook: ■ Changes the corresponding Siebel CRM activity. It uses the Siebel repeating pattern that contains the longest interval between occurrences and that can incorporate all occurrences of the chosen Outlook pattern. ■ To match Outlook and Siebel calendars, it identifies the list of exceptions for this activity.

How Siebel CRM Desktop Transforms a Repeating Calendar Event That Does Not Match Siebel Repeating Patterns

Table 48 describes how Siebel CRM Desktop transforms a repeating Calendar event that does not match a Siebel repeating pattern.

Table 48. How Siebel CRM Desktop Transforms a Repeating Calendar Event That Does Not Match a Siebel Repeating Pattern

Action in Microsoft Outlook	Work That Siebel CRM Desktop Performs
The user creates a repeating Calendar event that does not match a Siebel repeating pattern.	CRM Desktop does the following work in Outlook: ■ Creates the corresponding activity. To identify this activity, it uses the Siebel repeating pattern that contains the longest interval between occurrences and that can incorporate all occurrences of the chosen Outlook pattern. ■ To match the calendars in Outlook and CRM Desktop, it creates the list of exceptions for this activity.
The user changes a single occurrence for a repeating Calendar event that does not match any Siebel repeating pattern.	CRM Desktop does the following work in the Outlook Calendar: ■ Creates a new calendar item that is nonrepeating. ■ Deletes the occurrence that changed for the Outlook calendar item. ■ Creates an exception in Outlook for the occurrence date that changed. CRM Desktop does the following work in Outlook: ■ Adds an exception to the exceptions dates list of the activity. ■ Adds a new activity that does not repeat.

Table 48. How Siebel CRM Desktop Transforms a Repeating Calendar Event That Does Not Match a Siebel Repeating Pattern

Action in Microsoft Outlook	Work That Siebel CRM Desktop Performs
The user deletes a single occurrence for a repeating Calendar event that does not match a Siebel repeating pattern.	CRM Desktop does the following work in Outlook: ■ Adds an exception to the exceptions dates list of the activity. ■ Deletes this single occurrence of the activity.
The user deletes a repeating Calendar event that does not match a Siebel repeating pattern.	Deletes the corresponding repeating activity.

How Siebel CRM Desktop Transforms Siebel CRM Activities That Do Not Repeat

Table 49 describes how Siebel CRM Desktop transforms a Siebel CRM activity that is not repeated.

Table 49. How Siebel CRM Desktop Transforms a Siebel CRM Activity That Is Not Repeated

Work That Occurs in Siebel CRM	Work That Siebel CRM Desktop Performs
The user creates a Siebel CRM activity that is not repeated.	Creates the corresponding Outlook activity that does not repeat.
The user changes a Siebel CRM activity that is not repeated.	Changes the corresponding Outlook activity that does not repeat.
The user changes a Siebel CRM activity that is not repeated to an activity that is repeated.	
The user deletes a Siebel CRM activity that is not repeated.	Deletes the corresponding Outlook activity that does not repeat.

How Siebel CRM Desktop Transforms Siebel CRM Activities That Repeat

Table 50 describes how Siebel CRM Desktop transforms an activity in Siebel CRM that is repeated.

Table 50. How Siebel CRM Desktop Transforms a Repeated Activity in Siebel CRM

Work That Occurs in Siebel CRM	Work That Siebel CRM Desktop Performs
The user creates a repeating activity in Siebel CRM.	Creates a corresponding repeating Calendar event in Outlook.

Table 50. How Siebel CRM Desktop Transforms a Repeated Activity in Siebel CRM

Work That Occurs in Siebel CRM	Work That Siebel CRM Desktop Performs
The user changes a single occurrence of the repeating activity in Siebel CRM: ■ Creates a delete activity. ■ Creates a new activity that does not repeat.	Deletes the occurrence of the Outlook Calendar event for the date of the occurrence that is deleted in Siebel CRM.
The user can modify a repeating activity in Siebel CRM. The activity was created in Outlook from a repeating Calendar event, so the activity can originate in Siebel CRM or CRM Desktop can synchronize it to Siebel CRM from Outlook. If the user changes all instances of the repeating activity in Siebel CRM, then CRM Desktop deletes every instance of the repeating activity from the current day forward. Any instances that exist before the current day remain on the calendar. If the user deletes only one instance, then Siebel CRM still schedules every other instance. If the user changes a repeating Siebel CRM activity, then CRM Desktop does the following work: ■ Changes the following date of the initial activity to the date minus one occurrence: repeat until ■ Creates a new repeated Calendar event that the date of the change for the repeat determines. This situation is true until the date is the following date of the parent repeated Calendar event: repeat until ■ Relinks the delete records to the corresponding repeating activities, depending on the exception date.	CRM Desktop does the following work: ■ Changes the end date of the initial repeating activity ■ Creates a new repeating activity. The Siebel CRM activity determines this new repeating activity. ■ Changes the links for the delete activities depending on the changes that CRM Desktop synchronizes from the Siebel Server

How Siebel CRM Desktop Maps Fields Between a Siebel Calendar Appointment and a Microsoft Outlook Calendar Appointment

Table 51 describes how Siebel CRM Desktop maps some fields between a repeating Siebel calendar appointment and a repeating Outlook calendar appointment.

Table 51. How Siebel CRM Desktop Maps Fields Between a Siebel Calendar Appointment and a Outlook Calendar Appointment

Siebel CRM		Microsoft Outlook		
Frequency	**Start and End Date**	**Frequency**	**Occurrence**	**Start and End Date**
Daily	Start Date Repeat Until	Daily	Every 1 day	Start is Start date End by is Repeat Until
Weekly	Start Date Repeat Until	Weekly	Every 1 week Weekday is the weekday of the Siebel Start Date	Start is Start date End by is Repeat Until
Monthly	Start Date Repeat Until	Monthly	Every 1 month Day is day of Siebel Start Date	End by is Repeat Until
Quarterly	Start Date Repeat Until	Monthly	Every 3 months Day is day of Siebel Start Date	End by is Repeat Until
Yearly	Start Date Repeat Until	Yearly	Date is date of the Siebel Start Date	End by is Repeat Until

C XML Files Reference

This appendix describes the code in the XML files that Siebel CRM Desktop includes in the customization package. It includes the following topics:

Getting Information About Tags of the Metadata Files

The metadata files that Siebel CRM Desktop uses includes a number of tags that you can modify. XSD files include documentation for many of these tags. To get a copy of these files and to view the documentation that they contain, see Article ID 1541446.1 on My Oracle Support.

XML Code That Maps a Field

This topic describes the code of the siebel_basic_mapping.xml file. It includes the following topics:

For more information, see “Customizing Field Mapping” on page 160.

Example Code of the Siebel Basic Mapping File

The following code is an example of the siebel_basic_mapping.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<sd2_meta>
  <types>
    <type id="Contact" predefined_folder="10">
      <form message_class="IPM.Contact.SBL.Contact" icon="type_image:Contact:16"
large_icon="type_image:Contact:32" display_name="Contact">SBL Contact</form>
        <alt_messageclasses>
          <alt_messageclass ext="Private" display_name="Private Contact"
icon="type_image:Contact.Private:16"
large_icon="type_image:Contact.Private:32">SBL Contact</alt_messageclass>
        </alt_messageclasses>
          <custom_views default_name="Siebel Contacts">
            <view id="all_contacts" name="Siebel Contacts"></view>
          </custom_views>
        <field id=" First Name ">
        <reader class="mapi_std">
          <mapi_tag id="0x3A060000"></mapi_tag>
          <convertor class="string"></convertor>
        </reader>
        <writer class="Microsoft Outlook_std">
          <Microsoft Outlook_field id=" FirstName "></Microsoft Outlook_field>
          <convertor class="string"></convertor>
        </writer>
        </field>
          <field id="Location">
        <reader class="mapi_user">
          <user_field id="sbl Location" ol_field_type="1"></user_field>
          <convertor class="string"></convertor>
        </reader>
        <writer class="Microsoft Outlook_user">
          <user_field id="sbl Location" ol_field_type="1"></user_field>
            <convertor class="string"></convertor>
        </writer>
      </field>
    </type>
  </types>
</sd2_meta>
```

Type Tag of the Siebel Basic Mapping File

The type tag defines the Siebel CRM object that Siebel CRM Desktop maps to Microsoft Outlook.

CRM Desktop includes the following attributes in the type tag:

- **id.** Defines the ID or name of the Siebel CRM object that CRM Desktop maps to Outlook.
- **display_name.** Defines the name of the folder that CRM Desktop displays in the Outlook tree view that CRM Desktop uses to store the records of the Siebel CRM object.

- **folder_type.** Defines the type of the Outlook folder. The value for the folder_type attribute is typically 10 for a custom Siebel CRM object.
- **hidden_folder.** Determines if the folder that the display_name attribute specifies is visible in the Outlook tree view.
- **predefined_folder.** Defines the type of Outlook folder. If you must store Siebel CRM objects in a native Outlook folder, then CRM Desktop uses the predefined_folder attribute.
- **prohibit_user_modification.** Prohibits the modification to an object that you declare in the type tag. The default value is false.
- **ver.** Used during development. CRM Desktop does not apply any change to the description for the object that the id attribute defines until the value of the ver attribute increases.

Form Tag of the Siebel Basic Mapping File

The form tag defines the ID of the form that Siebel CRM Desktop uses to display the object that the type tag defines. The forms_xx.xml file describes the form that includes the corresponding ID. For more information, see "XML Code That Customizes Forms" on page 469.

CRM Desktop includes the following attributes in the form tag:

- **message_class.** Defines the Outlook message class for the form. This message class is an extension of native Outlook message classes. For example, IPM.Contact.SBL.Contact, or IPM.Contact.SBL.Account.
- **icon.** Defines the icon that CRM Desktop uses to display the following objects:
  - The object that the type tag in the Outlook table view defines.
  - The icon views for small icons and list view modes.
  - A form caption icon that displays in the corner in front of the form caption. CRM Desktop stores icons in the platform_images.xml file. The value of the icon attribute must be the key value of the required image.
- **large_icon.** If the user uses the large icon mode, then the large_icon attribute defines the icon that CRM Desktop uses to represent the object that the type tag defines. It stores icons in the platform_images.xml file. The value of the large_icon attribute must be the key value of the necessary image.
- **display_name.** Defines the name that CRM Desktop displays on the form caption.

Alt Message Classes Tag of the Siebel Basic Mapping File

To specify an alternative message class to an object in Outlook, you can use the alt_messageclasses tag. This tag is useful if a CRM object might be in one of several different states. For example, a contact might be shared or not shared.

This tag must include a set of alt_messageclass tags that specify each state of an object. The alt_messageclass tag can also specify a form that you use for an object with this message class. This way, you can use different forms for the same object, but in different states.

Each alt_messageclass tag includes the following attributes:

- **Ext.** Indicates an extension that CRM Desktop adds to an original message class.
- **display_name.** The same as the message_class tag.
- **icon.** The same as the message_class tag.
- **large_icon.** The same as the message_class tag.

Custom Views Tag of the Siebel Basic Mapping File

The custom_views tag defines a set of custom Outlook views that Siebel CRM Desktop applies for the object that the type tag defines. The default_name attribute sets the default view that CRM Desktop applies after the installation.

CRM Desktop includes the following tags in the custom_views tag:

- **view.** Describes each view. Each view tag includes the following attributes:
  - **id.** Defines the ID of the view. The view is described in the views.xml file.
  - **name.** Defines the name of the view that the id attribute specifies to display in the Current view menu. To access this menu in Outlook, the user chooses the View menu, Arrange By, and then the Current view menu.

The following code is an example of the custom_views tag:

```
<custom_views default_name="All Activities">
  <view id="all_activities" name="All Activities"></view>
  <view id="all_activities_by_duedate" name="Activities by Due Date"></view>
  <view id="all_activities_by_owner" name=" Activities by Owner"></view>
  <view id="all_activities_by_priority" name="Activities by Priority"></view></
custom_views>
```

Field Tag of the Siebel Basic Mapping File

The field tag describes the field mapping. It describes one field, so the number of field tags must be the same as the number of fields that are mapped.

The field tag includes the following attributes:

- **id.** Defines the field identifier. For example, the API name of the field. To assign a control to this field on a form, Siebel CRM Desktop also uses the value of this attribute in the forms_xx.xml file.
- **ver.** Used during development. CRM Desktop does not apply any change to the field description until the value of the ver attribute increases.

Writer Tag of the Siebel Basic Mapping File

The writer tag defines write access to the field that Siebel CRM Desktop maps to Outlook. It includes only the class attribute. The following values are available for the class attribute:

- binhex_link
- custom:links_first
- multiwriter
- Microsoft Outlook_document_content
- Microsoft Outlook_document_filename
- Microsoft Outlook_recipients
- Microsoft Outlook_std
- Microsoft Outlook_user

The writer tag can include the following tags:

- link_writer
- resolved_writer
- Microsoft Outlook_field
- user_field
- convertor
- writer

XML Code That Customizes Platform Configuration

This topic describes the XML code that you can use to customize the platform configuration. The following code is an example of the platform_configuration.xml file. For more information, see "Files in the Customization Package" on page 434:

```
<platform>
<items_remover>
<rules>
<type id="Mail" rule="skip" />
<type id="Task" rule="skip" />
<type id="Event" rule="skip" />
<type id="Action" rule="script" language="JScript">
<![CDATA[
  //"Calendar and Activities";
   //"To Do and Activities";
   //"Activities Only";
   var pim_id = item.PIMObjectId;
   if (!item["Appt PIM Flag"] && pim_id != null)
   {
     var pim_item = open_item(pim_id);
     if (pim_item != null)
       pim_item.remove();
   }
 true;  // allow to process this item
]]>
</type>
</rules>
</items_remover>
</platform>
```

XML Code That Customizes Synchronization

This topic describes the code of the connector_configuration.xml file. It includes the following topics:

- Example Code of the Connector Configuration File on page 464

For more information, see "Customizing Synchronization" on page 161.

Example Code of the Connector Configuration File

The following code is an example of the connector_configuration.xml file:

```
<root>
  <types>
    <type id="Opportunity">
      <view label="Opportunity" label_plural="Opportunities"
small_icon="type_image:Opportunity:16" normal_icon="type_image:Opportunity:24"
large_icon="type_image:Opportunity:48"></view>
        <synchronizer name_format=":[:(Name):]">
          <links>
            <link>Account Id</link>
            <link>Currency Code</link>
          </links>
        <natural_keys>
          <natural_key>
            <field>Name</field>
          </natural_key>
        </natural_keys>
        </synchronizer>
      </type>
    </types>
    <filter_presets>
      <preset name="Test filters">
        <type id="Action">
          <group link="and">
            <binary field="Planned" condition="ge">
              <value type="function">today</value>
            </binary>
          </group>
        </type>
    </preset>
  </filter_presets>
</root>
```

Types Tag of the Connector Configuration File

The types tag describes the types of objects to synchronize. It does not contain attributes. It does contain a set of type tags. You must describe these types in the siebel_basic_mapping.xml file.

Type Tag of the Connector Configuration File

The type tag describes the type to synchronize. You must describe it in the siebel_basic_mapping.xml file. It includes the id attribute that defines the ID of the object.

The type tag must contain the following tags:

- view
- synchronizer

View Tag of the Connector Configuration File

The view tag defines the type in the user interface. It defines the Filter Records tab on the Synchronization Control Panel dialog box.

The view tag includes the following attributes:

- **label.** Label that displays this object in the Synchronization Control Panel dialog box if CRM Desktop cannot resolve the name of this object.
- **label_plural**. Label that displays this object in the Synchronization Control Panel dialog box if the label is most appropriately displayed in plural.
- **small icon**. Defines a 16-by-16 pixel icon that CRM Desktop uses for this object on the Synchronization Control Panel dialog box.
- **normal_icon**. Defines the icon that displays next to the object type in the Filter Records tab of the Synchronization Control Panel dialog box.
- **large_icon**. Defines the icon that displays in the CRM Desktop Synchronization dialog box while CRM Desktop synchronizes this type of object.
- **suppress_sync_ui**. If suppress_sync_ui is true, then this attribute hides objects of this type from the Filter Records tab of the Synchronization Control Panel dialog box. If suppress_sync_ui is not defined, then CRM Desktop applies the false value, by default.

For more information, see "Controlling the Object Types That Siebel CRM Desktop Displays in the Filter Records Tab" on page 131.

Synchronizer Tag of the Connector Configuration File

The synchronizer tag describes attributes that the Synchronization Engine requires. For example, it describes relationships between objects or criteria to identify duplicate objects.

The synchronizer tag includes the name_format attribute that defines the format of the output string for objects of this type. Siebel CRM Desktop uses this string if objects of this type are displayed on the Check Issues, Resolve Conflicts, Resolve Duplicates, or Confirm Synchronization tab of the Control Panel in the CRM Desktop add-in.

The synchronizer tag can contain the following tags:

- links
- natural_keys

Links Tag of the Connector Configuration File

The links tag describes the references between types. You must specify it in the synchronizer tag. Note the following requirements:

- You must use the links tag to describe all fields that Siebel CRM Desktop uses to store references between objects. A link field is an example of a field that CRM Desktop uses to store a reference between objects.

- You must describe all links in the links tag.

The links and link tags do not contain attributes.

Example Code of the Links Tag

The following code is an example of the links tag:

```
<links>
  <link>Account Id</link>
  <link>Opportunity Id</link>
  <link>Created By</link>
  <link>Primary Owner Id</link>
</links>
```

Natural Key Tag of the Connector Configuration File

The natural_key tag is defined in the synchronizer tag. You use it to configure criteria to identify duplicated records during synchronization. The natural_key tag includes the following items:

- A set of natural_key tags that describe the criteria. Siebel CRM Desktop uses OR logic for all criteria that the natural_key tag describes.
- A set of field tags. Each of these tags includes a field name that CRM Desktop examines to identify duplicates. CRM Desktop uses AND logic for all field tags.

Example Code of the Natural Key Tag

The following code is an example of the natural_keys tag:

```
<natural_keys>
  <natural_key>
    <field>First Name</field>
    <field>Last Name</field>
  </natural_key>
  <natural_key>
    <field>Email Address</field>
  </natural_key>
</natural_keys>
```

In this code, if one of the following situations is true, then CRM Desktop detects two objects as duplicates:

- First Name AND Last Name contain the same values
- Email Address fields contain the same values

Filter Presets Tag of the Connector Configuration File

The filter_presets tag contains predefined filter criteria. The preset tag describes this criteria. The preset tag includes the name attribute. It defines the name for this criteria. The preset tag contains a set of type tags that specify filter criteria for each type.

The type tag defines the object type. Siebel CRM Desktop applies the filter criteria to this object type. You must specify the object type in the id attribute of this tag. The group tag describes a group of criteria.

Example Code of the Filter Presets Tag

To set the filter on a top-level object, you also need to apply the same filter on all dependent objects. For example, if you want to put a preset filter on Action, you have to define it on Action itself, in addition to the Action Attachment.

The following code is an example usage of the filter_presets tag of the connector_configuration.xml file:

```
<type id="Action">
  <group link="and">
    <binary field="Type" condition="ne">
      <value type="string">To Do</value>
    </binary>
  </group>
</type>
<type id="Attachment">
  <group link="or">
    <collection container_type="Action" foreign_key="ParentId">
      <primary_restriction>

        <group link="and">
          <binary field="FileSize" condition="le">
            <value type="integer">5242880</value>
          </binary>
        <group link="or">
          <binary field="FileExt" condition="eq">
            <value type="string">doc</value>
          </binary>
        </group>
        …and so on
      </primary_restriction>
      <container_restriction>
        <group link="and">
          <binary field="Type" condition="ne">
            <value type="string">To Do</value>
          </binary>
        </group>
      </container_restriction>
    </collection>
  </group>
</type>
```

Example Code That Sets the Size and Type of Field

This topic describes code you can use to set the size and type of field. For more information, see "Controlling the Size and Type of Synchronized Records" on page 139. The following code is an example usage of the group tag of the connector_configuration.xml file to set the size and type of field:

```
<group link="and">
  <binary field="FileSize" condition="le">
   <value type="integer">5242880</value>
   </binary>
   <group link="or">
   <binary field="FileExt" condition="eq">
   <value type="string">doc</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">docx</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">xls</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">xlsx</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">msg</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">txt</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">rtf</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">html</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">ppt</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">pptx</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">pdf</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">mht</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">mpp</value>
   </binary>
   <binary field="FileExt" condition="eq">
   <value type="string">vsd</value>
```

```
    </binary>
    </group>
  </group>
```

XML Code That Customizes Forms

This topic describes the code of the forms_xx.xml file. It includes the following topics:

For more information, see “Customizing Forms” on page 161.

Form Tag of the Forms File

The form tag describes the user interface for each custom Microsoft Outlook form. It can include the following attributes:

- **id.** Defines the unique name for the current form.
- **on_open**. JavaScript code that Siebel CRM Desktop runs if a user opens a form.
- **on_saving**. JavaScript code that CRM Desktop runs if a user attempts to save a form.
- **on_saved.** JavaScript code that CRM Desktop runs if a user saves a form.

The values for each attribute contain the name of the JavaScript function to run on a specific event. An example event is the event that occurs when a user opens a form.

The form tag contains the following tags:

- validation_rules
- script
- info_bar
- page

Each of these tags describe a specific part of the form. You can ignore each tag.

Example Code of the Form Tag

The following code is an example of the form tag. This code defines the Contact Note form:

```
<forms>
  <form id="SBL Contact Note" on_open="form_open()">
    <validation_rules>
      <rule message="Note Type is required." expression="item['Note Type'] != ''">
        <asserted_control id="NoteType"></asserted_control>
      </rule>
    </validation_rules>
    <script>
      <![CDATA[
        function form_open()
        {
          form.NoteType.required = true;
        }
      ]]>
    </script>
      <page id="General" tag="0x10A6" min_height="155" min_width="520">
        <cell>
          <stack layout="vert" padding="5">
            <cell>
              <stack layout="horz" spacing="3">
                <cell size="65">
                  <stack layout="vert" spacing="5">
                    <cell size="22">
                      <control class="static" id="0x20000">
                        <text>Type:</text>
                      </control>
                <cell size="22">
                  <control class="static" id="0x20002">
                    <text>Description:</text>
                  </control>
                </cell>
              </stack>
            </cell>
            <cell>
              <stack layout="vert" spacing="5">
                  <cell size="22">
                    <control class="combobox" id="NoteType" tab_order="1">
                      <source type="Contact.Contact_NoteNote TypePicklist"
field="Value" format=":[:(Label):]"></source>
                      <field>Note Type</field>
                    </control>
                  <cell>
                    <control id="0x103f" tab_order="2"></control>
                  </cell>
                </stack>
              </cell>
            </stack>
          </cell>
        </stack>
```

```
      </cell>
    </page>
  </form>
</forms>
```

Validation Rules Tag of the Forms File

The validation_rules tag includes descriptions of validation rules. There is no limit on the number of validation rules that you can define. Each rule is described in a rule tag.

The validation_rules tag can include the following attributes:

- **message.** Contains the message text as a value. If the validation rule that the rule tag describes fails, then Siebel CRM Desktop displays this message to the user.
- **expression.** Contains the validation expression as a value. If the expression returns a false value, then the validation rule fails.

The validation_rules tag can contain the following tags:

- **expression.** Performs the same function as the expression attribute. The only difference is that if you use the expression tag, then you must describe the validation expression in the CDATA section of the validation_rules tag.
- **asserted_control.** Defines a control to highlight if the validation rule fails. The ID of this control is defined in the id attribute of the asserted_control tag.

Example Code of the Validation Rules Tag

The following code is an example of the validation_rules tag:

```
<validation_rules>
  <rule message="Last Name is required." expression="item['Last Name'] != ''">
    <asserted_control id="0x1000"></asserted_control>
  </rule>
  <rule message="Business phone value format is incorrect.">
    <expression>
      <![CDATA[
        var phonetest = new RegExp(/\(?[0-9]{3}\)?[-. ]?[0-9]{3}[-. ]?[0-9]{4}/);
        item["Work Phone #"] != '' ? (item["Work Phone #"].match(phonetest) != null
? true : false) : true;
      ]]>
    </expression>
  </rule>
</validation_rules>
```

Script Tag of the Forms File

The script tag stores all JavaScript functions that Siebel CRM Desktop uses in the current form. You can use these scripts for different purposes, as required. You must describe all script functions in the CDATA section of the validation_rules tag.

Example Code of the Script Tag

The following code is an example of the script tag:

```
<script>
  <![CDATA[
    function form_open()
    {
      form.NoteType.required = true;
    }
    function form_save()
    {
      if (form.item["Created"] == null)
      {
        form.item["Created"] = (new Date()).getVarDate();
      }
    }
  ]]>
</script>
```

Info Bar Tag of the Forms File

The info_bar tag is an alternative to the page tag. You can use it to add a layout that the original form layout does not determine. You can use the info_bar tag to extend the original form but not to modify the original form.

For example, the predefined email form includes an Info Bar. The Info Bar tag includes the Share With Siebel content that Siebel CRM Desktop displays if the user clicks the Share Bar. The Info Bar does not affect other parts of the form.

Note the following differences between how you can use the info_bar tag and the page tag:

- To customize only a section of a predefined Outlook form, you can use the info_bar tab. For example, on the Mail, task, or Calendar, forms.
- To customize an entire form, you can use the page tag. For example, the page tag completely replaces the native Contact form in Outlook with a custom CRM Desktop form.

Page Tag of the Forms File

The page tag describes the layout of the form. A set of cell tags that contain the user interface elements and the data that makes up the form defines this layout. Note the following requirements for the cell tag:

- A cell tag can be empty or can contain a stack of cell elements in the stack tag or a control.
- A cell tag can contain a user interface element or a piece of data from a native Outlook object and from a standard or custom Siebel CRM object. If you must place more than one object in a cell, then you must use a stack tag in the cell tag.

A page tag can contain only one cell.

The page tag can include the following attributes:

- **id.** Contains the page name and is used nowhere else.
- **tag.** Contains the control ID of the first control on the page of a standard form. It is a page that Siebel CRM Desktop uses to apply a customized layout. For example, the first control on a General tab of a Contact form uses `id=0x402, Details page - 0?11cf`, and so on.
- **min_height.** Describes the minimum height, in pixels.
- **min_width.** Describes the minimum width, in pixels.

Example Code of the Page Tag

The following code is an example of the page tag:

```
<page id="General" tag="0x0402">
  <cell>
  </cell>
</page>
```

Cell Tag of the Page Tag of the Forms File

The cell tag is a layout cell that contains a stack of cells or a control. It defines the position of the stack. It includes the following attributes:

- **size.** Defines the cell size, in pixels. You must specify the size if the cell is situated in a stack of cells:
  - In a stack of cells that is arranged horizontally, the size defines the cell width.
  - In a stack of cells that is arranged vertically, the size defines the height.
- **attraction.** Defines cell docking. You must specify the attraction if the cell is in a stack of cells. The following values are available:
  - **near.** Cell docks to the left side of a horizontal stack of objects or to the start of a vertical stack of objects.
  - **far.** Cell docks to the right side of a horizontal stack of objects or to the end of a vertical stack of objects.
  - **both.** Cell consumes the entire free space. If more than one cell is set to `both`, then Siebel CRM Desktop divides the free space equally between all cells that are set to `both`.

  If the cell size is defined, then the default value is `near`. If there is no size, then the default value is `both`.

Stack Tag of the Forms File

The stack tag defines a stack of cells that is placed in a cell. Siebel CRM Desktop docks a cell that contains the near attribute in the following ways:

- To the starting border in a vertical stack

- To the left border in a horizontal stack

The stack tag includes the following attributes:

- **layout.** Defines the type of stack. Possible values include horz (horizontal) or vert (vertical).
- **spacing.** Defines the space between cells in a stack. The default value is 0.
- **padding.** Defines the space between the cell and the stack border. The cell defines the stack border. This cell contains the stack. The default value is 0.

Example Code of the Stack Tag

The following code is an example of the stack tag:

```
<page id="General" tag="0x0402">
  <cell>
    <stack layout="horz">
      <cell size="25"/>  <!--attraction="near"-->
      <cell size="30"/>
      <cell/>      <!--attraction="both"-->
      <cell attraction="both">
      <cell size="20" attraction="far">
    </stack>
  </cell>
</page>
```

Control Tag of the Forms File

The control tag defines a control that is located in a cell. It includes the following attributes:

- **id.** Defines the control ID. If you use the id attribute to specify the ID of a native Microsoft Outlook control, then Siebel CRM Desktop requires no more attributes except tab_order, if necessary. The class attribute defines the control type.
- **tab_order.** Defines the tab order of the control. CRM Desktop displays the control that contains the smallest tab_order value as the first control in the tab order. You can start at 1. If the control contains no tab_order, then CRM Desktop does not include the control in the tab order. For more information, see Making Sure Tab Order Is Unique on page 476.
- **class.** Defines the type of the control. Depending on the value of the class attribute, you must specify more attributes and tags. For more information, see “Types of Controls for the Control Tag of the Forms File” on page 476.
- **allow_negative.** For multi_currency control only.
- **caption.** For list control only.
- **type.** For Microsoft Outlook_view control only.
- **view_id.** For Microsoft Outlook_view control only.

The following controls support the control tag:

- button

- check box
- gradient_checkbox

The control tag includes the following tags. These tags depend on the value that the class attribute of the parent control tag contains:

- **source.** Used for multivalue controls, such as dropdown list, lookup, and mvg_primary_selector. It defines the objects that CRM Desktop displays in this control. It includes the following attributes:
  - **type.** Defines the object type that CRM Desktop displays in the current control. The basic_mapping.xml file must describe this object type.
  - **display_format.** Defines the object fields that CRM Desktop displays in a control. Used only for the mvg_primary_selector control.
  - **format.** The same as the display_format tag, but applied to other controls.
  - **field.** Defines the object field that CRM Desktop displays as chosen in a dropdown list. Used only for the dropdown list control.
  - **left_id.** Defines a field on a related object where CRM Desktop stores the ID of the parent object. CRM Desktop uses the left_id control on this parent object ID. It uses it only for the mvg_primary_selector control.
  - **item_value.** Defines a field on a related object where CRM Desktop stores the ID of the parent object. This is the ID of the chosen object. It is used only for the mvg_primary_selector control.
- **text.** Defines the text that CRM Desktop uses for the control. You use this tag primarily for the label control.
- **field**. Contains the field identifier that this control uses:
  - The siebel_basic_mapping.xml file must describe this field tag.
  - Contains the value attribute. CRM Desktop uses this attribute only for edit controls. This attribute describes the value type for the field.
  - The following values are available: string, binary, int, double, or currency.
  - The `API name` field is an example of a field identifier.

Example Code of the Control Tag

The following code is an example of the control tag:

```
<cell>
  <control id="0x10000" class="edit">
    <field value="string"> Name</field>
  </control>
</cell>
```

Making Sure Tab Order Is Unique

Make sure that the tab order is unique through the entire form. For example, if you add a field to the Contact form, then adjust the tab_order values for the tab_orders that this form uses. For example, if the tab_order value is 15, then you must set it to 17. You increment it by a value of one for the label and one for the field. This configuration makes sure the user can correctly tab through the form.

Types of Controls for the Control Tag of the Forms File

This topic describes the types of controls you can configure for the control tag of the forms_xx.xml file. It includes the following topics:

Values of the Control Tag of the Forms File

Table 52 describes the values you can specify for the class attribute of the control tag of the forms_xx.xml file.

Table 52. Values You Can Specify for the Class Attribute of the Control Tag

Value	Description
button	A button control. if you use this control, then it is not necessary to use a field tag.
checkbox	A check box control. The field that the field tag assigns to this control must include the following configuration: ■ The name of the field ■ The field type must be Boolean
combobox	A simple control that allows the user to choose any value from a list. For more information, see "Combobox Control of the Forms File" on page 477.
datetime	A datetime control that allows the user to choose a date from a calendar. The field tag must contain the date or datetime field.

Table 52. Values You Can Specify for the Class Attribute of the Control Tag

Value	Description
dropdown	A control that displays a menu when the user clicks the control. For example, if the user clicks Addresses on the contact form, then Siebel CRM Desktop displays Personal Addresses or Business Addresses. For more information, see "Dropdown Control of the Forms File" on page 478.
edge	A panel control that includes a border. If you use this control in a cell where the size is 1, such as with a separator, then it is not necessary to use a field tag.
edit	A simple edit box control.
gradient_checkbox	A control that behaves like a check box control, but uses a different graphical interface. This control displays on the Sharing bar.
lookup	A control that the user can use to choose any object. CRM Desktop uses a lookup control to establish a relationship between objects. For example, to link an account with a contact. For more information, see "Lookup Control of the Forms File" on page 478.
multi_currency	A control that CRM Desktop uses to display the values from more than one field, such as price, revenue, and so on. An example usage of the multi_currency control is where CRM Desktop must display the amount and currency values in a single field. For more information, see "Multicurrency Control of the Forms File" on page 479.
mvg_primary_selector	A control that CRM Desktop uses to display the primary association in a many-to-many (M:M) relationship. For more information, see "MVG Primary Selector Control of the Forms File" on page 479.
static	A static control that you can use as a label on a form. If you use this control, then it is not necessary to use a field tag.
subform	A group of controls that you can use to display the fields of one object on the form of another object. For example, you can display a Siebel CRM activity on the native form for the Outlook task or calendar . For more information, see "Subform Control of the Forms File" on page 480.

Combobox Control of the Forms File

If you set the class attribute of the control tag to combobox, then you must specify the following tags in the control tag:

- **source.** Describes the list values for this control. The source tag includes the following attributes:
  - **type.** Contains the ID of a list that the siebel_basic_mapping.xml file describes.
  - **field.** Contains the name of the field that Siebel CRM Desktop displays as a list value.
  - **format.** Defines the mask for this field output. Attributes are usually the same for all lists that drop down. It uses the following format:

```
field="Value"

format=":[:(Label):]"
```

Although Label is a variable, you must specify it as an absolute value, as described here.

- **field.** A field of an object that stores a value that the user chooses in a list.

Example Code of the Combobox Control

The following code illustrates usage of the combobox control:

```
<control id="0x20105" class="combobox">
  <source type="ContactLeadSourcePicklist" field="Value" format=":[:(Label):]"/>
  <field>LeadSource</field>
</control>
```

Dropdown Control of the Forms File

The dropdown control of the forms_xx.xml file is a button that includes menu options. If the user clicks this button, then Siebel CRM Desktop displays the menu. You add menu items for this menu to a script.

If you set the class attribute of the control tag to dropdown, then you must specify the following tags in the control tag:

- **control.** Contains more tags that you can use to describe the dropdown control.
- **text.** The value in the control tag that you can use to specify the text of the dropdown control.

To specify the caption for the dropdown control, you can use the caption attribute.

Example Code of the Dropdown Control

The following code illustrates usage of the dropdown control:

```
<cell size="22">
  <control id="dd_contacts" class="dropdown" caption="#lbl_contacts" tab_order="1"
visible="false"></control>
</cell>
```

Lookup Control of the Forms File

If you set the class attribute of the control tag to lookup, then you must specify the following tags in the control tag:

- **source.** Describes the list values for this lookup control. The source tag includes the following attributes:
  - **type.** Contains the ID of an object that Siebel CRM Desktop uses in this control.
  - **field.** Not used.
  - **format.** Defines the mask for this field output. For example:

```
format=":[:(First Name) :]:[:(Last Name):]"
```

- **resource_id.** Defines the ID of the description for the lookup dialog box that the lookup_view_defs.xml file describes. For more information, see "Customizing the SalesBook Control" on page 163.

- **field.** The field of an object that stores the ID of the object that the user chooses in a lookup object.

Example Code of the Lookup Control

The following code illustrates usage of the lookup control:

```
<control id="0x20100" class="lookup">
  <source type="All Items" format=":[:(FirstName) :]:[:(LastName):]"
resource_id="lookup:all_types"/>
  <source type="Contact" format=":[:(FirstName) :]:[:(LastName):]"
resource_id="lookup:contacts"/>
</control>
```

Multicurrency Control of the Forms File

If you set the class attribute of the control tag to multi_currency, then you must specify the following tags in the control tag:

- **value_field.** Contains the amount field name.
- **currency_field.** Contains the currency field name.
- **exchangedate_field.** Contains the exchange date field value. This tag is optional.

Example Code of the Multi Currency Control

The following code illustrates usage of the multi_currency control:

```
<control id="1" class=" multi_currency " tab_order="1">
  <value_field>Revenue</value_field>
  <currency_field>Currency</currency_field>
  <exchangedate_field>Date</exchangedate_field>
</control>
```

MVG Primary Selector Control of the Forms File

If you set the class attribute of the control tag to mvg_primary_selector, then you must specify the following tags in the control tag:

- **source.** Behavior is similar to the lookup control. The source tag includes the following attributes:
  - **type.** Defines the many-to-many association ID that Siebel CRM Desktop uses for this control.
  - **linking_field.** Contains the field name of this association where the ID of the parent object is saved.
  - **flag_field.** Contains the field name that CRM Desktop uses to set the primary flag.

- **display_format.** Defines the output mask.

Subform Control of the Forms File

The following code illustrates usage of the subform control:

```
<cell size="22">
  <control class="subform" id="activity_subform">
    <cell size="22">
      <stack layout="horz" spacing="20" padding="6">
        <cell>
          <control class="static" id="ActivityLabel">
            <text>Activity Name:</text>
          </control>
        </cell>
        <cell>
          <control class="edit" id="ActivityName">
            <field>Name</field>
          </control>
        </cell>
      </stack>
    </cell>
  </control>
</cell>
```

Web Page Control of the Forms File

If you set the class attribute of the control tag to web_page, then you must specify the url attribute in the control tag. The following code illustrates usage of the web_page control:

```
<control class="web_page" id="linkedin_search">
    <url>http://www.linkedin.com/</url>
  </control>
```

You can specify a static or a dynamic URL as the value of the url attribute. If the URL is dynamic, then JavaScript supports it. For example, you can present a dynamic personal page for a business contact on the Contact form. The following is an example of this JavaScript:

```
if (!is_new)
  form.linkedin_search.navigate = "http://www.linkedin.com/pub/dir/?last=" +
form.item['Last Name'] + "&first=" + form.item['First Name'];
```

If the url attribute is not set, then Siebel CRM Desktop loads the about:blank page, by default.

XML Code That Customizes Toolbars

This topic describes the code of the toolbars.xml file. It includes the following topics:

For more information, see “Customizing Toolbars” on page 162.

Example Code of the Toolbars File

The following code is an example of the toolbars.xml file:

```
<button id="meeting_with_contact" name="#btn_meeting_with_contact"
small_image="orcl_meeting_with_contact:16">
  <action class="scriptable" id="meeting_with_contact"/>
</button>
```

Toolbars Tag of the Toolbars File

The toolbars tag is the root tag of the toolbars.xml file. It describes the toolbar that Siebel CRM Desktop adds to a native Outlook form or to the Outlook window.

The caption attribute of the toolbars tag defines the toolbar caption.

The toolbars tag includes the following tags:

- **for.** Determines if CRM Desktop displays this tag in the Outlook window or the Outlook form, depending on if the value of the tag is explorer or inspector.
- **button.** A description of a button on a toolbar. For more information, see "Button Tag of the Toolbars Tag of the Toolbars File" on page 481.

Button Tag of the Toolbars Tag of the Toolbars File

The button tag includes the following attributes:

- **name.** The caption for the button.
- **small_image.** The resource ID of the icon that Siebel CRM Desktop uses as the small icon for this button.
- **large_image.** The resource ID of the icon that CRM Desktop uses as a large icon for this button. The large_image attribute is valid for Microsoft Outlook 2007 but CRM Desktop ignores it for Microsoft Outlook 2003.
- **begin_group.** Determines if CRM Desktop displays the separator of the toolbar button for this button. It is useful if you must group toolbar buttons.

Action Tag of the Button Tag

The button tag includes the action tag. This tag defines the action that Siebel CRM Desktop calls if the user clicks the button. You can use a predefined action or write a custom action. You must set this action in the class attribute of the action tag.

The action tag includes the class attribute. The class attribute can include the following values:

- **create_attachment.** Opens the Select Attachment dialog box that allows the user to choose a file to attach to the current object. You can use the accept_type attribute to specify the object that can include an attachment. The accept_type attribute defines the button visibility, depending on the object type that is currently chosen. The value of the accept_type attribute must be the object type that you must make visible for this button.

- **create_linking_item.** Creates a new object and associates it with the current object. You can also specify the object types where CRM Desktop displays this button. The type tags that reside in a types tag must describe these object types.

The action tag includes the attachment tag. To specify attachment settings, you must specify the following attributes of the attachment tag:

- **type.** The type of the attachment object.
- **name_field.** The name of the field of the attachment object where CRM Desktop stores the name of the file.
- **body_field.** The name of the field of the attachment object where CRM Desktop stores the body of the file.
- **linking_field.** The name of the field of the attachment object where CRM Desktop stores the reference to the parent object.

Example Code of the Action Tag of the Scriptable Action

Siebel CRM Desktop supports the scriptable action class. The action tag of the scriptable action includes the id attribute. You can use the id attribute in a script to specify the action to perform. The following example specifies a button with a scriptable action:

```
<button id="new_account" name="#btn_new_account">
  <action class="scriptable" id="new_account"/>
</button>
```

In this example, CRM Desktop passes the value for the new_account attribute to the script when it handles the click event of the button. The script includes predefined logic that the new_account attribute starts.

XML Code That Customizes Dialog Boxes

This topic describes the code of the dialogs.xml file. It includes the following topics:

For more information, see “Customizing Dialog Boxes” on page 162.

Dialog Tag of the Dialogs File

The dialog tag describes each dialog box. It is similar to the form tag of the forms_xx.xml file but it does not support the on_saved and on_saving attributes. The dialog tag in the dialogs.xml file behaves in the same way as the form tag in the forms_xx.xml file except for the following differences:

- The dialog box description includes the layout tag and the appearance tag.

- The dialogs.xml file does not contain a validation_rules tag.
- You cannot use native Outlook controls in the dialogs.xml file.

Layout Tag of the Dialogs File

The layout tag is similar to the page tag of the forms_xx.xml file but it includes different attributes. This includes the following attributes:

- **sizable.** Determines if the user can change the size of the dialog box.
- **visible.** Sets dialog box visibility during creation. If the visible attribute is set to false, then Siebel CRM Desktop creates the dialog box in the background. To make it visible, you must use JavaScript code that changes the value for this attribute.
- **caption.** Defines the caption for the dialog box.
- **small_icon.** Defines the icon that displays next to the caption. The value of this attribute must be an ID of an image resource from CRM Desktop.

Appearance Tag of the Dialogs File

The appearance tag defines the position and size of the dialog box. It includes the following attributes:

- **height.** Defines the height of the dialog box, in pixels.
- **width.** Defines the width of the dialog box, in pixels.
- **position.** Defines the position of the dialog box in a screen. The position attribute can contain the following values:
  - **parent_center.** Displayed on the center of a parent window.
  - **desktop_center.** Displayed on the center of the desktop.
  - **custom.** A custom position.
  - **top.** The starting position of the dialog box. The position attribute must include a custom value.
  - **left.** The left position of the dialog box. The position attribute must include a custom value.

XML Code That Customizes Views

This topic describes the code of the views.xml file. This file includes the set of str tags that specify the configuration of the Outlook view. The only important attribute of this tag defines the unique name, or ID, for the view. For more information, see "Customizing Views" on page 163.

The following code is an example of the views.xml file:

```
<res_root>
  <str key="sample_view">
    <![CDATA[<?xml version="1.0"?>
<view type="table">
  <viewname>Sample view</viewname>
  <linecolor>8421504</linecolor>
  <linestyle>3</linestyle>
  <gridlines>1</gridlines>
  <newitemrow>0</newitemrow>
  <usequickflags>0</usequickflags>
  <collapsestate/>
  <previewstyle>color:Blue</previewstyle>
  <arrangement>
    <autogroup>0</autogroup>
    <collapseclient/>
  </arrangement>
  <column>
    <name>HREF</name>
    <prop>DAV:href</prop>
    <checkbox>1</checkbox>
  </column>
  <column>
    <maxrows>4294901760</maxrows>
    <heading>Organization</heading>
    <prop>urn:schemas:contacts:sn</prop>
    <type>string</type>
    <width>987</width>
    <style>padding-left:3px;;text-align:left</style>
    <editable>1</editable>
    <userheading>Organization</userheading>
  </column>
  <orderby>
    <order>
      <heading>Organization</heading>
      <prop>urn:schemas:contacts:sn</prop>
      <type>string</type>
      <userheading>Organization</userheading>
      <sort>asc</sort>
    </order>
  </orderby>
  <multiline>
    <width>0</width>
  </multiline>
</view>]]>
  </str>
</res_root>
```

XML Code That Customizes the SalesBook Control

This topic describes the code of the lookup_view_defs.xml file. It includes the following topics:

For more information, see "Customizing the SalesBook Control" on page 163.

Example Code of the Lookup View Definitions File

The following code is an example of the lookup_view_defs.xml file:

```
<res_root>
  <array key="all_lookup_types">
    <item value="Account"></item>
    <item value="Contact"></item>
    <item value="Opportunity"></item>
  </array>
  <lookup_view_def key="lookup:contacts">
    <display name="Contacts"></display>
    <filter dasl="[http://schemas.microsoft.com/mapi/proptag/0x001A001E] &gt;=
'IPM.Contact.SBL.Contact' AND [http://schemas.microsoft.com/mapi/proptag/
0x001A001E] &lt;= 'IPM.Contact.SBL.Contact'"></filter>
    <view id="contacts:salesbook"></view>
    <quick_lookup dasl_format="[http://schemas.microsoft.com/mapi/id/{00062004-
0000-0000-C000-000000000046}/8005001E] = '%s'"></quick_lookup>
    <type id="Contact"></type>
  </lookup_view_def>
</res_root>
```

Array Tag of the Lookup View Definitions File

The array tag defines a set of types that is available for the SalesBook control. The user cannot use the SalesBook control to choose an object type until you describe this type in the array tag. Also, you must specify the type ID as a value attribute of the item tag.

The following code is an example of the array tag:

```
<array key="all_lookup_types">
  <item value="Account"></item>
  <item value="Contact"></item>
  <item value="Opportunity"></item>
</array>
```

Lookup View Definition Tag of the Lookup View Definitions File

The lookup_view_def tag describes the configuration for the SalesBook control. You can specify as many configurations as you require. The key attribute defines the unique ID, or name, for this configuration.

The lookup_view_def tag includes the following tags:

- **display.** The name attribute of the display tag defines the name of this configuration. Siebel CRM Desktop displays it as a list value in the corner of the SalesBook control.
- **filter.** The dasl attribute of the filter tag describes the dasl filter that CRM Desktop applies to all objects that the array tag describes. The user can only view the objects that match this filter in the SalesBook control.
- **view.** The id attribute of the view tag defines the view that CRM Desktop applies to the list in the SalesBook control. You must describe this view ID in the views.xml file.
- **quick_lookup.** The dasl_format attribute of the quick_lookup tag defines the dasl filter that CRM Desktop applies to the quick search feature of the SalesBook control. This feature allows the user to enter any text to filter records to simplify finding a field. The user enters text in the edit box on a SalesBook form.

  The following example code allows the user to view records if the File As field is the same as the string:

  ```
  <quick_lookup dasl_format="[http://schemas.microsoft.com/mapi/id/{00062004-
  0000-0000-C000-000000000046}/8005001E] = '%s'"></quick_lookup>
  ```

  where:

  - The *File As* field is ([http://schemas.microsoft.com/mapi/id/{00062004-0000-0000-C000-000000000046}/8005001E]).
  - The *quick search* is entered as ('%s').
- **type.** The ID attribute of the type tag defines the type of object that CRM Desktop creates if the user clicks New in a SalesBook control. If you do not specify this attribute, then the user cannot create a new object.

XML Code That Provides Meta Information

This topic describes the code of the siebel_meta_info.xml file. It includes the following topics:

- Open With URL Template Tag of the Siebel Meta Information File on page 491
- Picklist Tag of the Siebel Meta Information File on page 492
- Master Filter Expression Tag of the Siebel Meta Information File on page 492

For more information, see "Customizing Meta Information" on page 164.

Siebel Meta Info Tag of the Siebel Meta Information File

The SiebelMetaInfo tag is a root tag. It does not contain attributes.

Common Settings Tag of the Siebel Meta Information File

The common_settings tag does not contain tags. It can contain subtags that you can use to specify common options for the Web Service Connector. Siebel CRM Desktop supports the following subtags:

- **max_commands_per_batch.** Defines the maximum number of commands that CRM Desktop can place in a single batch. If CRM Desktop cannot interpret the value of this tag as a positive integer value, then it does not apply any restrictions on the number of commands.
- **max_ids_per_command.** Defines the maximum number of object IDs that CRM Desktop can specify in a search specification for each independent request if a user performs a query by ID. It is the maximum number of record IDs that can be related to the parent record.

For example:

```
<common_settings>
  <max_commands_per_batch>50</max_commands_per_batch>
  <max_ids_per_command>50</max_ids_per_command>
</common_settings>
```

Object Tag of the Siebel Meta Information File

You can use the object tag to specify an object type that Siebel CRM Desktop supports. Table 53 describes the tags that CRM Desktop supports.

Table 53. Tags of the Object Tag of the siebel_meta_info.xml File

Tag	Type	Description
EnableGetIDsBatching	Binary: yes or no	Allows or disallows batching for IDs for each command. The following values are valid: ■ **False.** Disallows ID batching for a get command. ■ **True.** Allows ID batching for a get command.
IntObjName	String	Name of the integration object that CRM Desktop uses for requests.
IsAssociation	Binary: yes or no	Indicates if this type of object is an association object.
IsCascadeDelete	Binary: yes or no	Not currently used.
IsTopLevel	Binary: yes or no	Indicates if a request for this type of object must be wrapped in a request for an object of some parent type.
Label	String	Label that CRM Desktop uses for this type of object in the user interface.
LabelPlural	String	Plural label that CRM Desktop uses for this type of object in the user interface.
SiebMsgXmlCollectionElemName	String	Name of collection XML element that CRM Desktop uses in a Siebel message.
SiebMsgXmlElemName	String	Name of the XML element that CRM Desktop uses in a Siebel message.
SyncFrequency	Numeric	Identifies how often CRM Desktop synchronizes the type, measured in seconds. If you set SyncFrequency to: ■ **0.** CRM Desktop synchronizes the type during every synchronization session. ■ **A positive integer.** CRM Desktop queries the Siebel Server for the records of this type in the time interval you define, starting from the time the type was last queried.
TypeId	String	Unique ID of this type of object.

Table 53. Tags of the Object Tag of the siebel_meta_info.xml File

Tag	Type	Description
UpsertBusObjCacheSize	Numeric	Request attribute that defines the cache size for each upsert operation for each object type. The Siebel API uses this information. The following values are valid: ■ **5.** Default value for all types. ■ **0.** Special value that you can use to resolve a problem that might exist with primaries.
ViewMode	String	Default ViewMode for this type of object.

Field Tag of the Siebel Meta Information File

You can use the field tag to specify an object field. You must nest the field tag in the definition of an object type. Table 54 describes the tags that Siebel CRM Desktop supports.

Table 54. Tags of the Field Tag of the siebel_meta_info.xml File

Tag	Type	Description
DataType	String	Indicates the data type. The Web Service Connector uses the value for this attribute to do data conversion.
HasPicklist	Binary: yes or no	Indicates if this field is a bounded list.
IOElemName	String	Name of the XML element that CRM Desktop uses in Siebel messages for values from this field.
IsCompositeId	Binary: yes or no	Not currently used.
IsFake	Binary: yes or no	If the value for this tag is yes, then CRM Desktop does not use the value from this field in any requests to the API.
IsFilterable	Binary: yes or no	Indicates if this field is available to choose a filter expression on the control panel.
IsMVGField	Binary: yes or no	Not currently used.
IsNullable	Binary: yes or no	The Synchronization Engine uses this tag to break a circular reference.
IsPartOfUserKey	Binary: yes or no	Indicates if this field is part of a user key.

Table 54. Tags of the Field Tag of the siebel_meta_info.xml File

Tag	Type	Description
IsPrimaryKey	Binary: yes or no	Indicates if CRM Desktop uses the value from this field as the primary key for the object. Only one field on an object type can be marked as the primary key.
IsReadonly	Binary: yes or no	Not currently used.
IsRefObjId	Binary: yes or no	Indicates if CRM Desktop uses this tag as a foreign key field. The Web Service Connector and the Synchronization Engine use this tag.
IsRequired	Binary: yes or no	Not currently used.
IsTimestamp	Binary: yes or no	Indicates if CRM Desktop uses the value from this field as an object timestamp. You can specify only one timestamp for each type of object.
Label	String	The label that displays in the user interface.
Name	String	Unique name of the field. If the IsFake tag for this field is set to no, and if this field is not present in requests to the Siebel Server, then this name must be identical to the field name of the Integration Component from the API.
OrderNumber	Numeric	Indicates the order number of this field. The Web Service Connector uses this tag internally. If several fields are defined that hold a reference to a parent record, then their order numbers must reflect the nesting order of the parent types. This situation occurs if a relationship exists between two parent types where one of the parent types is nested in the other parent type.
PicklistCollectionType	String	Type of list items.
PicklistIsStatic	Binary: yes or no	Indicates if the associated list is static or dynamic.
PicklistTypeId	String	Name of the type of list object. You must use the picklist tag to specify the list in the siebel_meta_info.xml file.
RefObjIsParent	Binary: yes or no	Indicates if the object type that is referenced is a parent. The Web Service Connector uses this tag connector to build the hierarchy for the object type.
RefObjTypeId	String	The name of the object type that this field references. The siebel_meta_info.xml file must define this object type.

Extra Command Options Tag of the Siebel Meta Information File

To specify extra options that Siebel CRM Desktop passes to a command element on each request, you can use the extra_command_options tag in the definition of an object type. To specify each tag, you can use the option subtag with the following tags:

- **Name.** Name of the extra command attribute.
- **Value.** Value of the extra command attribute.

Open With URL Template Tag of the Siebel Meta Information File

You can use the open_with_url_tmpl tag in the definition of an object type. You use this tag to specify a template that Siebel CRM Desktop uses to open records of this object type in the Siebel Web Client. For more information, see "Setting the URL That Siebel CRM Desktop Uses to Open the Siebel Web Client" on page 109.

The open_with_url_tmpl tag includes the following format:

```
<open_with_url_tmpl>
  <![CDATA[
  "URL template"
  ]]>
</open_with_url_tmpl>
```

CRM Desktop uses macros in the CDATA section for the attributes that it uses in the open_with_url_tmpl template. Each attribute includes the following format:

```
:[:(attribute):]
```

A series of attributes in the command uses the following format. All brackets, parentheses and colons are required:

```
:[:(protocol):]://
:[:(hostname):]:[:(port):][:(own_id):][:(language_code):][:(parent_id):]
```

where:

- *protocol* is automatically replaced with the URL protocol. The value is http or https.
- *hostname* is automatically replaced with the name of the Siebel Server.
- *port* is automatically replaced with the port number of the Siebel Server.
- *own_id* is automatically replaced with the object ID that CRM Desktop uses to open the Siebel Web Client.
- *language_code* identifies the language. For example ENU. It is part of the URL. For example, /sales_enu/.
- *parent_id* identifies the Id of the parent object. For example, the object Id of the parent account object of the Account_Note type.

CRM Desktop replaces each macro that exists in the command with the actual value at runtime. A CDATA section must enclose the template definition.

For example, you can use the following code for account objects:

```
<open_with_url_tmpl>
  :[:(protocol):]://:[:(hostname):]::[:(port):]/sales/_enu/
  SWECmd=GotoView&SWEView=Account+List+View&SWERF=1&SWEBU=1&SWEApplet0=
  Account+Entry+Applet&SWERowId0=:[:(own_id):]
  ]]>
</open_with_url_tmpl>
```

Picklist Tag of the Siebel Meta Information File

You can use the picklist tag to specify a static list. Table 55 describes the tags that Siebel CRM Desktop supports.

Table 55. Tags of the Picklist Tag of the siebel_meta_info.xml File

Tag	Type	Description
TypeID	String	Unique name for the list.
SrcObjectTypeId	String	Name of the object type that CRM Desktop uses to retrieve items for this list. The siebel_meta_info.xml file must define this object type.
CollectionTypeFldName	String	Name of the field on the original object that contains the type for the list items.
ValueFldName	String	Name of the field on the original object that CRM Desktop uses to retrieve values for the list items.
LabelFldName	String	Name of the field on the original object that CRM Desktop uses to retrieve labels for the list items.
LangFldName	String	Name of the field on the original object that contains the language code.

Master Filter Expression Tag of the Siebel Meta Information File

You can nest the master_filter_expr tag in the definition of a list. You can use this tag to specify a filter expression that Siebel CRM Desktop applies to any request for items of this list. You must use a CDATA section to enclose the value for this tag and you must display the values for the search specification attributes of the target object type.

Glossary

access control
The set of Siebel CRM mechanisms that control the records that the user can access and the operations that the user can perform on the records.

account
A financial entity that represents the relationships between a company and the companies and people with whom the company does business.

account team
Users who possess access to the account record. A user who is assigned to the account is a member of the account team.

ActiveX
A loosely defined set of technologies developed by Microsoft for sharing information among different applications.

ActiveX control
A specific way to implement ActiveX technology. It denotes reusable software components that use the component object model (COM) from Microsoft. ActiveX controls provide functionality that is encapsulated and reusable to programs. They are typically, but not always, visual in nature.

activity
Work that a user must track. Examples include a to-do, email sent to a contact, or a calendar appointment with a contact.

activity (Siebel CRM)
An object in the Action business component of the Siebel data model that organizes, tracks, and resolves a variety of work, from finding and pursuing an opportunity to closing a service request. An Activity also captures an event, such as scheduling a meeting or calendar appointment that occurs at a specific time and displays in the calendar.

activity template (Siebel CRM)
An activity that is defined in an activity template. While the activity for a template is stored in the same object as a transactional activity, this document uses a different term. A template activity essentially behaves like reference data and contains a subset of the attributes for an activity, plus some more attributes that are only relevant to being part of a template.

calendar appointment (Microsoft Outlook)
A record in the Microsoft Outlook calendar or Siebel Web application calendar that reserves time to do something, such as a calendar appointment to schedule a meeting with a customer or to reserve time to complete work in a given time frame.

attendee (Microsoft Outlook)
A person included in the calendar appointment, such as an organizer or a participant.

authentication
Process of verifying the identity of a user.

business component
A logical representation of one or more Siebel tables that usually contains information for a particular functional area, such as opportunity, account, contact, or activity. A business component can be included in one or more business objects.

business object
A logical representation of CRM entities, such as accounts, opportunities, activities, and contacts, and the logical groupings and relationships among these entities. A business object uses links to group business components into logical units. The links provide the one-to-many relationships that govern how the business components interrelate in this business object. For example, the opportunity business object groups the opportunity, contact, and activities business components.

business object (activity)
The object that is the parent of or related to the activity. For example, a service request, opportunity, marketing campaign, order orchestration process, and so on.

business object (interaction)
The object that is the focus of the communication between the customer and the organization. For example, a service request, opportunity, contract, and so on.

child business component
A business component that represents the many in the one-to-many relationship between two business components in a parent-child relationship.

child record
An instance of the child business component.

client computer
The computer that the Siebel CRM Desktop user uses. This is the computer where you install the CRM Desktop add-in.

consumer
In Siebel CRM, a consumer is a person with a party of usage type Customer. In Outlook, a consumer is visible from the Contacts folder and is flagged with the Customer check box.

contact
A person with whom a user might be required to phone or email to pursue a selling relationship. Various business objects can refer to a contact, and this does not require a relationship between the customer and contact. In Siebel CRM, a contact attribute in the context of a business object is a party that might or might not have a relationship defined. In Outlook, a contact attribute in the context of a business object is the same as the Contact folder. Therefore, a contact can be a consumer and can also be an employee of an organization.

contact points
Methods of contacting a contact other than through a postal address, such as such as email, telephone, and fax.

GlobalObjectId

An attribute on a calendar appointment record in Outlook that the user can use to correlate a shared calendar appointment between meeting attendees. Meeting attendees in Outlook include their own copy of the calendar appointment, but all copies include the same value for the GlobalObjectId.

CRM (Customer Relationship Management)

A software application that helps a business track customer interactions.

CRM contact

A contact who uses a user interface where the interface uses a CRM style and is shared with CRM.

CRM Desktop add-in

The technology for Siebel CRM Desktop that resides on the client computer that is provided in the form of a Microsoft Outlook add-in. The Microsoft Outlook add-in performs important work, including storing and displaying Siebel CRM data in native Outlook, and synchronizing PIM and nonPIM data with the Siebel Server.

See also "Microsoft Outlook add-in" on page 498.

current view

The Microsoft Outlook view that displays content from the Outlook folder that is currently chosen.

custom view

A view that a user creates to control the amount of detail that displays in a particular folder. The user can create a filter or change the order of the columns and how the columns are arranged in the new custom view.

customer

A party with whom a user maintains a selling relationship. This party can be an organization or a person. Various business objects can refer to a customer. In Siebel CRM, a customer attribute in the context of a business object can be a person or an organization that includes the party usage type of Customer. In Outlook, a customer attribute in the context of a business object can be an organization or a contact that is flagged as a consumer.

customer team

A group of several employees from the deploying organization or partners who actively work with a customer, including nonsales personnel, such as product marketing, partners, or customer service. The customer team provides the ability to control the visibility of the customer information by associating a person with a business object.

customization

The process of modifying Siebel CRM Desktop to meet the specific requirements of your organization.

customization package

A logical collection of metadata files that is associated with a particular responsibility. A customization package is deployed to the client computer.

cyclical synchronization

A potential synchronization problem when two or more synchronizations form a circular loop. A cyclical synchronization occurs when a single transaction repeatedly loops between servers.

data synchronization
The process of checking for differences between two or more different sets of data, then updating the data sets so that the data in each set is consistent.

DHTML
Dynamic HTML, a combination of technologies that you use to create dynamic Web sites. It can be a combination of HTML 4.0, Style Sheets, and JavaScript.

direct link
A type of link that possesses a one-to-one relationship between one object type and another object type. A link between one account and one opportunity is an example of a direct link.

Dynamic HTML (DHTML)
See DHTML.

encryption
The method of encoding data for security purposes.

form
A generic concept that Microsoft Outlook uses to present information about a single record and data related to that record in a form layout. Each control in the form is a separate attribute or collection of related data. A form can also support different tabs so that details of a child record can be displayed as separate lists.

hash value
A fixed-size string that is obtained as a result of cryptographic transformation from a cryptographic hash function.

homepage
A user interface component in Microsoft Outlook that displays a collection of information from Outlook and CRM applications, and potentially external Web content that is embedded.

household
Provides a way to group consumers.

inbound Web service
A Web service that the Siebel Server makes available.

integration object instance
Data that is organized in the format or structure of the integration object. It is also referred to as a Siebel message object.

interaction (Siebel CRM)
The tracking of customer communications with an organization in the context of the channels that this communication uses and the business objects that they reference. An interaction can take the form of a phone call, email, chat request, Web collaboration, or communication through another channel. An interaction in Siebel CRM Desktop provides an historical view of the communication that occurred. For example, Sales uses interactions to capture communications with a customer during the sales cycle. To pursue an opportunity, a user can log a call as an interaction that the representative made.

installation package

An installation executable that includes the application binaries and any necessary instructions for completing the customization package installation in Microsoft Outlook. It also includes details that are required to connect the application server for the initial synchronization.

lead

An unqualified sales opportunity that often represents the first contact in the opportunity management process. After a lead is qualified it can be converted to an opportunity.

list view

A generic concept in a PIM application that presents information in a list. Each row in the list is a separate record and each column in the list is a separate field in the record.

lookup control

A control that is available in Microsoft Outlook that allows the user to view records in a list, and then choose one or more records to associate with the current item. To identify the subset of data that the user can choose, a lookup control typically includes the capability to specify a search condition.

meeting

A calendar appointment in Microsoft Outlook that includes at least one participant.

metadata files

XML files that hold information on how the user experience must be shaped. The CRM Desktop add-in uses metadata files to do field mapping with the user interface, look ups in the user interface, application object mapping, and general representation of the user interface.

offline

A mode that the user can use with Siebel CRM Desktop but where the user cannot access the Siebel Server. When in offline mode, CRM Desktop uses data in the local data to perform operations. Synchronization is delayed until the user is online.

online

A mode that the user can use with Siebel CRM Desktop while connected to the Siebel Server. The user can synchronize data with the Siebel Server at regular intervals or when the user performs an update. Similar to offline mode, when in online mode CRM Desktop uses data in the local data store to perform operations.

opportunity

A qualified sales engagement that represents potential revenue where a sales representative is willing to officially commit to the pipeline and to include revenue in the sales forecast. The sales representative monitors the opportunity life cycle. This representative might be compensated depending on the results of cumulative sales and potentially how well the representative maintains details about the opportunity.

organization team

Includes the sales groups who possess ownership of the associated prospect, customer, or products with the opportunity, or who are involved for a particular size of deal or with a specific sales stage, and partner organizations that can help close the deal.

organizer

In Microsoft Outlook, the person who created the calendar appointment.

Microsoft Outlook data

Data that is created in the native Microsoft Outlook application.

Microsoft Outlook folder

A folder in Microsoft Outlook that contains a collection of data, such as email messages in the Inbox folder, or sent email messages in the Sent Items folder. In the context of this book, the Outlook folder might also contain Siebel CRM data.

Microsoft Outlook object

An entity that is native to Microsoft Outlook. Examples of Outlook objects include an email, calendar appointment, contact, and so on.

Microsoft Outlook add-in

A program that performs important work, including storing and displaying Siebel CRM data in native Microsoft Outlook and synchronizing PIM and nonPIM data with the Siebel Server.

See also PIM; Siebel Server.

Microsoft Outlook portlet

A portlet that uses data in a Microsoft Outlook folder that includes a custom view filter. The Outlook portlet includes ActiveX characteristics.

Microsoft Outlook standard view

A default Microsoft Outlook view that exists without Siebel CRM Desktop. The Outlook view provides different ways of viewing the same information in a folder by placing the information in different arrangements and formats.

mvg link

A type of link that possesses a one-to-many relationship between one object type and another object type. A link between one opportunity and many contacts is an example of an MVG link.

parent business component

A business component that provides the one in a one-to-many relationship between two business components in a parent-child relationship.

parent record

An instance of the parent business component.

parent-child relationship

The relationship between the parent business component and the child business components that are related to the parent.

participant

In native Microsoft Outlook, the person who is invited to the meeting.

participant of interaction

The people who participate in an interaction. The participant can include an internal representative of the organization, such as a resource, agent, sales representative, and so on. The participant can also include an external representative, such as a customer, contact of a customer, account, or a site. In a help desk or in an employee self-service application, a participant can be an employee.

personalization
The process where the user tailors the user interface and behavior of Microsoft Outlook.

PIM
Personal Information Manager. An application that typically helps a user to manage a list of contacts, calendar entries, email, and so on. Microsoft Outlook, Google email, and Thunderbird are examples of PIMs.

personal information manager (PIM)
See PIM.

PIM data
Personal information that refers to data that is stored in native Microsoft Outlook that relates to a contact, calendar appointment, and so on.

portlet
A user interface component that is managed and displayed in the home page. The home page is composed of multiple portlets.

position
An entity in the Siebel data model. The user position determines the records that the user can view and the operations that the user can perform on the records in a given Microsoft Outlook view.

property set
A logical memory structure that Siebel CRM Desktop uses to pass data between business services. Siebel EAI data is represented in the property set.

recipient
The person who receives an email.

record
A specific instance of the business component, also known as a CRM record, or an object in native Microsoft Outlook, also known as the Outlook record.

responsibility
An entity in the Siebel data model that determines the views that the user can access in Microsoft Outlook. For example, the responsibility of the sales representative allows the user to access the My Opportunities view, whereas the responsibility of the Siebel CRM developer allows the user to access administration views. A Siebel CRM developer or system administrator defines the responsibilities.

sales team
The users who possess access to an opportunity record. A user who creates the opportunity record is automatically part of the sales team. Other users can also be assigned to the sales team so that they can collaborate on the opportunity.

side pane
A user interface component that is available in native Microsoft Outlook that is analogous to a task pane or action pane in the Siebel Web Client. This region of the user interface is typically available on the right

side of the user interface. It displays a collection of data and actions that the user can choose and that are appropriate for the context that the user uses to access data.

Siebel Business Application

An application that is part of Siebel CRM, such as Siebel Call Center.

Siebel CRM data

Business data that is created in the CRM Desktop add-in, data that is created in the client of a Siebel Business Application, such as Siebel Call Center, or data that resides in the Siebel database on the Siebel Server. Examples include an opportunity, account, or activity.

Siebel CRM Desktop

A solution provided by Oracle that includes modifications to the standard Microsoft Outlook capabilities that allows the user to work with CRM records and business processes from the Siebel CRM Desktop user interface.

Siebel Server

The server that runs the Siebel Server software. The Siebel Server processes business logic and data access for Microsoft Outlook.

Siebel Web services framework

Provides access to an existing Siebel business service or workflow process as a Web service to be consumed by an external application.

SOAP

Simple Object Access Protocol, a protocol that allows a user or program to interact with Web services by exchanging XML messages that conform to SOAP.

Simple Object Access Protocol (SOAP)

See SOAP.

standard Microsoft Outlook

The native Microsoft Outlook application without the CRM Desktop add-in.

synchronization

A process that exchanges transactions between Oracle's Siebel CRM Desktop for Microsoft Outlook and Microsoft Outlook. This synchronization makes sure that CRM data is the same on the Siebel Server and in Outlook.

synchronization filter

Criteria that are considered during data synchronization so that some records are included and other records are excluded from processing during synchronization.

task (Microsoft Outlook)

A part of a set of actions that accomplish a job, solve a problem or completes an assignment. In the native Microsoft Outlook application, a task is a collection of simple business objects on the user level. A task can be used as a reminder and also as a tracking tool for an effort that is scheduled compared to an actual effort.

task (Siebel CRM)

A logical unit of work that is performed by a user to finish a business operation. From the perspective of the Microsoft Outlook user, a task is the view representation of a logical unit of work that the user must perform. The task is presented in Siebel CRM as a link that can be clicked in the task pane. The view, or series of views, where the user performs this unit of work is then displayed. It is part of the Task UI solution.

Web services

Self-contained, modular applications that can be described, published, located, and called over a network. Web services perform encapsulated business functions, ranging from a simple request-reply to full business process interactions. Web services use components, Internet standards, and protocols, such as HTTP, XML code, and SOAP.

See also SOAP.

Index