


Diese Leseprobe haben Sie beim
 edv-buchversand.de heruntergeladen.
Das Buch können Sie online in unserem
Shop bestellen.
[Hier zum Shop](#)

Kapitel 3

Integration über Remote Function Call und SAP .NET Connector

Der direkte Zugriff auf ein SAP-System mit C# oder Java lässt sich mit den SAP-Konnektoren realisieren. In diesem Kapitel stellen wir diese Methoden vor und erläutern die Vorgehensweise an einfachen Beispielen.

Bevor wir Ihnen in diesem Kapitel die Integration von Microsoft- und SAP-Lösungen über die SAP-Konnektoren zeigen, klären wir zunächst einige wichtige Begriffe in diesem Umfeld. Sie sollten diese Begriffe kennen, um die folgenden Ausführungen nachvollziehen zu können.

Sämtliche real im betriebswirtschaftlichen Umfeld existierenden Objekte sind im SAP-System als *Business-Objekte* abgebildet. Diese Objekte kapseln die Geschäftsprozesse und zugehörigen Daten. Wenn eine Anwendung ein Business-Objekt verwenden möchte, werden nur die Informationen zu den Objektmethoden benötigt. So kann ein Entwickler mit den Business-Objekten arbeiten und die entsprechenden Methoden nutzen, ohne die Details der Implementierung zu kennen oder gar beachten zu müssen.

Business-Objekte

Der Zugriff auf ein Business-Objekt wird mit *Business Application Programming Interfaces* (BAPIs) realisiert. Bei BAPIs handelt es sich um SAP-Standard-Schnittstellen. Alle BAPIs im SAP-System sind als Funktionsbausteine realisiert, die sich im *Function Builder* des SAP-Systems befinden. Jeder Funktionsbaustein, der einem BAPI zugrunde liegt, unterstützt die Technologie des *Remote Function Calls* (RFC).

BAPI

Als RFC bezeichnet man den Aufruf eines Funktionsbausteins innerhalb eines SAP-Systems durch ein anderes, externes System bzw. eine Applikation. Es handelt sich bei RFC also um eine Schnittstelle, mit der Funktionsbausteine aufgerufen und ausgeführt sowie Daten übermittelt und abgefragt werden können.

Remote Function Call

Für die Entwicklung von Applikationen, die RFCs nutzen, hat SAP schon vor vielen Jahren ein Software Development Kit herausgebracht: das RFC-SDK. Der zentrale Bestandteil des RFC-SDK war die dynamische RFC-Bibliothek (Dynamic Link Library, DLL) **librfc32.dll**. Dabei handelt es sich um eine

librfc32.dll

Bibliothek, die Funktionen für die RFC-Kommunikation bereitstellt. Später wurden weitere Komponenten hinzugefügt, die die Nutzung der `librfc32.dll` vereinfachten und erweiterten. Den Fokus bildeten zunächst COM-fähige Programmiersprachen, für die verschiedene Objekte zur Verfügung gestellt wurden, die die `librfc32.dll` kapseln. Bei dem *Component Object Model* (COM) handelt es sich um eine Technologie, die von Microsoft für die Kommunikation zwischen verschiedenen Prozessen entwickelt wurde.

SAP .NET Connector
und
SAP Java Connector

In diesem Kapitel stellen wir Ihnen die Konnektoren *SAP .NET Connector* und *SAP Java Connector* vor. Sie dienen im Prinzip auch nur zur Kapselung der `librf32.dll`. Auf Basis dieser Werkzeuge lassen sich Proxy-Klassen erstellen, über die eine Kommunikation mit dem SAP-System möglich ist. Um den Zugriff auf SAP-Systeme ausgehend von einer in Java oder mit Visual Studio .NET für die Microsoft-Plattform entwickelten Applikation zu realisieren, stellen die beiden Konnektoren die modernste Lösung dar, die SAP für diese Integration bereitstellt. Wir gehen im Folgenden insbesondere auf den SAP .NET Connector ein und zeigen die Verwendung dieses Konnektors anhand eines Beispiels. Mit diesem Konnektor lassen sich Applikationen in Visual Studio entwickeln, die über RFC auf SAP-Systeme zugreifen können. In Kapitel 6, »Verwendung von Drittanbieter-Add-ons«, stellen wir Ihnen mit Theobald ERPConnect eine weitere Software vor, die noch umfangreichere Möglichkeiten als der SAP .NET Connector bietet.

3.1 Remote Function Call im Überblick

Um einen kundeneigenen Funktionsbaustein remotefähig zu machen, um also den Zugriff auf diesen Funktionsbaustein per RFC ausgehend von einem externen System zu ermöglichen, muss dieser Funktionsbaustein in der ABAP Workbench als remotefähig gekennzeichnet werden. Wie das geht, beschreiben wir am Beispiel von Webservices in Abschnitt 4.2.2, »Webservice einrichten«. Für das Beispiel in diesem Kapitel verwenden wir BAPIs und die dazugehörigen, bereits im System vorhandenen Funktionsbausteine.

RFC-Typen SAP unterscheidet verschiedenen Typen von RFCs für den Zugriff auf die SAP-Systeme. Es gibt die folgenden Typen (siehe dazu auch <http://s-prs.de/v619018>):

- **Synchronous RFC (sRFC) für den lesenden Zugriff**
Hierbei handelt es sich um die erste Version des RFC. Die Daten werden direkt abgerufen oder geschrieben. Dabei werden alle Lese- oder Schreib-

operationen nacheinander ausgeführt, unabhängig davon, ob ein Fehler aufgetreten ist. Das ist für das Schreiben der Daten problematisch, denn der Aufruf kann im Fehlerfall nicht wiederholt werden, da es sonst unter Umständen zu doppelten Datensätzen kommen könnte. Aus diesem Grund wird sRFC nicht mehr verwendet.

- **Transactional RFC (tRFC) für den transaktionalen Zugriff**
Bei dieser Art des RFC erfolgt die Ausführung der Lese- und/oder Schreiboperationen im SAP-System garantiert nur einmal (transaktional).
- **Queued RFC (qRFC) zur Lastverteilung**
Die Daten werden angefragt, und die Ausführung des aufrufenden Programms wird direkt fortgesetzt. Die Anfrage selbst wird in einer Warteschlange (Queue) gespeichert und abgearbeitet. Damit lassen sich Anfragen parallelisieren und die Systeme besser auslasten. Das Resultat ist eine bessere Performance.
- **Background RFC (bgRFC) zur Hintergrundverarbeitung**
Dieser RFC-Typ wird zum transaktionalen Schreiben verwendet. Es handelt sich um die Nachfolgetechnologie aus den vorher genannten Typen tRFC und qRFC. Der Aufruf erfolgt im SAP-System wie beim tRFC garantiert nur einmal. Außerdem werden die Anfragen über eine Warteschlange (Queue) abgearbeitet, damit sichergestellt ist, dass die Anfragen auch in der richtigen Reihenfolge abgearbeitet werden.

Sowohl mit dem SAP .NET Connector als auch mit dem SAP Java Connector können Sie alle diese RFC-Typen einsetzen. In unseren Beispielen werden wir immer bgRFC verwenden. Dazu ist keine besondere Einstellung notwendig.

3.2 Remote Function Call mit C# über den SAP .NET Connector

In diesem Abschnitt zeigen wir Ihnen, wie Sie einen remotefähigen Funktionsbaustein aus einem SAP-System in einer in C# programmierten Microsoft-Windows-Anwendung einsetzen können. Dazu verwenden wir den SAP .NET Connector. Wir erklären Ihnen zunächst, wie Sie den SAP .NET Connector installieren und verwenden und welche weiteren Vorbereitungen notwendig sind. Im Anschluss zeigen wir Ihnen, wie Sie den Funktionsbaustein aus dem SAP-System mithilfe des Konnektors aufrufen können.

3.2.1 Installation des SAP .NET Connectors

Sie können den SAP .NET Connector unter der Adresse <https://service.sap.com/connectors> aus dem SAP Service Marketplace herunterladen. Auf dieser Seite können Sie außerdem die **librfc32.dll** sowie den SAP Java Connector herunterladen, den wir in Abschnitt 3.3, »Remote Function Call mit Java über den SAP Java Connector«, verwenden werden. Für den Download benötigen Sie einen S-Benutzer für den SAP Service Marketplace (siehe Abschnitt 1.4.2, »Integration in SAP Fiori«).

Technische Voraussetzungen

Der SAP .NET Connector steht in der aktuellen Version 3.0 (NCo 3.0) sowohl für das Microsoft .NET Framework 2.0 (und die Versionen 3.0 und 3.5) als auch für das Microsoft .NET Framework 4.0 (und höher) jeweils in einer 32-Bit- und einer 64-Bit-Version zur Verfügung. In SAP-Hinweis 85863 finden Sie Informationen zur Freigabe- und Support-Strategie für den SAP .NET Connector. Für die Version des 2.0 .NET Frameworks benötigen Sie die Microsoft C++-Runtime-DLLs in Version 8.0, für .NET 4.0 Version 10. Der .NET Connector wurde in C++ geschrieben und verwendet Funktionen, die in dieser Bibliothek enthalten sind. Diese können Sie auf den folgenden Microsoft-Seiten als Installationspaket herunterladen:

- Für 64 Bit:
<http://s-prs.de/v619019>
- Für 32 Bit:
<http://s-prs.de/v619020>

Der SAP .NET Connector 3.0 ist mit allen SAP-Backend-Systemen ab Release 4.0B kompatibel und unterstützt Unicode- und Nicht-Unicode-Systeme. Eine Abhängigkeit zur **librfc32.dll**, die in früheren Versionen zusätzlich zum Konnektor eingebunden werden musste, besteht nicht mehr.

Installation

Haben Sie die passende C++-Laufzeitumgebung installiert und die entsprechende Version des SAP .NET Connectors heruntergeladen, können Sie diesen über die Installationsdatei (mit der Dateiondung .msi) installieren. Führen Sie diese Datei aus, um den Installationsassistenten zu starten. Wählen Sie auf der zweiten Seite der geführten Installationsprozedur die Option **None**, um alle zum SAP .NET Connector gehörigen DLLs in das angegebene Verzeichnis zu entpacken. Alternativ können Sie diese DLLs auch in den *Global Assembly Cache* (GAC) Ihres Rechners installieren (siehe Abbildung 3.1), wenn die zu entwickelte Applikation später auf diesem Rechner laufen soll. Idealerweise binden Sie die DLLs aber in Ihr Projekt ein und geben sie auch über dieses Projekt weiter.

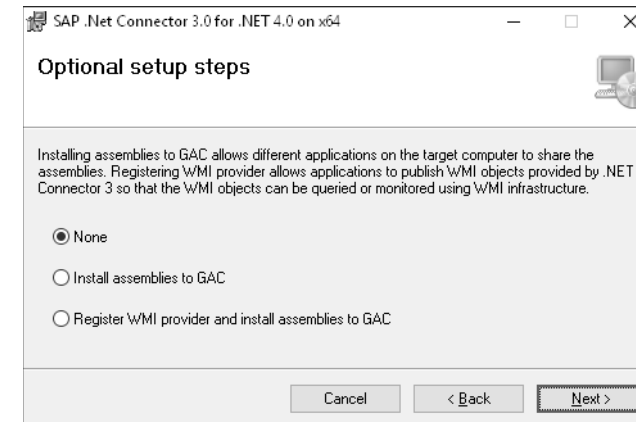


Abbildung 3.1 Zielverzeichnis für die DLLs auswählen

Im angegebenen Verzeichnis befinden sich anschließend die folgenden DLL-Dateien:

- **libicudcnumber.dll**
- **rsc4n.dll**
- **sapnco.dll**
- **sapnco_utils.dll**

Damit ist die Installation des SAP .NET Connectors bereits abgeschlossen.

3.2.2 SAP .NET Connector in ein Visual-Studio-Projekt einbinden

Um in unserem Beispiel einen Funktionsbaustein aus einer C#-Applikation aufzurufen, möchten wir eine Funktion (d. h. eine Methode) eines BAPI verwenden. Dazu müssen wir zunächst ein geeignetes BAPI auswählen. Im *BAPI Explorer* des SAP-Systems können Sie alle im System vorhandenen BAPIs ansehen und sich die zugehörigen Methoden anzeigen lassen. Den BAPI Explorer öffnen Sie mit Transaktion BAPI (siehe Abbildung 3.2).

BAPI im Explorer anzeigen

Für das folgende Beispiel verwenden wir Funktionen aus dem Business-Objekt *Customer*. Wechseln Sie im BAPI Explorer zur alphabetischen Ansicht, und navigieren Sie zum Knoten **Customer**. Hier sehen Sie die verfügbaren BAPIs für dieses Business-Objekt sowie alle Methoden, die in diesen BAPIs zur Verfügung stehen. Wir werden die Methode *GetList* verwenden, die in dem Funktionsbaustein **BAPI_CUSTOMER_GETLIST** implementiert ist. Damit können Sie Kundeninformationen aus dem SAP-System aufrufen. Den Namen dieses Funktionsbausteins finden Sie auf der Registerkarte auf der

rechten Seite **Details**, nachdem Sie die Methode in der Baumstruktur ausgewählt haben.

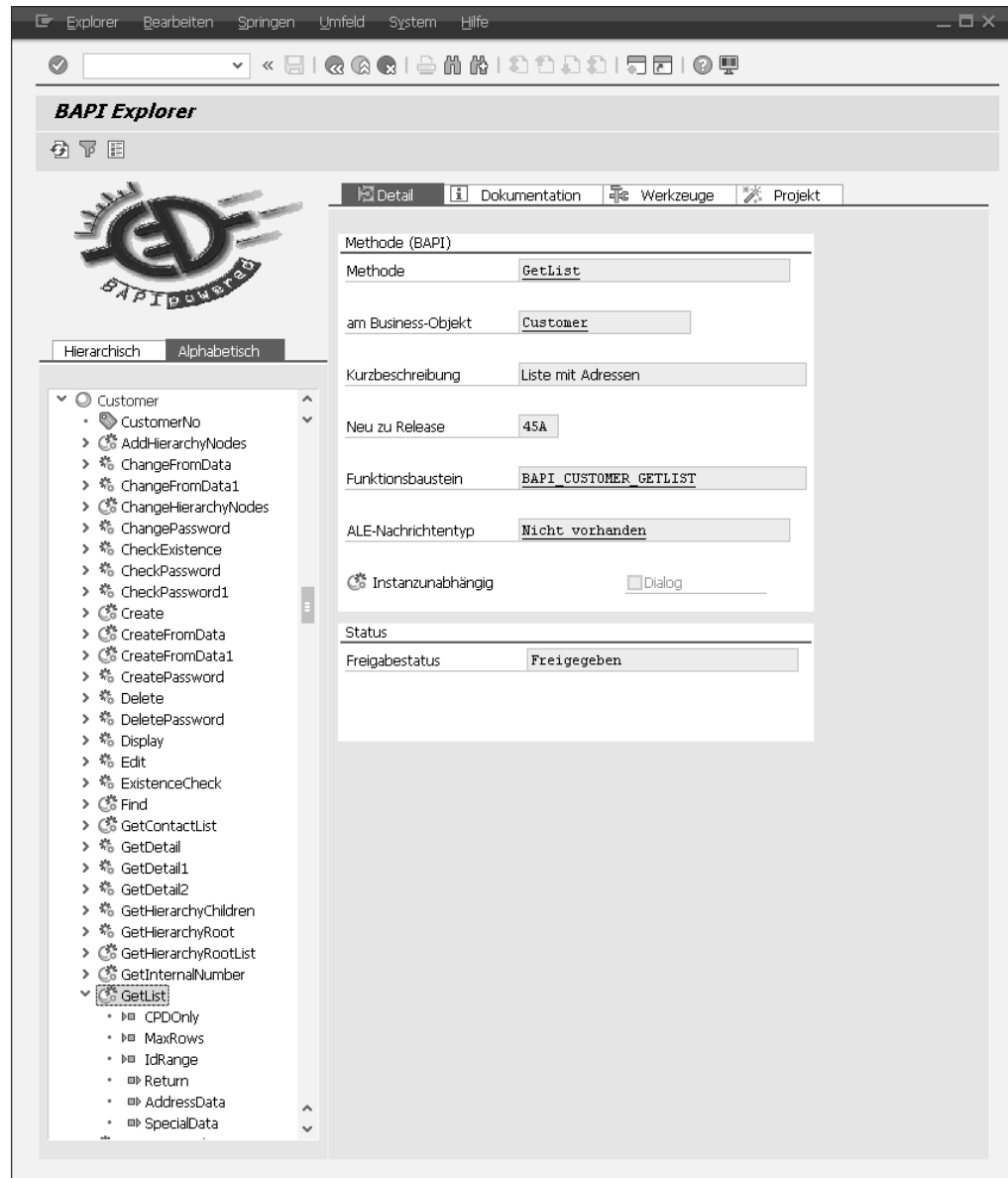


Abbildung 3.2 BAPI Explorer

Visual-Studio-Projekt anlegen Als Nächstes erstellen wir eine Konsolen-Applikation mit C# in der Microsoft-Entwicklungsumgebung Visual Studio. Von dieser Applikation aus

wollen wir den Funktionsbaustein später über einen RFC aufrufen. Für die Entwicklung dieses Szenarios müssen Sie Visual Studio installiert haben. Die kostenlose *Visual Studio Community Edition* können Sie unter <https://www.visualstudio.com/de/downloads> herunterladen und installieren.

Öffnen Sie dann Visual Studio, und legen Sie zunächst ein neues Projekt an. Fügen Sie diesem Projekt die im vorangehenden Abschnitt installierten DLL-Bibliotheken des SAP .NET Connectors als Verweis hinzu. Klicken Sie dafür mit der rechten Maustaste auf das Projekt, und wählen Sie im Kontextmenü den Eintrag **Hinzufügen • Verweis** aus. Sie können auch direkt mit der rechten Maustaste auf den Knoten **Verweise** in der Projektmappe klicken und dann den Eintrag **Verweis hinzufügen ...** im Kontextmenü auswählen (siehe Abbildung 3.3).

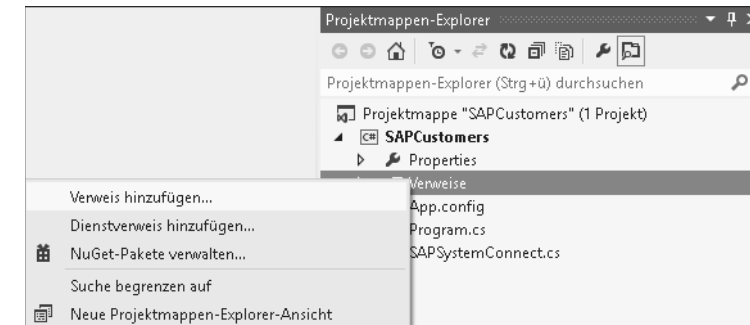


Abbildung 3.3 Verweis hinzufügen

Im sich daraufhin öffnenden Verweis-Manager klicken Sie auf **Durchsuchen** und wählen dann die DLLs **sapnco.dll** und **sapnco_utils.dll** aus (siehe Abbildung 3.4).

SAP-.NET-Connector-Verweis hinzufügen

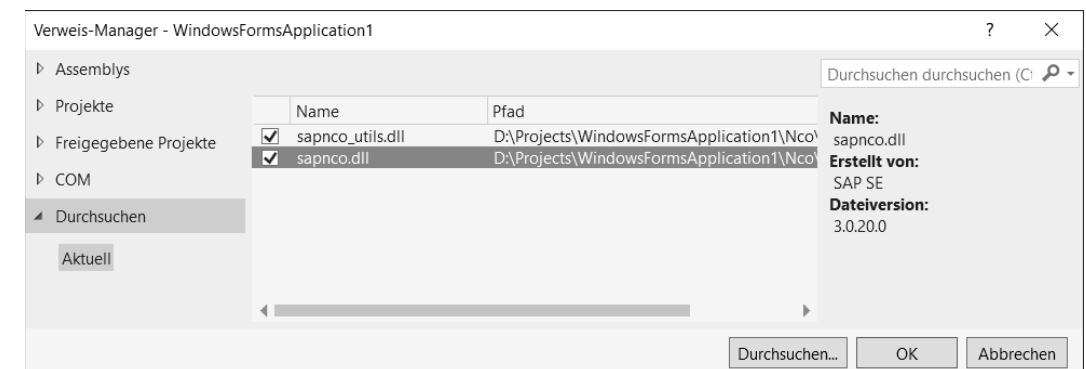


Abbildung 3.4 Verweis-Manager in Visual Studio

Klasse für die SAP-Systemverbindung

Fügen Sie dem Projekt nun eine neue Klasse hinzu. Klicken Sie dazu mit der rechten Maustaste in der Projektmappe auf das Hauptprogramm SapCustomers, und wählen Sie im Kontextmenü **Hinzufügen • Klasse**. Diese Klasse soll die Verbindung zum SAP-System herstellen und verwalten. Nennen Sie die Klasse »SAPSystemConnect«.

Wechseln Sie in den Quellcode dieser Klasse, und fügen Sie ihr über die Anweisung `using` einen Verweis auf den Konnektor `SAP.Middleware.Connector` hinzu (siehe Abbildung 3.5).

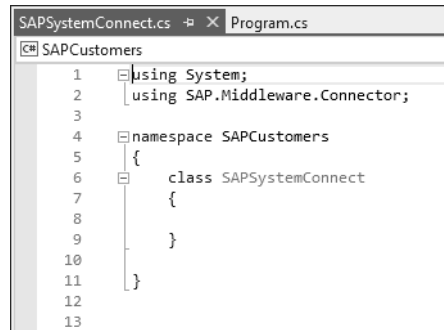


Abbildung 3.5 SAP .NET Connector in die C#-Klasse einbinden

Verbindung zum SAP-System implementieren

Um eine Verbindung zu einem SAP-System herzustellen, müssen Sie nun das Interface `IDestinationConnection` implementieren. Am einfachsten ist es, im Quellcode direkt hinter der Klassenbezeichnung den Namen des Interfaces, getrennt durch einen Doppelpunkt anzufügen:

```
class SAPSystemConnect : IDestinationConfiguration
```

Klicken Sie anschließend mit der rechten Maustaste auf den Namen des Interfaces `IDestinationConfiguration`, und wählen Sie im Kontextmenü den Eintrag **Schnittstelle implementieren** (siehe Abbildung 3.6). Visual Studio legt dann automatisch den notwendigen Programmcode für die Implementierung dieser Schnittstelle an.

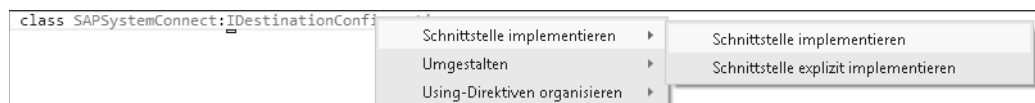


Abbildung 3.6 Schnittstelle durch Visual Studio implementieren lassen

Funktion mit Verbindungsparametern

Im nächsten Schritt wird die Funktion `GetParameters` mit den Verbindungsparametern (Serveradresse, Systemnummer, Client, Benutzername und Kennwort, Sprache, Timeout der Verbindung und Anzahl gleichzeitiger Verbindungen) angelegt. Deren Quellcode sehen Sie in Listing 3.1.

```
public RfcConfigParameters GetParameters(string
    destinationName)
{
    RfcConfigParameters parameters =
        new RfcConfigParameters();
    if ("Dev".Equals(destinationName))
    {
        parameters.Add(RfcConfigParameters.AppServerHost,
            "Server");
        parameters.Add(RfcConfigParameters.SystemNumber,
            "10");
        parameters.Add(RfcConfigParameters.Client,
            "010");
        parameters.Add(RfcConfigParameters.User,
            "Benutzer");
        parameters.Add(RfcConfigParameters.Password,
            "Kennwort");
        parameters.Add(RfcConfigParameters.Language,
            "DE");
        parameters.Add(RfcConfigParameters.PoolSize,
            "5");
        parameters.Add(RfcConfigParameters.ConnectionIdleTimeout,
            "600");
    }
    return parameters;
}
```

Listing 3.1 Verbindungsparameter zum SAP-System festlegen

In diesem Beispiel werden die Verbindungsparameter auf die angegebenen Werte gesetzt, wenn für den Übergabeparameter `destination` der Wert »Dev« gesetzt ist. Die Idee hierbei ist, dass verschiedene SAP-Systeme im Quellcode vorkonfiguriert werden. Mit dem Parameter lässt sich dann beim Aufruf der Verbindungsfunktion bestimmen, zu welchem System die Verbindung hergestellt werden soll. Sie werden im folgenden Abschnitt im Hauptprogramm noch sehen, wie die Verbindung erstellt wird (siehe Listing 3.6).

3.2.3 Aufruf des BAPI implementieren

Anschließend erstellen Sie eine Klasse `Customers`. Diese Klasse enthält die Funktion `GetCustomerDetails` mit den Parametern für das Zielsystem (`destination`, d. h. die RFC-Verbindung) und die Kundennummer

Klasse für Zugriff auf die Kundendaten

(customerID), die zum Aufruf des BAPI für das Business-Objekt Customer benötigt werden. Die vollständige Implementierung dieser Klasse sehen Sie in Listing 3.2. Im Anschluss erklären wir die einzelnen Schritte.

```
class Customers
{
    public void GetCustomerDetails(RfcDestination destination, string
customerID)
    {
        RfcRepository repository = destination.Repository;
        IRfcFunction customerList =
            repository.CreateFunction("BAPI_CUSTOMER_GETLIST");
        customerList.Invoke(destination);
        IRfcTable idRange = customerList.GetTable("IdRange");
        idRange.SetValue("SIGN", "I");
        idRange.SetValue("OPTION", "EQ");
        idRange.SetValue("LOW", customerID);
        customerList.SetValue("idrange", idRange);
        IRfcTable addressData = customerList.GetTable("AddressData");
        customerList.Invoke(destination);
        for (int cuIndex = 0; cuIndex < addressData.RowCount; cuIndex++)
        {
            addressData.CurrentIndex = cuIndex;
            Console.WriteLine(addressData.GetString("Customer"));
            Console.WriteLine(addressData.GetString("Name"));
            Console.WriteLine(addressData.GetString("Street"));
            Console.WriteLine(addressData.GetString("Country") + "-"
                + addressData.GetString("Postl_Cod1") + " "
                + addressData.GetString("City"));
        }
    }
}
```

Listing 3.2 Vollständige Klasse SAPSystemConnect.cs

Interface für die BAPI-Funktion

Zunächst erstellen Sie ein Interface customerList für die Funktion BAPI_CUSTOMER_GETLIST und verknüpfen dieses mit dem Zielsystem (siehe Listing 3.3).

```
RfcRepository repository = destination.Repository;
IRfcFunction customerList =
    repository.CreateFunction("BAPI_CUSTOMER_GETLIST");
customerList.Invoke(destination);
```

Listing 3.3 Programmteil: BAPI-Funktion anbinden

Nun definieren Sie einen Bereich für die Abfrage der Kundendaten. Dabei definieren Sie die übergebene Kundennummer (customerID) als Untergrenze sowie EQ als Eingrenzung, um nur die Daten dieses einen Kunden abzufragen (siehe Listing 3.4).

Einschränkung
auf eine Kunden-
nummer

```
IRfcTable idRange = customerList.GetTable("IdRange");
idRange.SetValue("SIGN", "I"); // I = inklusiv
idRange.SetValue("OPTION", "EQ"); // EG = Equal, BT = between
idRange.SetValue("LOW", customerID);
customerList.SetValue("idrange", idRange);
```

Listing 3.4 Programmteil: Kundennummernbereich festlegen

Im nächsten Schritt wird die Ergebnistabelle abgerufen. Zudem werden alle Zeilen dieser Tabelle in der Kommandozeile ausgegeben (siehe Listing 3.5). Typischerweise gibt es pro Kundennummer nur einen Datensatz und damit auch nur eine Zeile.

Ergebnis abrufen

```
IRfcTable addressData = customerList.GetTable("AddressData");
customerList.Invoke(destination);
for (int cuIndex = 0; cuIndex < addressData.RowCount;
    cuIndex++)
{
    addressData.CurrentIndex = cuIndex;
    Console.WriteLine(addressData.GetString("Customer"));
    Console.WriteLine(addressData.GetString("Name"));
    Console.WriteLine(addressData.GetString("Street"));
    Console.WriteLine(addressData.GetString("Country") + "-"
        + addressData.GetString("Postl_Cod1") + " "
        + addressData.GetString("City"));
}
```

Listing 3.5 Programmteil: Ergebniszeile ausgeben

Damit haben Sie die Herstellung einer RFC-Verbindung zum Abruf der Adressdaten eines Kunden aus einem SAP-System in einem C#-Programm implementiert.

Im Hauptprogramm rufen Sie die implementierten Funktionen nun auf und parametrisieren diese mit entsprechend Kommandozeilen-Parametern für das verwendete SAP-System (in unserem Beispiel »Dev«) sowie eine Kundennummer (siehe Listing 3.6).

Aufruf der
Funktionen

```

static void Main(string[] args)
{
    SAPSystemConnect sapCfg = new SAPSystemConnect();
    RfcDestinationManager.RegisterDestinationConfiguration(sapCfg);
    RfcDestination rfcDest = null;
    rfcDest = RfcDestinationManager.GetDestination(args[0]);

    Customers customer = new Customers();
    customer.GetCustomerDetails(rfcDest, args[1]);
    System.Environment.Exit(0);
}

```

Listing 3.6 Hauptprogramm: Programmfunktionen ausführen und Parameter übergeben

Test über die Kommandozeile

Das Programm können Sie nach dem Kompilieren über die Kommandozeile testen. In Abbildung 3.7 sehen Sie das Ergebnis. Wichtig ist, dass Sie beim Aufruf der durch die Kompilierung erzeugten .exe-Datei die Parameter für das Zielsystem (Dev) und eine Kundennummer übergeben.

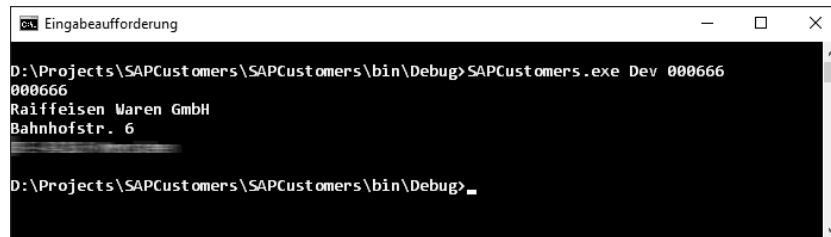


Abbildung 3.7 Das Programm mit Abruf einer Kundenadresse testen

Dieses einfache Beispiel sollte veranschaulichen, wie ein Zugriff mittels C# auf ein SAP-System über einen RFC erfolgen kann. Das Beispiel kann allerdings nur als Proof of Concept dienen und sollte so nicht im produktiven Umfeld eingesetzt werden, da es beispielsweise keinerlei Fehlerabfrage beinhaltet.

3.3 Remote Function Call mit Java über den SAP Java Connector

Für die Implementierung von RFCs mit der Programmiersprache Java können Sie vergleichbar zum SAP .NET Connector für .NET-Entwicklungen den *SAP Java Connector* (SAP JCo) verwenden. Damit Sie beide Konnektoren vergleichen können, setzen wir in diesem Abschnitt das gleiche Beispiel

um, das wir im vorangegangenen Abschnitt für den SAP .NET Connector erstellt haben.

3.3.1 Installation des SAP Java Connectors

Sie können den SAP Java Connector ebenso wie den SAP .NET Connector unter der Adresse <https://service.sap.com/connectors> im SAP Service Marketplace herunterladen. Die zum Zeitpunkt der Drucklegung dieses Buches aktuelle Version des SAP Java Connectors lautet 3.0.17. Sie benötigen Java Software Development Kit (JDK) bzw. Java Runtime Environment (JRE) ab Version 5.0. Der SAP Java Connector steht neben Microsoft Windows auch für andere Plattformen zur Verfügung. So gibt es beispielsweise Versionen für Linux, Apples macOS, Oracle Solaris, Hewlett Packard Unix (HP-UX) oder IBM AIX.

SAP-seitig unterstützt der SAP Java Connector Systeme ab Release 3.1H. Er unterstützt alle SAP-Komponenten, die über BAPIs oder RFCs angesprochen werden können. Mit dem SAP Java Connector können Sie ebenfalls alle in Abschnitt 3.1, »Remote Function Call im Überblick«, vorgestellten RFC-Typen verwenden.

Nach dem Herunterladen des SAP Java Connectors aus dem SAP Service Marketplace entpacken Sie die .zip-Datei in ein beliebiges Verzeichnis, z. B. `D:\Projekte\SAPJCo`. In Abbildung 3.8 sehen Sie die enthaltenen Dateien und Verzeichnisse.

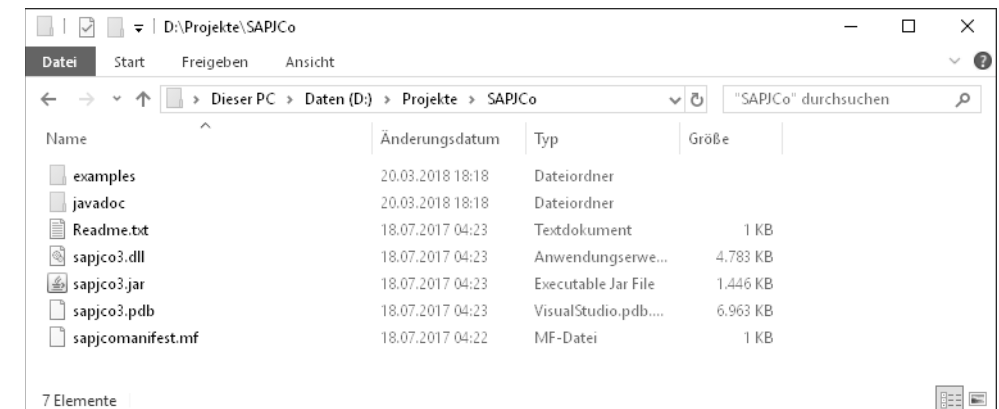


Abbildung 3.8 Den SAP Java Connector installieren

Fügen Sie den gewählten Verzeichnis-Pfad anschließend zu Ihrer Path-Umgebungsvariable hinzu (siehe Abbildung 3.9). Die Umgebungsvariablen Ihres Windows-Systems finden Sie unter den Systemeigenschaften. Wählen Sie hier den Button **Umgebungsvariablen**.

Unterstützte
Java-Versionen

Unterstützte
SAP-Releases

Installation

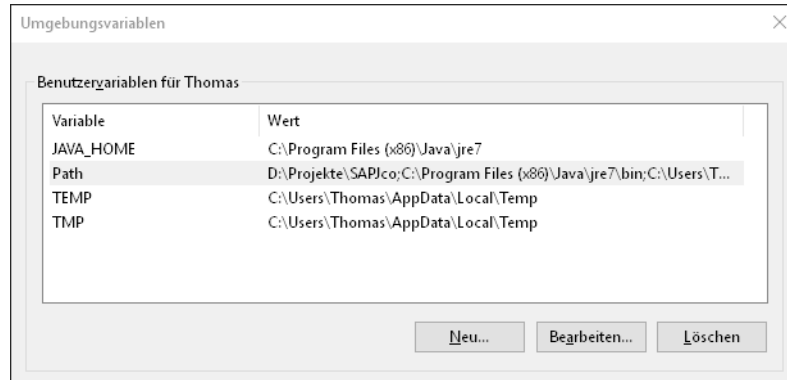


Abbildung 3.9 Umgebungsvariable setzen

Für die produktive Verwendung des SAP Java Connectors benötigen Sie die folgenden Dateien aus der .zip-Datei:

- sapjco3.jar
- sapjco3.dll

Beide Dateien sollten im gleichen Verzeichnis abgelegt sein.

3.3.2 SAP Java Connector in ein Eclipse-Projekt einbinden

Unsere Beispielanwendung zur Demonstration des Datenzugriffs über den SAP Java Connector entwickeln wir in Eclipse für Windows. Das in diesem Abschnitt beschriebene Vorgehen können Sie aber natürlich auf jede andere Entwicklungsumgebung übertragen.

Projekt anlegen

Starten Sie Eclipse, und legen Sie ein neues Java-Projekt an. Wählen Sie dazu den Menüpfad **File • New • Java Project**. Wählen Sie einen beliebigen Projektnamen, z. B. »JCo Demo«. Die anderen Einstellungen können Sie unverändert lassen (siehe Abbildung 3.10). Im Bereich **JRE** können Sie wählen, welche Java-Laufzeitumgebung Sie für das Projekt verwenden möchten. Wie im vorangehenden Abschnitt 3.3.1, »Installation des SAP Java Connectors«, aufgeführt, werden die JRE-Versionen ab 5.0 vom SAP Java Connector unterstützt. Nachdem Sie Ihre Einstellungen vorgenommen haben, klicken Sie auf **Finish**.

SAP Java Connector in das Projekt einbinden

Klicken Sie nun in der Navigationsleiste mit der rechten Maustaste auf den eben angelegten Projektordner, und wählen Sie **Build Path • Configure Build Path** im Kontextmenü. Im sich daraufhin öffnenden Fenster können Sie nun auf der Registerkarte **Libraries** den SAP Java Connector Ihrem Projekt hinzufügen. Klicken Sie dazu auf den Button **Add External JARs**. Im sich öffnenden Dialog wählen Sie die Datei **sapjco3.jar** aus (siehe Abbildung 3.11). Schließen Sie dann den Dialog mit dem Button **Apply and Close**.

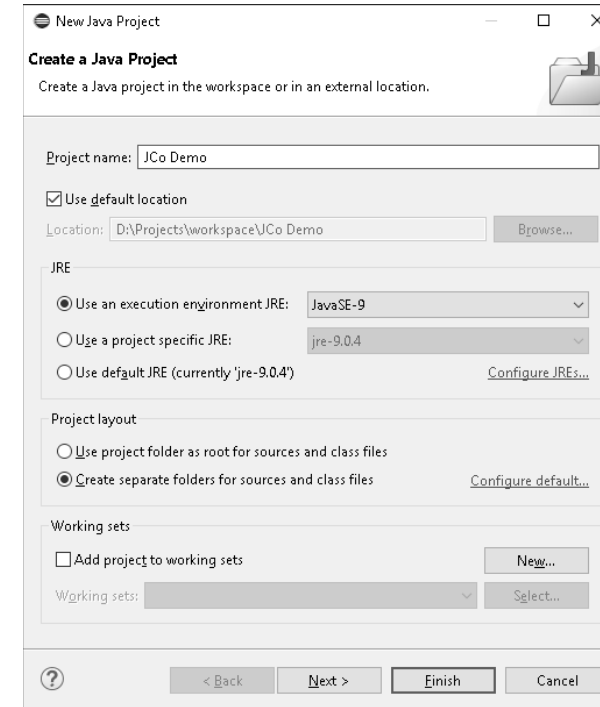


Abbildung 3.10 Neues Java-Projekt anlegen

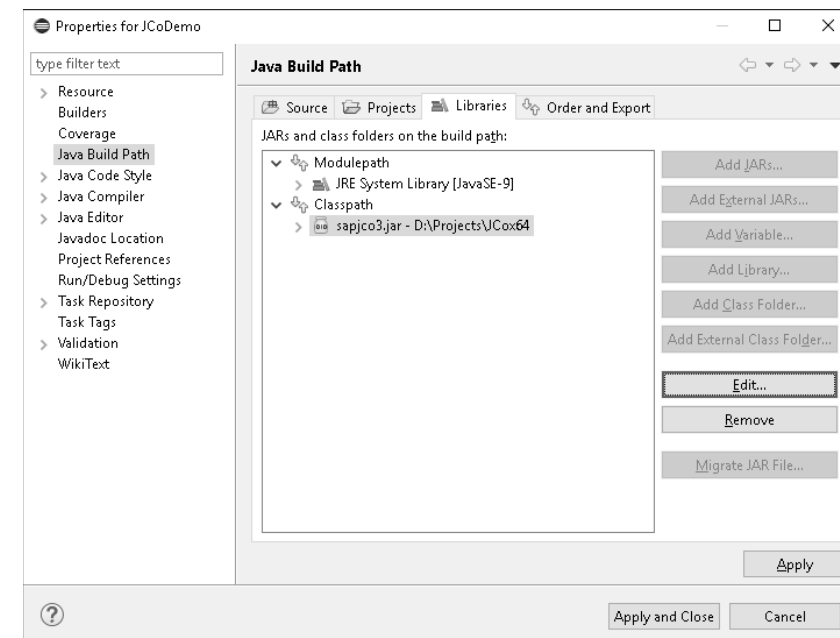


Abbildung 3.11 SAP Java Connector zum Projekt hinzufügen

Klasse für das Programm hinzufügen

Fügen Sie nun eine neue Klasse zu Ihrem Projekt hinzu. Diese soll später das eigentliche Programm enthalten. Klicken Sie dazu wieder mit der rechten Maustaste auf den Projektordner, und wählen Sie **New • Class** im Kontextmenü. Vergeben Sie unter **Package** einen Namen für das Paket dieses Projekts (hier: »de.sappress.demo«). Im Feld **Name** geben Sie einen Namen für die Klasse ein, z. B. »Program« (siehe Abbildung 3.12). Die vorgegebenen Einstellungen für die **Superclass** und die **method stubs** lassen Sie stehen. Diese Methoden werden dem Projekt automatisch hinzugefügt, sodass Sie sie nicht später manuell hinzuzufügen müssen. Klicken Sie abschließend auf **Finish**.

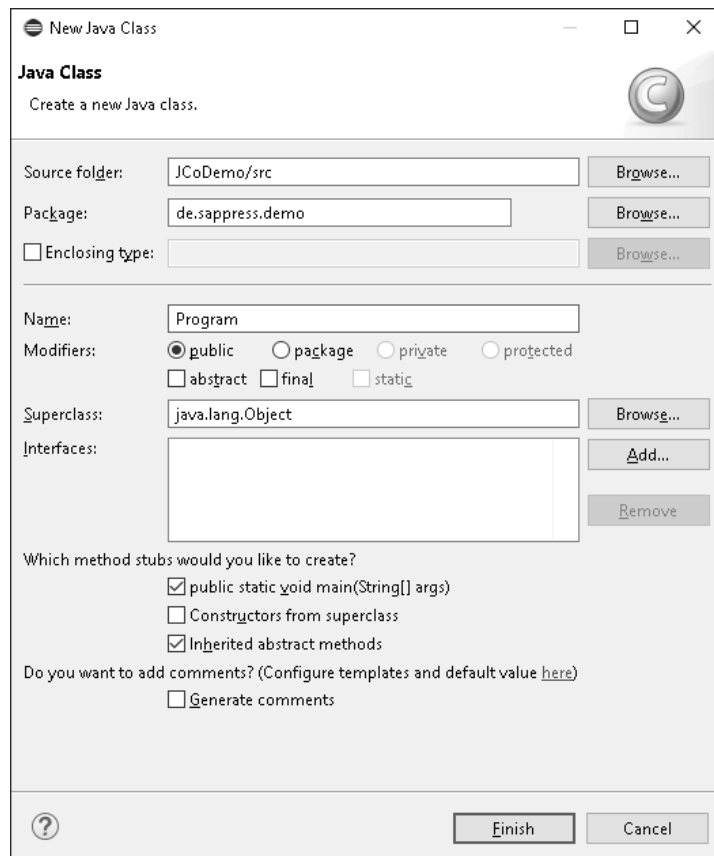


Abbildung 3.12 Neue Klasse anlegen

Das Projekt ist nun soweit vorbereitet, um mit der Implementierung der Anwendung zu starten. Alle Elemente des Projekts werden Ihnen im Package Explorer angezeigt (siehe Abbildung 3.13). Im ersten Schritt implemen-

tieren wir nun die Verbindung zu einem SAP-System, um dann im zweiten Schritt einen RFC abzusetzen und das Ergebnis in der Konsole auszugeben.

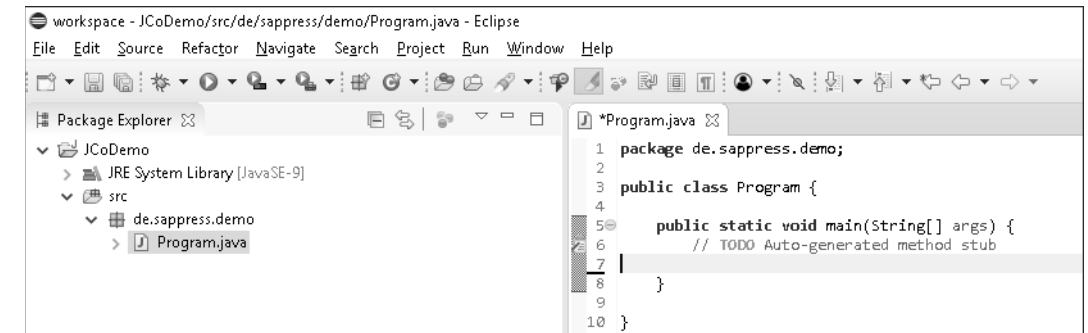


Abbildung 3.13 Vorbereitetes Projekt im Package Explorer

3.3.3 Verbindung zum SAP-System implementieren

Die Verbindung zu einem SAP-System wird mit der Klasse `JCoDestinationManager` hergestellt. Es gibt unterschiedliche Wege, die Verbindungsinformationen in dieser Klasse zu hinterlegen. Die einfachste Variante ist es, eine Datei mit der Endung `.jcoDestination` in das Basisverzeichnis des Projekts zu legen. Diese Datei kann mit der Methode `getDestination` der Klasse `JCoDestinationManager` geladen werden. Im Produktivumfeld ist es jedoch nicht sinnvoll, die Verbindungsdaten (insbesondere Benutzernamen und Kennwörter) statisch im Code abzulegen. Implementieren Sie deshalb einfach das Interface `DestinationDataProvider` in Ihrem Projekt, um die Verbindungsinformationen zu implementieren.

Verbindungsklasse

Weiterführende Informationen zur Client-Programmierung für den SAP Java Connector

Sollten Sie weiterführende Informationen zur Client-Programmierung benötigen, finden Sie diese in der SAP-Dokumentation unter:

<http://s-prs.de/v619021>



In unserem Beispiel werden wir die Variante mit der `.jcoDestination`-Datei implementieren, um das Beispiel nicht zu verkomplizieren. Legen Sie dazu eine Datei zum Projekt an, indem Sie mit der rechten Maustaste auf den Projektordner klicken und **New • File** wählen. Geben Sie der Datei z. B. den Namen »SAPSYSTEM.jcoDestination« (siehe Abbildung 3.14). Klicken Sie dann auf **Finish**.

»jcoDestination«-Datei anlegen

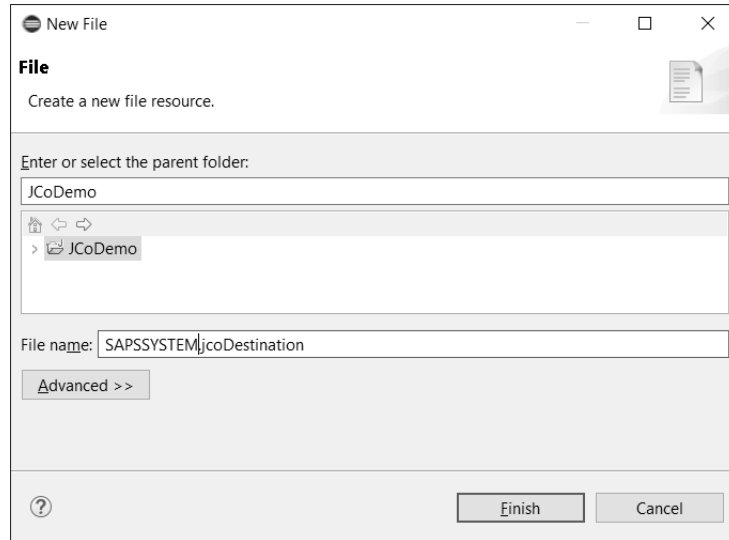


Abbildung 3.14 Verbindungsinformationsdatei anlegen

Öffnen Sie nun diese Datei, und fügen Sie den Quellcode aus Listing 3.7 ein.

```
jco.client.client=010
jco.client.sysnr=10
jco.client.ashost=SAPServer
jco.client.lang=de
jco.client.user=Benutzer
jco.client.passwd=Kennwort
jco.destination.peak_limit=10
jco.destination.pool_capacity=5
```

Listing 3.7 Inhalt der Datei »SAPSYSTEM.jcoDestination«

Beim Speichern wird das Kennwort zum Zugriff auf das SAP-System automatisch nach dem Message Digest Algorithm 5 (MD5) codiert, was zumindest einen gewissen Schutz bietet. Die Eigenschaft `peak_limit` definiert die maximale Anzahl der gleichzeitig verwendeten Verbindungen. Die Eigenschaft `pool_size` definiert die Poolgröße, also die gleichzeitig geöffneten Verbindungen

Klasse zur
Herstellung der
Verbindung

Legen Sie nun eine Klasse an, in der die Funktionen zum Herstellen der Verbindung implementiert werden. In unserem Beispiel nennen wir diese Klasse »Connection«. Den Quellcode dieser Klasse finden Sie in Listing 3.8.

```
package de.sappress.demo;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;

public class Connection
{
    static String SAPSYSTEM = "SAPSYSTEM";
    public static void ConnectionUsingPool()
        throws JCoException
    {
        JCoDestination destination =
            JCoDestinationManager.getDestination(SAPSYSTEM);
        destination.ping();
        System.out.println();
        System.out.println("Attributes:");
        System.out.println(destination.getAttributes());
        System.out.println();
    }
}
```

Listing 3.8 Klasse »Connection« mit der Funktion »ConnectionUsingPool«

Im Hauptprogramm, der im vorangehenden Abschnitt angelegten Klasse Program, können Sie nun die Verbindungsherstellung öffnen (`ConnectionUsingPool()`, siehe Listing 3.9). In der Konsole sollten Ihnen daraufhin alle Eigenschaften (Attribute) der Verbindung ausgegeben werden.

Verbindungs-
herstellung öffnen

```
package de.sappress.demo;
import com.sap.conn.jco.JCoException;
public class Program {
    public static void main(String[] args)
        throws JCoException
    {
        Connection.ConnectionUsingPool();
    }
}
```

Listing 3.9 Verbindungsherstellung im Hauptprogramm

War dieser Test erfolgreich, können Sie den vollständigen Funktionsaufruf des BAPI, wie in Listing 3.10 gezeigt, implementieren.

Funktionsaufruf
des BAPI

```

package de.sappress.demo;
import com.sap.conn.jco.JCoDestination;
import com.sap.conn.jco.JCoDestinationManager;
import com.sap.conn.jco.JCoException;
import com.sap.conn.jco.JCoFunction;
import com.sap.conn.jco.JCoTable;

public class Connection {
    static String SAPSYSTEM = "SAPSYSTEM";
    public static void FunctionCall(String customerID)
        throws JCoException
    {
        JCoDestination destination =
            JCoDestinationManager.getDestination(SAPSYSTEM);
        JCoFunction customerList =
            destination.getRepository().getFunction(
                "BAPI_CUSTOMER_GETLIST");
        JCoTable idRange =
            customerList.getTableParameterList().getTable("IDRANGE");
        idRange.addRow();
        idRange.setValue("SIGN", "I");
        idRange.setValue("OPTION", "EQ");
        idRange.setValue("LOW", customerID);

        customerList.execute(destination);
        JCoTable addressData =
            customerList.getTableParameterList().getTable("ADDRESSDATA");
        addressData.firstRow();
        System.out.println(addressData.getString("CUSTOMER"));
        System.out.println(addressData.getString("NAME"));
        System.out.println(addressData.getString("STREET"));
        System.out.println(addressData.getString("CITY"));
    }
}

```

Listing 3.10 Die Java-Klasse »Connection« implementieren

Hauptprogramm Für das Hauptprogramm fügen Sie den Quellcode aus Listing 3.11 ein.

```

package de.sappress.demo;
import com.sap.conn.jco.JCoException;
public class Program {
    public static void main(String[] args)
        throws JCoException {

```

```

        Connection.FunctionCall(args[0]);
    }
}

```

Listing 3.11 Hauptprogramm: Java-Klasse »Program«

Den Aufruf des Programms müssen Sie nun noch mit einem Parameter ergänzen. Dazu gehen Sie zu den Eigenschaften (**Properties**) Ihres Java-Projekts. Im Bereich **Run/Debug Settings** wählen Sie die Registerkarte **Arguments** aus und ergänzen im Textfeld **Program Arguments** eine (gültige) Kundennummer.

Jetzt kann das Programm getestet werden. Abbildung 3.15 zeigt die Ausgabe in der Konsole. Auch in diesem Beispiel haben wir zur Vereinfachung wieder auf das Abfangen von Fehlern verzichtet.

Programm testen

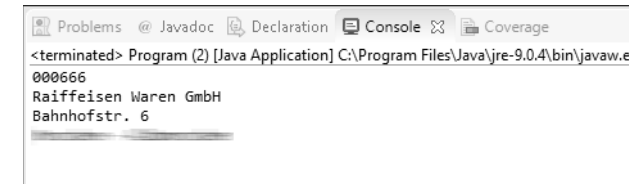


Abbildung 3.15 Erfolgreicher Test der Konsolenanwendung

Wenn Sie den Quellcode des gerade erstellten Java-Programms mit dem des C#-Programms vergleichen (siehe Listing 3.4 in Abschnitt 3.2.3, »Aufruf des BAPI implementieren«), werden Sie eine große Ähnlichkeit feststellen. Wir können also festhalten, dass beide Konnektoren sehr ähnlich arbeiten und sich auf einem sehr ähnlichen Niveau befinden. Je nach Umgebung – Java oder C# – können Sie den passenden Konnektor auswählen.

3.4 Zusammenfassung

SAP-Konnektoren zu verwenden, um aus einer .NET- oder Java-Entwicklung Funktionen in einem SAP-System aufzurufen (oder umgekehrt), ist die einfachste und direkteste Art der Kommunikation mit einem SAP-System. Da Sie selbst Funktionen in ABAP schreiben und diese per RFC remote erreichbar machen können, stehen Ihnen damit im Prinzip alle Möglichkeiten zur Verfügung, um auch komplexe Geschäftsprozesse durch eine Nicht-SAP-Applikation, z. B. eine Windows-Applikation, abzubilden. Die Systeme sind anschließend eng verbunden (im Vergleich z. B. zur Anbindung per Webservice, die wir in Kapitel 4, »Integration über SOAP-Webservices«, erläutern).

Vorteile von SAP-Konnektoren

Nachteile Der Preis dieser Flexibilität ist, dass Sie die gesamte Kommunikation auch selbst kontrollieren müssen. Dabei sollten Sie auch alle erdenklichen Fehlerfälle abhandeln. Gibt es Änderungen im SAP-Programmcode, beispielsweise nach einem Release-Wechsel, müssen Sie eventuell Ihre selbst geschriebenen Funktionen und gegebenenfalls auch Ihren .NET- oder Java-Code anpassen.



Weiterführende Informationen zu den SAP-Konnektoren

Auf den folgenden SAP-Seiten finden Sie weitere Informationen zu den beiden in diesem Kapitel beschriebenen Konnektoren und zu **librfc32.dll**:

- SAP .NET Connector:
<http://s-prs.de/v619022>
- SAP Java Connector:
<http://s-prs.de/v619023>
- SAP NetWeaver RFC Lib 32 / RFC SDK:
<http://s-prs.de/v619024>

Alternativen von Drittanbietern Speziell für die Anbindung einer Applikation im .NET-Umfeld gibt es mit Theobald ERPConnect einen alternativen Konnektor, der Ihnen im Vergleich zum SAP .NET Connector bereits einige Dinge abnimmt. So lässt sich die Verbindung zu einem SAP-System etwas leichter implementieren. Theobald ERPConnect stellen wir deswegen in Kapitel 6, »Verwendung von Drittanbieter-Add-ons«, etwas genauer vor.

Anbindung über einen Proxy Neben Windows-Forms-, Konsolen- oder Webanwendungen ließe sich beispielsweise in C# auch eine Art Proxy implementieren, der als Schnittstelle zwischen dem SAP-System und einer JavaScript-Anwendung agiert und dieser alle Daten im JSON-Format zur Verfügung stellt. Damit könnte die eigentliche Applikation in einem JavaScript-Framework (z. B. jQuery, Sencha oder AngularJS) entwickelt werden. Solche Applikationen haben den Vorteil, web-, betriebssystem- und sogar geräteunabhängig zu laufen (z. B. auch auf mobilen Endgeräten). Anpassungen, die durch eine Änderung innerhalb des SAP-Systems notwendig werden, müssten dann »nur« im Proxy vorgenommen werden. Der Proxy wäre hier eine Windows-Server-Komponente.