COMVERSE

Converged Billing Suite

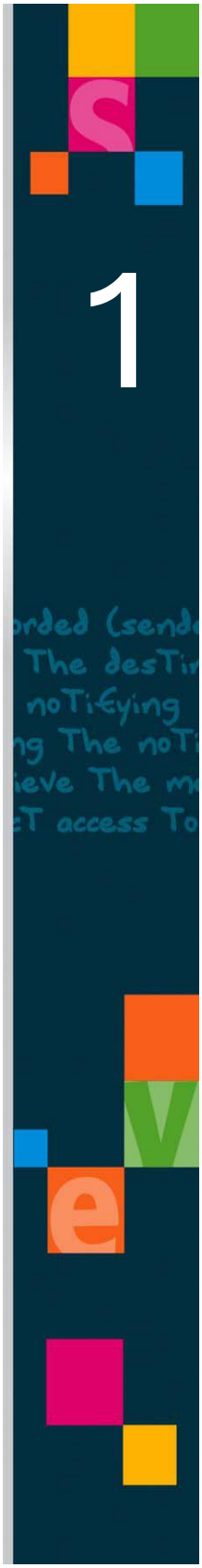
3

Client SDK

CBS-3.0-CONVERGED-SDKG-2008-09-15

Chapter 1

Overview



Overview

The client SDK interface is an object-oriented programming interface to the remote public interface layer. As such, this interface provides the following capabilities:

- Provides an abstraction layer over the actual implementation of the physical transport
- protocol used to communicate with the remote public interface layer
- Provides local/remote transparency to:
- Client consumers deployed as fat client applications
- Remote web applications
- Local web applications
- Applications that are directly integrated with the server
- Facilitates the use of common business logic
- Provides support for client-side caching, client-side enumeration, client-side defaulting, and basic client-side validation support
- Facilitates the transfer of out-of-band data (for example, auditing) to the server

Required Knowledge

To use the client SDK with the Comverse ONE solution, you must have an understanding of how the Comverse ONE solution works. Similarly, to use the client SDK interface to work with an optional module, you must understand how that optional module works. For in-depth information on the workings of the Comverse ONE solution and its optional modules, see the Comverse ONE solution documentation set.

Client Classes and Methods

Client classes are available only in the client SDK and provide services for Unified API objects. Services can be basic, such as creating or deleting an object, or more complex, such as creating an initialized class bundle.

Each method available in a `Client` class has a corresponding method in a `CommonBusiness` class, which performs light data validations on the inputs to the method in the Client class. The two methods have the same name.

For example, the `accountAddBundleInstance` method in the `AccountClient` class has a twin method in the `AccountCommonBusiness` class, which performs data validation on inputs to accountAddBundleInstance. When you call the accountAddBundleInstance method in the `AccountClient` class, the common business method is automatically executed.

To create and use a `Client` class, use code like the following (lines related to client objects and methods are shown in bold):

```
// Import the appropriate class libraries
import com.comverse.api.csm.base.client.AccountClient;
import com.comverse.api.csm.base.data.AccountIdentifier;
import com.comverse.api.csm.base.message.AccountGetOutputMessage;
import com.comverse.api.framework.client.UserContext;
import com.comverse.api.framework.errors.ApiException;
```

```
try
{
      // Create the input object
      AccountIdentifier accountIdentifier = new AccountIdentifier();
      // Set fields on the input object
      accountIdentifier.setAccountExternalId("E121554");
      //...
      // Create the client object
      AccountClient acctClient = new AccountClient();
      // Call the accountGet client method
      AccountGetOutputMessage acctCreateOut =
      acctClient.accountGet( accountIdentifier );
}
      catch (ApiException ex)
{
      ex.printStackTrace();
}
```

Common Business Logic

Common business logic consists primarily of quick validations on the data passed into any method, and is performed by common business classes An example is setting default values on some fields so that the fields pass the validations.

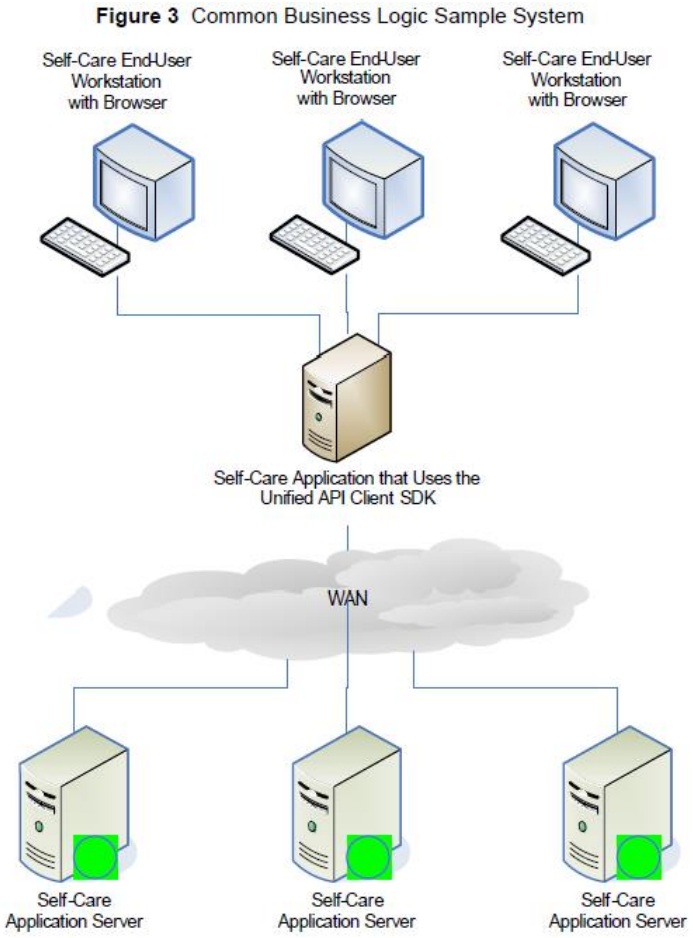
The client SDK provides a framework for executing common business logic in either the client SDK or on the server, depending on the value of the `client.business-logic-enabled` setting in the CCBSClientConfiguration.xml. This property can be set to true or false, which causes the following behavior:

- **true**: Causes common business logic to be executed in the client SDK before the call is sent to the server, thus improving overall latency. The common business logic is executed by the code in a method within a client class, for example in a method within the AccountClient class.
- **false**: Causes the logic to be executed on the server, with the possible risk of expending time and resources sending a call to the server only to discover that something simple is wrong with the input data.

The business logic in the common business classes (that is, the "shared" business logic) is always executed at some point. If the property is missing, then the client relies on the default setting for the property, which is false. If the underlying setting is false (either from an actual property setting or the default property setting), then the client passes an indicator value in the call to the server indicating that the shared business logic has not been applied. The server then executes this logic on the server before executing the purely server side business logic.

As an example, assume that you are implementing a self-care system and that end users access the system's web pages via browsers. The browsers communicate with a self-care client application (written using the Unified API client SDK) that is deployed on an application server. Assume also that there are several self-care application servers deployed throughout a large geographic region, but that there is a centralized Unified API server that these self-care clients talk to, and the Unified API server is deployed somewhere across a wide-area network (WAN). See Figure 3, "Common Business Logic Sample System."

Figure 3 Common Business Logic Sample System



You would most likely want client.business-logic-enabled to be set to true. That way, common business validations can occur as quickly as possible — you can find out if a common business validation fails without an expensive trip across the WAN. The capacity of the self-care application servers will be somewhat degraded because it will take slightly longer for each Unified API client call to complete on the self-care application server because the extra common business logic i1s executed on the self-care application server instead of the main Unified API application server. You would therefore need to plan the self-care application server capacity to account for the extra processing load.